

# MusicFormats maintainance guide

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

0.9.66 – August 28, 2022

Jacques Menu

This document presents the internal working of MusicFormats. It is part of the MusicFormats documentation, to be found at <https://github.com/jacques-menu/musicformats/tree/dev/documentation>.

```
1 void msrSegment::browseData (basevisitor* v)
2 {
3     // ... ..
4
5     for (
6         std::list<S_msrMeasure>::const_iterator i = fSegmentElementsList.begin ();
7         i != fSegmentElementsList.end ();
8         ++i
9     ) {
10         // browse the element
11         msrBrowser<msrMeasure> browser (v);
12         browser.browse (*(i));
13     } // for
14
15     // ... ..
16 }
```

```
1 void msr2msrTranslator::visitStart (S_msrClef& elt)
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalMsrOahGroup->getTraceMsrvVisitors ()) {
5             gLogStream <<
6                 "--> Start visiting msrClef" <<
7                 ", line " << elt->getInputLineNumber () <<
8                 std::endl;
9         }
10    #endif
11
12    fCurrentVoiceClone->
13        appendClefToVoice (elt);
14 }
```

# List of Figures

1.1	<a href="#">The MusicFormats architecture</a>	8
16.1	<a href="#">The OAH classes hierarchy</a>	120
19.1	<a href="#">The solo rests problem.</a>	168
19.2	<a href="#">Humdrum-Scot keys</a>	180
19.3	<a href="#">The MSR classes hierarchy.</a>	182
19.4	<a href="#">Three segments in a voice</a>	194
30.1	<a href="#">Zoltán Kodály's Mikrokosmos III Wandering</a>	233
30.2	<a href="#">The LilyPondIssue34 score.</a>	234
68.1	<a href="#">Braille for HelloWorld.xml with interpretation</a>	382
68.2	<a href="#">Un Petit Air, par Jean Abou-Samra</a>	385

# Listings

14.1	<code>countnotes.cpp</code> . . . . .	89
14.2	<code>Visiting &lt;scaling /&gt;</code> . . . . .	90
14.3	<code>msrDoubleTremolo::browseData (basevisitor* v)</code> . . . . .	91

# Contents

<b>I</b>	<b>Overview of MusicFormats</b>	<b>4</b>
1	Introduction . . . . .	5
1.1	Acknowledgements . . . . .	5
1.2	Prerequisites . . . . .	5
1.3	Chronology . . . . .	6
1.4	Zsh vs Bash. . . . .	7
1.5	The GitHub repository . . . . .	7
2	Documentation. . . . .	9
2.1	L <sup>A</sup> T <sub>E</sub> X macros . . . . .	11
2.2	About this document . . . . .	13
2.3	The MusicFormats architecture . . . . .	13
2.4	User guide . . . . .	13
2.5	API guide . . . . .	13
2.6	Maintainance guide . . . . .	13
3	Building MusicFormats . . . . .	14
3.1	Cloning the repository. . . . .	14
3.2	One-shot partial <code>make</code> runs . . . . .	14
3.3	<code>cmake</code> configuration . . . . .	15
3.4	Library <code>Makefile</code> . . . . .	15
3.5	CLI amplies <code>Makefile</code> . . . . .	15
3.6	Building MusicFormats in practise . . . . .	15
4	Code base structure . . . . .	17
4.1	The <code>libmusicxml</code> folder . . . . .	17
4.1.1	Embedding <code>libmusicxml</code> in MusicFormats . . . . .	18
4.1.2	Upgrading the supported MusicXML version . . . . .	18
4.2	The <code>doc</code> folder . . . . .	20
4.3	The <code>schemas</code> folder . . . . .	21
4.4	The <code>src</code> folder . . . . .	21
4.5	The <code>validation</code> folder . . . . .	25
5	Components . . . . .	26
5.1	Components terminology . . . . .	26
5.2	Formats . . . . .	28
5.3	Representations . . . . .	28
5.4	Passes . . . . .	29
5.5	Generators . . . . .	30
5.6	Converters . . . . .	31
5.7	Running a service . . . . .	31
6	Command line samples . . . . .	35
7	Warning and errors (WAE) . . . . .	36
8	The trace facility. . . . .	37
8.1	Activating the trace . . . . .	37
8.2	Trace categories . . . . .	38
8.3	Using traces in practise . . . . .	38
8.4	Debugging traces handling. . . . .	38

9	Multi-lingual support . . . . .	39
10	Textual input and output . . . . .	40
10.1	Indented output streams . . . . .	40
10.2	Creating indented output streams . . . . .	41
10.3	Indenting the output . . . . .	42
10.4	Printing descriptions . . . . .	43
11	Binary data output . . . . .	46
12	CPU measurements . . . . .	47
<b>II</b>	<b>Programming style and conventions</b>	<b>49</b>
13	Programming style and conventions . . . . .	50
13.1	Files naming conventions . . . . .	50
13.2	Adding C++ files . . . . .	52
13.3	Renaming C++ files . . . . .	53
13.4	Source code layout . . . . .	53
13.5	Defensive programming . . . . .	54
13.6	Sanity checks . . . . .	55
13.7	JMI comments . . . . .	55
13.8	Exported symbols . . . . .	56
13.9	Smart pointers . . . . .	56
13.10	Files contents layout . . . . .	58
13.11	<code>#define DEBUG*</code> code sections . . . . .	59
13.12	Identifiers choice conventions . . . . .	59
13.13	Exceptions and warnings/errors reporting . . . . .	61
13.14	Exporting symbols for Windows DLLs . . . . .	63
13.15	Dynamic type checking . . . . .	64
13.16	Input line numbers . . . . .	64
13.17	Static declarations . . . . .	65
13.18	Avoiding MusicFormats multiple initializations . . . . .	65
13.19	Enumeration types . . . . .	66
13.20	yes/no enumerations types . . . . .	68
13.21	Boolean values anyway . . . . .	69
13.22	On/off values . . . . .	69
13.23	Iterating over numeration types . . . . .	70
13.24	Rational numbers . . . . .	72
13.25	Default values . . . . .	73
13.26	<code>create*</code> methods . . . . .	73
13.27	<code>get*()</code> , <code>set*()</code> and <code>fetch*()</code> methods . . . . .	74
13.28	<code>initialize*()</code> and <code>finalize*()</code> methods . . . . .	75
13.29	<code>*asString()</code> and <code>*fromString()</code> functions . . . . .	76
13.30	<code>translate*()</code> methods and <code>convert*()</code> functions . . . . .	77
13.31	context arguments . . . . .	78
13.32	Sorting and <code>compare*()</code> methods . . . . .	79
13.33	Mutually dependent classes . . . . .	80
13.33.1	Pre-declarations . . . . .	80
13.33.2	Simple mutual dependency using separate header files . . . . .	81
13.33.3	More complex mutual dependencies . . . . .	83
13.34	Templates and functional programming usage . . . . .	84

<b>III</b>	<b>The two-phase visitors pattern</b>	<b>86</b>
14	The two-phase visitors pattern . . . . .	87
14.1	Basic mechanism . . . . .	87
14.2	Browser template classes . . . . .	88
14.3	A first example: counting notes in MusicXML data . . . . .	89
14.4	A more complex example . . . . .	89
14.5	Data browsing order. . . . .	91
14.6	Selectively inhibiting data browsing . . . . .	91
14.6.1	Inhibiting data browsing in the code . . . . .	91
14.6.2	Inhibiting data browsing by options. . . . .	93
14.7	Adapting visitors to data browsing order with booleans . . . . .	94
14.8	Adapting visitors to data browsing order with stacks . . . . .	96
14.9	Avoiding the visiting pattern by cascading . . . . .	97
<b>IV</b>	<b>MusicFormats components</b>	<b>98</b>
15	MusicFormats components (MFC) . . . . .	99
15.1	Versions numbers . . . . .	99
15.2	Versions descriptions . . . . .	100
15.3	Versions histories . . . . .	100
15.4	Components descriptions . . . . .	101
15.5	Multi-components . . . . .	102
15.6	Versions history creation . . . . .	104
15.6.1	Representations and passes components creation . . . . .	104
15.6.2	Generators and converters components creation . . . . .	105
15.6.3	MusicFormats library component creation . . . . .	107
15.6.4	Version and history options handling . . . . .	109
15.7	Accessing versions in regular handlers . . . . .	110
15.8	Getting current version numbers . . . . .	111
15.8.1	Current version numbers in options . . . . .	111
15.8.2	Current version numbers in formats . . . . .	113
15.8.3	Current version numbers in passes . . . . .	114
<b>V</b>	<b>Options and help</b>	<b>117</b>
16	Options and help (OAH) . . . . .	118
16.1	OAH basics . . . . .	118
16.2	Features . . . . .	119
16.3	OAH classes inheritance . . . . .	119
16.4	Atoms expecting a value. . . . .	121
16.4.1	The <code>oahAtomStoringAValue</code> class . . . . .	121
16.4.2	The <code>oahBooleanAtom</code> special case. . . . .	122
16.4.3	Checking whether an option has been selected. . . . .	123
16.4.4	The <code>oahAtomStoringAValue</code> sub-classes. . . . .	124
16.5	Pure help atoms . . . . .	125
16.5.1	Pure help atoms without a value . . . . .	125
16.5.2	Pure help atoms expecting a value . . . . .	125

16.6	Options implicitly storing a value . . . . .	126
16.7	Options and help handling. . . . .	127
16.8	Basic OAH types . . . . .	128
16.9	Prefixes handling . . . . .	128
16.10	argc/argv versus oahOptionsVector . . . . .	128
16.11	Applying options . . . . .	129
16.12	A OAH atoms collection. . . . .	130
16.13	An option and help example. . . . .	130
16.14	Options and help introspection . . . . .	131
16.15	Insider versus regular handlers. . . . .	132
16.16	Deciphering the options and arguments . . . . .	133
16.16.1	Options and arguments multi-pass analysis . . . . .	133
16.16.2	Pure help runs . . . . .	135
16.16.3	Applying options. . . . .	135
16.16.4	Early handling of some options . . . . .	137
16.17	Implementing the -find option . . . . .	139
16.18	Checking options consistency . . . . .	141
16.19	Adding new options . . . . .	141
16.19.1	Representations' vs. passes' options. . . . .	142
16.19.2	Using an existing OAH atom class . . . . .	142
16.19.3	Creating a new OAH atom class without a value . . . . .	144
16.19.4	Creating a new OAH atom class EXpecting a value . . . . .	146
16.20	Extra options . . . . .	149
16.21	man pages generation . . . . .	150
16.22	Specific global OAH groups . . . . .	150
16.23	Visiting OAH groups . . . . .	151

## VI Representations 152

17	Representations general principles. . . . .	153
17.1	Trees vs graphs, denormalization. . . . .	153
17.2	Denormalization . . . . .	153
17.2.1	Descriptions sharing . . . . .	153
17.2.2	Multiple voices. . . . .	154
17.3	Newborn clones . . . . .	154
17.4	Deep clones . . . . .	156
17.5	Inheritance . . . . .	156
17.5.1	Single inheritance . . . . .	156
17.5.2	Single inheritance for smart pointers . . . . .	157
17.5.3	Multiple inheritance for visitors. . . . .	158
17.5.4	Multiple inheritance in other classes . . . . .	160
17.5.5	Reversibility . . . . .	161
18	Displaying formats . . . . .	162
18.1	Display categories . . . . .	162
18.2	Displaying in practise . . . . .	162

19	Music Scores Representation (MSR).	163
19.1	MSR basic types	164
19.2	Data matching across formats	165
19.3	Lengths.	166
19.4	Sounding and displayed durations	166
19.5	Measure positions and moments	167
19.6	Rests and skips	167
19.7	Solo notes and rests	168
19.8	Linear versus time-oriented representation	168
19.9	Spanners	169
19.10	Uplinks, direct uplinks and sidelinks	169
19.11	Printing descriptions	170
19.12	Pitches	171
19.13	Octaves	173
19.14	Durations.	174
19.15	Alterations	174
19.16	Accidentals	174
19.17	Durations.	175
19.18	Tempos	175
19.18.1	Tempos notes	176
19.18.2	Tempos tuplets	176
19.18.3	Tempos description	178
19.19	Clefs	179
19.20	Keys	179
19.21	Time signatures	180
19.22	MSR classes inheritance	181
19.23	Books	184
19.24	Scores	184
19.25	Part groups	184
19.26	Parts	185
19.27	Staves	185
19.28	Voice elements	185
19.29	Voices	186
19.30	Measures	187
19.31	Repeats patterns and replicas	189
19.32	Beat repeats	189
19.33	Measure repeats.	191
19.34	Multiple full-bar rests	192
19.35	Barlines	192
19.36	Repeats.	192
19.37	Segments	194
19.38	Notes and rests	194
19.39	Grace notes groups	195
19.40	Chords	195
19.41	Tuplets	196
19.42	Harmonies and figured bass similarities.	197
19.43	Harmonies	198
19.44	Figured bass	198
19.45	Lyrics	198
19.46	MIDI.	199



20	MSR time-oriented representation . . . . .	200
20.1	Note events . . . . .	200
20.2	Simultaneous notes chunks . . . . .	201
20.3	Measures slices . . . . .	201
20.4	Measures slices sequences . . . . .	201
20.5	Building the measures slices . . . . .	202
20.5.1	Part measures slices . . . . .	202
20.5.2	Staff measures slices . . . . .	203
20.6	Solo notes and rests . . . . .	203
20.7	A measures slices example . . . . .	204
21	Path to voice . . . . .	205
22	LilyPond Scores Representation (LPSR) . . . . .	206
22.1	LPSR basic types . . . . .	206
22.2	Adapting LilyPond code generation to the target version number . . . . .	207
23	Braille Scores Representation (BSR) . . . . .	208
23.1	BSR basic types . . . . .	208
23.2	Representing cells . . . . .	208
24	MusicXML Scores Representation (MXSR) . . . . .	211
24.1	MusicXML elements and attributes . . . . .	211
24.2	The <code>xmlelement</code> and <code>xmlattribute</code> types . . . . .	212
24.3	Enumeration types for <code>xmlelement</code> 's <code>fType</code> . . . . .	213
24.4	Classes for the <code>xmlements</code> . . . . .	214
24.5	<code>xmlelement</code> trees . . . . .	215
24.5.1	Creating <code>xmlelement</code> trees from textual data . . . . .	216
24.5.2	Printing <code>xmlelement</code> trees . . . . .	216
24.6	The <code>SXMLFile</code> type . . . . .	216
<b>VII</b>	<b>Passes</b>	<b>218</b>
25	The passes . . . . .	219
25.1	Translating MusicXML data to an MXSR format . . . . .	219
25.1.1	MusicXML coverage . . . . .	219
25.2	Translating an MXSR to an MSR . . . . .	219
25.3	Translating an MSR to an MXSR . . . . .	219
25.4	Translating an MSR to another MSR . . . . .	219
25.5	Translating an MSR to an LPSR . . . . .	219
25.6	Translating an LPSR to LilyPond code . . . . .	220
25.7	Translating an MSR to an BSR . . . . .	220
25.8	Translating a BSR to another BSR . . . . .	220
25.9	Translating an MXSR to Guido . . . . .	220
26	LilyPond code generation . . . . .	221
26.1	Basic principle . . . . .	221
26.2	Generating Scheme functions in the LilyPond output . . . . .	222
27	Braille generation . . . . .	223
27.1	Basic principle . . . . .	223
27.2	Output files name and contents options . . . . .	224
27.3	Braille generators . . . . .	224
27.4	Writing braille cells . . . . .	225
28	MusicXML generation . . . . .	227
28.1	Basic principle . . . . .	227
28.2	Creating an <code>xmlelement</code> . . . . .	227
28.3	Creating an <code>xmlelement</code> tree . . . . .	228
28.4	Browsing the visited MSR score . . . . .	229
28.5	Ancillary functions to create MXSR data . . . . .	230
29	Guido code generation . . . . .	231
29.1	Basic principle . . . . .	231

<b>VIII</b>	<b>Generators</b>	<b>232</b>
30	The generators . . . . .	233
30.1	MusicAndHarmonies . . . . .	233
30.2	Mikrokosmos3Wandering . . . . .	233
30.3	LilyPondIssue34 . . . . .	234
<b>IX</b>	<b>Converters</b>	<b>235</b>
31	The converters . . . . .	236
31.1	xml2ly . . . . .	236
31.2	xml2brl . . . . .	236
31.3	xml2xml . . . . .	237
31.4	xml2gmn . . . . .	237
31.5	msdlconverter . . . . .	237
<b>X</b>	<b>Interfaces</b>	<b>238</b>
32	Library interfaces . . . . .	239
33	Representations interfaces . . . . .	240
33.1	MSR interfaces . . . . .	240
33.2	LPSR interfaces . . . . .	240
33.3	MSDL interfaces . . . . .	240
34	Passes interfaces . . . . .	241
34.1	Translating MusicXML data to an MXSR . . . . .	242
34.2	Translating an MXSR to an MSR . . . . .	242
34.3	Translating an MSR to an MXSR . . . . .	242
34.4	Translating an MSR to another MSR . . . . .	242
34.5	Translating an MSR to an LPSR . . . . .	242
34.6	Translating an LPSR to LilyPond code . . . . .	242
34.7	Translating an MSR to an BSR . . . . .	242
34.8	Translating a BSR to another BSR . . . . .	242
34.9	Translating an MXSR to Guido . . . . .	242
35	Converters interfaces . . . . .	243
<b>XI</b>	<b>Distributions and versions</b>	<b>245</b>
36	MusicFormats distributions . . . . .	246
36.1	GitHub actions . . . . .	246
36.1.1	Creating the distributions . . . . .	251
36.1.2	Security issue in recent MacOS™ versions . . . . .	252
37	MusicFormats branches and versions . . . . .	254
<b>XII</b>	<b>Selected topics</b>	<b>260</b>
38	Initializations . . . . .	261
38.1	Options and help initializations . . . . .	261
38.2	Representations initializations . . . . .	262
38.2.1	MSR initialization . . . . .	262
38.2.2	LPSR initialization . . . . .	263
38.2.3	BSR initialization . . . . .	263
38.3	Passes initializations . . . . .	263
38.4	Converters initializations . . . . .	263
39	The OAH atoms collection . . . . .	266
39.1	OAH macro atoms . . . . .	266
39.2	A OAH macro atom example . . . . .	268
39.3	LilyPond octave entry . . . . .	269

40	Measures handling . . . . .	271
40.1	Voices contents . . . . .	271
40.2	Voice elements . . . . .	271
40.3	Measure elements . . . . .	272
40.4	Appending measure elements to a measure . . . . .	272
40.5	Appending measures to a segment . . . . .	274
40.6	Appending measures to a voice . . . . .	276
40.7	Translating from MXSR to MSR . . . . .	278
40.8	Translating from MXSR to MSR . . . . .	279
40.9	Translating from MSR to MSR . . . . .	280
40.10	Translating from MSR to LPSR . . . . .	280
40.11	Translating from LPSR to LilyPond . . . . .	280
41	Finalizations . . . . .	281
41.1	Clones vs non-clones finalization . . . . .	281
41.2	The finalization methods . . . . .	282
41.3	Finalizing parts . . . . .	285
41.4	Finalizing staves . . . . .	286
41.5	Finalizing voices . . . . .	286
41.6	Finalizing repeats . . . . .	288
41.7	Finalizing measures . . . . .	288
41.7.1	Finalizing regular measures . . . . .	289
41.7.2	Finalizing harmonies measures . . . . .	292
41.7.3	Finalizing figured bass measures . . . . .	293
41.8	Determining measure positionss . . . . .	295
42	Tempos handling . . . . .	296
42.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	296
42.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	296
42.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	296
42.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	296
42.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	296
43	Notes handling . . . . .	297
43.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	297
43.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	297
43.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	297
43.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	297
43.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	297
44	Segments handling . . . . .	298
44.1	Segments creation . . . . .	298
44.1.1	Creating a new last segment for a voice . . . . .	299
44.1.2	Creating a new last segment for a voice from its first measure . . . . .	300
44.2	Appending measures to a segment . . . . .	301
44.3	Translating from MXSR to MSR . . . . .	302
44.4	Translating from MXSR to MSR . . . . .	302
44.5	Translating from MSR to MSR . . . . .	302
44.6	Translating from MSR to LPSR . . . . .	302
44.7	Translating from LPSR to LilyPond . . . . .	302
45	Beat repeats handling . . . . .	303
45.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	303
45.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	303
45.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	303
45.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	303
45.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	303

46	Measure repeats handling . . . . .	304
46.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	304
46.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	304
46.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	304
46.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	304
46.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	304
47	Multiple full-bar rests handling . . . . .	305
48	Repeats handling . . . . .	306
48.1	Translating repeats from MXSR to MSR . . . . .	306
48.2	Translating repeats from MXSR to MSR . . . . .	311
48.3	Translating repeats from MSR to MSR . . . . .	311
48.4	Translating repeats from MSR to LPSR . . . . .	311
48.5	Translating repeats from LPSR to LilyPond . . . . .	312
49	Voices handling . . . . .	313
49.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	313
49.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	313
49.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	313
49.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	313
49.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	313
50	Staves handling . . . . .	314
50.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	314
50.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	314
50.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	314
50.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	314
50.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	314
51	Parts handling . . . . .	315
51.1	Parts browsing . . . . .	315
51.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	316
51.3	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	316
51.4	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	316
51.5	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	316
51.6	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	316
52	Part groups handling . . . . .	317
53	Scores handling . . . . .	319
53.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	319
53.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	319
53.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	319
53.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	319
53.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	319
54	Books handling . . . . .	320
54.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	320
54.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	320
54.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	320
54.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	320
54.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	320
55	Ornaments handling . . . . .	321
55.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	321
55.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	321
55.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	321
55.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	321
55.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	321
56	Ties handling . . . . .	322
56.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	322
56.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	322
56.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	322
56.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	322
56.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	322

57	Dynamics handling . . . . .	323
57.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	323
57.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	323
57.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	323
57.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	323
57.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	323
58	Beams handling . . . . .	324
58.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	324
58.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	324
58.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	324
58.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	324
58.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	324
59	Slurs handling . . . . .	325
59.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	325
59.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	325
59.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	325
59.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	325
59.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	325
60	Grace notes groups handling . . . . .	326
60.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	326
60.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	326
60.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	326
60.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	326
60.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	326
61	Chords handling . . . . .	327
61.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	327
61.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	327
61.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	327
61.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	327
61.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	327
62	Tuplets handling . . . . .	328
62.1	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	328
62.2	Translating from MXSR to MSR ( <a href="#">src/passes/mxsr2msr/</a> ) . . . . .	328
62.3	Translating from MSR to MSR ( <a href="#">src/passes/msr2msr/</a> ) . . . . .	328
62.4	Translating from MSR to LPSR ( <a href="#">src/passes/msr2lpsr/</a> ) . . . . .	328
62.5	Translating from LPSR to LilyPond ( <a href="#">src/passes/lpsr2lilypond/</a> ) . . . . .	328
63	Harmonies handling . . . . .	329
63.1	Harmonies staves and voices . . . . .	330
63.2	Harmonies staves creation . . . . .	330
63.3	Translating harmonies from MXSR to MSR . . . . .	331
63.4	Translating harmonies from MXSR to MSR . . . . .	333
63.4.1	First <code>S_harmony</code> visit. . . . .	333
63.4.2	Second <code>S_harmony</code> visit . . . . .	334
63.4.3	Attaching <code>msrHarmony</code> instances to notes . . . . .	335
63.4.4	Populating <code>msrHarmony</code> instances . . . . .	336
63.4.5	First <code>S_harmony</code> visit. . . . .	337
63.4.6	Inserting <code>msrHarmony</code> instances in the part harmonies voice . . . . .	338
63.5	Translating harmonies from MSR to MSR . . . . .	339
63.6	Translating harmonies from MSR to LPSR . . . . .	340
63.7	Translating harmonies from LPSR to LilyPond . . . . .	343

64	Figured bass handling . . . . .	345
64.1	Figured bass description . . . . .	345
64.2	Figured bass staves and voices . . . . .	347
64.3	Figured bass staves creation . . . . .	347
64.4	Translating figured bass from MXSR to MSR . . . . .	348
64.4.1	First <code>S_figured_bass</code> visit . . . . .	350
64.4.2	Second <code>S_figured_bass</code> visit . . . . .	351
64.4.3	Attaching <code>msrFiguredBass</code> instances to notes . . . . .	352
64.4.4	Populating <code>msrFiguredBass</code> instances . . . . .	353
64.4.5	Inserting <code>S_msFiguredBass</code> instances in the part figured bass voice . . . . .	354
64.5	Translating figured bass from MSR to MSR . . . . .	355
64.6	Translating figured bass from MSR to LPSR . . . . .	356
64.7	Translating figured bass from LPSR to LilyPond . . . . .	358
65	Lyrics handling . . . . .	360
65.1	Translating from MXSR to MSR ( <code>src/passes/mxsr2msr/</code> ) . . . . .	360
65.2	Translating from MXSR to MSR ( <code>src/passes/mxsr2msr/</code> ) . . . . .	360
65.3	Translating from MSR to MSR ( <code>src/passes/msr2msr/</code> ) . . . . .	360
65.4	Translating from MSR to LPSR ( <code>src/passes/msr2lpsr/</code> ) . . . . .	360
65.5	Translating from LPSR to LilyPond ( <code>src/passes/lpsr2lilypond/</code> ) . . . . .	360
66	MIDI handling . . . . .	361
66.1	Translating from MXSR to MSR ( <code>src/passes/mxsr2msr/</code> ) . . . . .	361
66.2	Translating from MXSR to MSR ( <code>src/passes/mxsr2msr/</code> ) . . . . .	361
66.3	Translating from MSR to MSR ( <code>src/passes/msr2msr/</code> ) . . . . .	361
66.4	Translating from MSR to LPSR ( <code>src/passes/msr2lpsr/</code> ) . . . . .	361
66.5	Translating from LPSR to LilyPond ( <code>src/passes/lpsr2lilypond/</code> ) . . . . .	361
<b>XIII</b>	<b>MusicFormats Scripting Language (MFSL)</b>	<b>362</b>
67	MFSL (MusicFormats Scripting Language) . . . . .	363
67.1	A script example . . . . .	363
67.2	Implementation principles . . . . .	364
67.3	The contents of the MFSL folder . . . . .	364
67.4	The MFSL basic types . . . . .	365
67.5	The MFSL Makefile . . . . .	365
67.6	Locations handling . . . . .	366
67.7	Tokens description . . . . .	366
67.8	The driver . . . . .	367
67.9	Lexical analysis . . . . .	369
67.9.1	Flex options . . . . .	369
67.9.2	Flex regular expressions . . . . .	370
67.10	Syntax and semantic analysis . . . . .	372
67.10.1	Bison options for MFSL . . . . .	372
67.10.2	The MFSL tokens . . . . .	373
67.10.3	The MFSL non-terminals and axiom . . . . .	373
67.11	Interface to the MFSL parser . . . . .	374
67.12	Running the example MFSL script . . . . .	375
67.12.1	Error recovery . . . . .	376

<b>XIV</b>	<b>Music Scores Description Language (MSDL)</b>	<b>377</b>
68	MSDL (Music Scores Description Language)	378
68.1	Main features of MSDL	378
68.2	MSDL basic types	378
68.3	What the MSDL converter does	379
68.3.1	LilyPond generation	379
68.3.2	Braille generation	379
68.3.3	MusicXML generation	380
68.3.4	Guido generation	380
68.4	A first example	380
68.5	First example output from the MSDL converter	381
68.5.1	LilyPond output	381
68.5.2	Braille output	382
68.5.3	MusicXML output	382
68.5.4	Guido output	383
68.6	A more realistic example	384
68.7	Multi-language support	385
68.7.1	Multi-language messages handling	385
68.7.2	Multi-language keywords handling	385
68.8	Lexical analysis	385
68.9	Music Scores Descriptions Representation (MSDR)	385
68.10	Syntax and semantic analysis	385
68.10.1	Error recovery	385
<b>XV</b>	<b>Debugging</b>	<b>386</b>
69	Debugging	387
69.1	Useful options	387
69.2	Removing the results of a build	388
69.3	Reverting to a previous MusicFormats version	388
70	Locating a bug with Git's bisection	389
70.1	Locating a bug at random in the Git log	390
70.2	Locating a bug in the commits with Git's bisection	391
70.3	Locating the bug in the code base	392
<b>XVI</b>	<b>Indexes</b>	<b>394</b>

## Part I

# Overview of MusicFormats



# Chapter 1

## Introduction

This document presents the design principles and architecture of MusicFormats, as well as information needed to maintain it. It is part of the MusicFormats documentation, to be found at <https://github.com/jacques-menu/musicformats/tree/master/doc>.

All the MusicXML examples mentioned can be downloaded from <https://github.com/jacques-menu/musicformats/tree/master/musicxml>.

They are grouped by subject in subdirectories, such as [basic/HelloWorld.xml](#).

The MSDL examples can be found at <https://github.com/jacques-menu/musicformats/tree/master/msdl>.

### 1.1 Acknowledgements

Many thanks to Dominique Dominique Fober, the designer and maintainer of the `libmusicxml2` library!

### 1.2 Prerequisites

In order to maintain MusicFormats, one needs to do the following:

- obtain a working knowledge of C++ programming. The code base of MusicFormats uses classes, simple inheritance, and templates;
- study the architecture of MusicFormats, which can be seen in figure ?? [Architecture], page ??, and is presented in more detail at:  
<https://github.com/jacques-menu/musicformats/blob/master/doc/musicformatsArchitecture/musicformatsArchitecture.pdf>

In this document, all paths to files are relative to the MusicFormats source code directory.

## 1.3 Chronology

Dominique Fober created `libmusicxml2` long before this author had the need for a library to read MusicXML data, in order to convert it to LilyPond. In the picture showing the architecture of MusicFormats in figure ?? [Architecture], page ??, Dom's work is essentially represented by the MusicXML, MXSR and Guido boxes at the top. He did more than this, of course, to provide `libmusicxml2` to users!

This author's work started with `xml2ly`, initially named `xml2lilypond`, whose goal was to:

- perform as least as well as `musicxml2ly`, provided by LilyPond;
- provide as many options as needed to meet the user's needs.

The `*.cpp` files in `samples` were examples of the use of the library. Among them, `xml2guido` has been used since in various contexts. The diagram in figure ?? [Architecture], page ??, was created afterwards, and it would then have consisted of only MusicXML, MXSR and Guido, with passes 1, 2 and 3.

When tackling the conversion of MusicXML to LilyPond, this author created MSR as the central internal representation for music score. It is meant to capture the musical contents of score in fine-grain detail, to meet the needs of creating LilyPond code first, and Braille later. The only change made to the existing MXSR format has been to add an input line number to `xmlElement`.

The conversion from MSR to BSR music was two-pass from the beginning, first creating a BSRformat with unlimited line and page lengths, and then constraining that in a second BSR would take the numbers of cell per line and lines per page into account. This was frozen in autumn 2019 due to the lack of interest from the numerous persons and bodies that this author contacted about `xml2brl`. The current status is the braille output is that the cells per line and lines per page values are ignored.

The creation of MusicXML code from MSR data was then added to close a loop with MusicXML2xml, with the idea that it would make MusicFormats a kind of swiss knife for textual formats of music scores.

Having implemented a number of computer languages in the past, this author was then tempted to design MSDL, which stands for Music Scores Description Language. The word *description* has been preferred to *programming*, because not all musicians have programming skills. The basic aim of MSDL is to provide a musician-oriented way to describe a score that can be converted to various target textual forms.

`src/clisamples/Mikrokosmos3Wandering.cpp` has been written to check that the MSR API was rich enough to go this way. The API was enriched along the way.

Having MSR, LPSR and BSR available, as well as the capability to generate MusicXML, LilyPond Guido and Braille, made writing a first draft of the MSDL converter, with version number 1.001, rather easy. The initial output target languages were MusicXML, LilyPond, MusicXML and Braille.

This document contains technical information about the internal working of the code added to MusicFormats by this author as their contribution to this great piece of software.

## 1.4 Zsh vs Bash

Although the shell mentioned in the MusicFormats user guide is Bash, most scripts and functions supplied for MusicFormats maintainance are Zsh scripts. This is because of the magical globbing pattern qualifier `**/` Zsh supplies, which makes `find` seldom needed.

For example, in `scripts/ZshDefinitionsForMusicFormats.zsh`, adding the `include/` folder alongside `src/` is done this way:

```
1 function addInclude ()
2 {
3     set -x
4     echo "--> INCLUDE_DIR: ${INCLUDE_DIR}"
5
6     rm -rf ${INCLUDE_DIR}
7     mkdir -p ${INCLUDE_DIR}
8
9     cd ${SRC_DIR}
10
11     rsync -R **/*.h ${INCLUDE_DIR_NAME}
12
13     mv ${INCLUDE_DIR_NAME} ..
14
15     git add ../${INCLUDE_DIR_NAME}/*
16     set +x
17 }
```

This creates the same folders hierarchy as the one in `src/`, with only the `*.h` header files in it.

## 1.5 The GitHub repository

The GitHub repository, hosted at <https://github.com/jacques-menu/musicformats>, contains two branches types:

- the default `master` 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.

Figure 1.1: The MusicFormats architecture



## Chapter 2

# Documentation

The MusicFormats documentation is written in L<sup>A</sup>T<sub>E</sub>X, the pictures being created with the TikZ/PGF package, see <https://www.bu.edu/math/2013/08/tikzpgfmanual.pdf>.

All the L<sup>A</sup>T<sub>E</sub>X files have an initial '`! TEX root`' comment. This is a TeXShop specific feature, allowing a documentation to be composed from any of the files that it imports, such as:

```
1 % !TEX root = MusicFormatsMaintenanceGuide.tex
```

The documentation/ directory contains:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/documentation > ls -sal
2 total 48
3  0 drwxr-xr-x@ 10 jacquesmenu  staff      320 Feb 28 07:42 .
4  0 drwxr-xr-x  38 jacquesmenu  staff     1216 Feb 27 12:14 ..
5 48 -rw-r--r--@  1 jacquesmenu  staff    22532 Feb 28 07:49 .DS_Store
6  0 drwxr-xr-x  18 jacquesmenu  staff      576 Feb 28 08:27 CommonLaTeXFiles
7  0 drwxr-xr-x  38 jacquesmenu  staff     1216 Feb 18 08:39 IntroductionToMusicXML
8  0 drwxr-xr-x  57 jacquesmenu  staff     1824 Feb 28 10:08 MusicFormatsAPIGuide
9  0 drwxr-xr-x 116 jacquesmenu  staff     3712 Feb 28 10:22 MusicFormatsMaintenanceGuide
10 0 drwxr-xr-x  53 jacquesmenu  staff     1696 Feb 28 10:07 MusicFormatsUserGuide
11 0 drwxr-xr-x  27 jacquesmenu  staff      864 Feb 14 08:54 graphics
12 0 drwxr-xr-x   5 jacquesmenu  staff      160 Jan 23 16:33 presentation
```

The CommonLaTeXFiles/ directory contains L<sup>A</sup>T<sub>E</sub>X settings used by the various documentation files and the code for pictures:

```
1 jacquesmenu@macmini:~/musicformats-git-dev/documentation/CommonLaTeXFiles > ls -sal *.tex
2  8 -rwxr--r--@  1 jacquesmenu  staff      241 Aug 17 14:21 CreateMSRScoreRepresentation.zsh
3  8 -rw-r--r--   1 jacquesmenu  staff      507 Jun 28 20:52 LaTeXBoxes.tex
4 16 -rw-r--r--   1 jacquesmenu  staff    6494 Jun 28 20:52 LaTeXCommonSettings.tex
5  8 -rw-r--r--   1 jacquesmenu  staff    1907 Jun 28 20:52 LaTeXDivisionsCommands.tex
6  8 -rw-r--r--   1 jacquesmenu  staff      957 Jun 28 20:52 LaTeXFontsAndColors.tex
7  8 -rw-r--r--   1 jacquesmenu  staff      604 Jun 28 20:52 LaTeXGraphicsAndPictures.tex
8  8 -rw-r--r--   1 jacquesmenu  staff    1128 Jun 28 20:52 LaTeXIndexing.tex
9 24 -rw-r--r--   1 jacquesmenu  staff    10728 Jun 28 20:52 LaTeXListings.tex
10  8 -rw-r--r--   1 jacquesmenu  staff     1527 Jun 28 20:52 LaTeXMusicFormatsCommands.tex
11 24 -rw-r--r--   1 jacquesmenu  staff    11735 Aug 14 16:50 LaTeXMusicFormatsFilesAndFolders.
    tex
12  8 -rw-r--r--@  1 jacquesmenu  staff    2151 Jun 28 20:52 LaTeXMusicFormatsNames.tex
13  8 -rw-r--r--   1 jacquesmenu  staff      441 Jun 28 20:52 LaTeXMusicNotation.tex
14  8 -rw-r--r--   1 jacquesmenu  staff    1535 Jun 28 20:52 LaTeXReferencing.tex
15 32 -rw-r--r--   1 jacquesmenu  staff   15145 Aug 14 16:50 LaTeXShortcuts.tex
16  8 -rw-r--r--   1 jacquesmenu  staff    2171 Jun 28 20:52 LaTeXTablesAndLists.tex
17 40 -rw-r--r--@  1 jacquesmenu  staff   17746 Jun 28 20:52 MSRClassesHierarchyPicture.tex
18 40 -rw-r--r--@  1 jacquesmenu  staff   16711 Aug 17 14:21 MSRScoreRepresentation.tex
19  8 -rw-r--r--   1 jacquesmenu  staff      321 Jun 28 20:52 MusicFormats.ist
```

```

20 48 -rw-r--r-- 1 jacquesmenu staff 21749 Jun 28 20:52 MusicFormatsArchitecturePicture.
    tex
21 8 -rw-r--r-- 1 jacquesmenu staff 1558 Aug 15 23:36 OAHClassesHierarchyPicture.log
22 16 -rw-r--r--@ 1 jacquesmenu staff 5906 Aug 16 08:02 OAHClassesHierarchyPicture.tex

```

It is to be noted that `documentation/CommonLaTeXFiles/MSRScoreRepresentation.tex` is generated from `documentation/CommonLaTeXFiles/MSRClassesHierarchyPicture.tex` with this script, which should be run only if there is any change in the MSR classes hierarchy:

```

1 jacquesmenu@macmini-1:~/musicformats-git-dev/documentation/CommonLaTeXFiles > cat
    CreateMSRScoreRepresentation.zsh
2 #!/bin/zsh
3
4 # create a LaTeX file for the user guide from the maintainance one
5
6 sed 's/msr//g' MSRClassesHierarchyPicture.tex \
7 | \
8 sed 's/The MSR classes hierarchy/The MSR music score representation/g' \
9 > \
10 MSRScoreRepresentation.tex

```

Directory `graphics/` contains PNG files showing screenshots of the results of using the MusicFormats tools.

Directory `libmusicxml2Presentation/` contains a presentation of `libmusicxml2` written by Dominique Fober.

Directory `IntroductionToMusicxml/` contains a presentation done by this author at the 'MUSIC ENGRAVING IN THE 21ST CENTURY – DEVELOPMENTS AND PERSPECTIVES' conference at Mozarteum in Salzburg in January 2020 (<https://www.uni-mozarteum.at/en/kunst/music-engraving-conference.php>).

$\text{\LaTeX}$  needs to be run *three* times when the chapter/section/subsection hierarchy is modified. Check that the last page number, at the bottom of any page, is not less than the one before.

The following files contain the current MusicFormats version number and date:

- the `src/MusicFormatsVersionNumber.h` and `src/MusicFormatsVersionDate.h` files are used by the C++ code base;
- file `MusicFormatsVersionNumber.txt` and file `MusicFormatsVersionDate.txt` are used by the  $\text{\LaTeX}$  source files

Those files should be re-generated when a new version of MusicFormats is created, for example:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > scripts/SetMusicFormatsVersionNumber.bash "
    0.9.61"
2 ==> PWD is:
3 /Users/jacquesmenu/musicformats-git-dev
4
5 ==> Writing MusicFormats version number 0.9.61 to MusicFormatsVersionNumber.txt
6
7 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 2 13:43:04 2022 MusicFormatsVersionNumber.txt
8 0.9.61
9 ==> PWD is:
10 /Users/jacquesmenu/musicformats-git-dev/src
11
12 ==> Writing MusicFormats version number 0.9.61 to MusicFormatsVersionNumber.h
13
14 8 -rw-r--r--@ 1 jacquesmenu staff 45 Mar 2 13:43:04 2022 MusicFormatsVersionNumber.h
15 #define MUSICFORMATS_VERSION_NUMBER "0.9.61"

```

and:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > scripts/SetMusicFormatsVersionDate.bash "
  March 2, 2022"
2 ==> PWD is:
3 /Users/jacquesmenu/musicformats-git-dev
4
5 ==> Writing MusicFormats version date March 2, 2022 to MusicFormatsVersionDate.txt
6
7 8 -rw-r--r--@ 1 jacquesmenu staff 14 Mar 2 13:43:32 2022 MusicFormatsVersionDate.txt
8 March 2, 2022
9
10 ==> PWD is:
11 /Users/jacquesmenu/musicformats-git-dev/src
12
13 ==> Writing MusicFormats version date March 2, 2022 to MusicFormatsVersionDate.h
14
15 8 -rw-r--r--@ 1 jacquesmenu staff 50 Mar 2 13:43:32 2022 MusicFormatsVersionDate.h
16 #define MUSICFORMATS_VERSION_DATE "March 2, 2022"

```

Avoid editing these files manually. In particular, `MusicFormatsVersionNumber.txt` should **NOT** be terminated by an end of line, since its contents is used in the name of library files generated in `build/lib`.

## 2.1 L<sup>A</sup>T<sub>E</sub>X macros

The MusicFormats documentation uses a number of macros both to simplify formatting of frequent texts and to feed the many indexes at the end. All of them are grouped in `documentation/CommonLaTeXFiles`:

```

1 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/documentation/CommonLaTeXFiles >
  ls -sal LaTeX*.tex
2 8 -rw-r--r-- 1 jacquesmenu staff 507 Jun 28 20:52 LaTeXBoxes.tex
3 16 -rw-r--r-- 1 jacquesmenu staff 6494 Jun 28 20:52 LaTeXCommonSettings.tex
4 8 -rw-r--r-- 1 jacquesmenu staff 1907 Jun 28 20:52 LaTeXDivisionsCommands.tex
5 8 -rw-r--r-- 1 jacquesmenu staff 957 Jun 28 20:52 LaTeXFontsAndColors.tex
6 8 -rw-r--r-- 1 jacquesmenu staff 604 Jun 28 20:52 LaTeXGraphicsAndPictures.tex
7 8 -rw-r--r-- 1 jacquesmenu staff 1128 Jun 28 20:52 LaTeXIndexing.tex
8 24 -rw-r--r-- 1 jacquesmenu staff 10728 Jun 28 20:52 LaTeXListings.tex
9 8 -rw-r--r-- 1 jacquesmenu staff 1527 Jun 28 20:52 LaTeXMusicFormatsCommands.tex
10 24 -rw-r--r--@ 1 jacquesmenu staff 11735 Jul 29 09:02 LaTeXMusicFormatsFilesAndFolders.
    tex
11 8 -rw-r--r--@ 1 jacquesmenu staff 2151 Jun 28 20:52 LaTeXMusicFormatsNames.tex
12 8 -rw-r--r-- 1 jacquesmenu staff 441 Jun 28 20:52 LaTeXMusicNotation.tex
13 8 -rw-r--r-- 1 jacquesmenu staff 1535 Jun 28 20:52 LaTeXReferencing.tex
14 32 -rw-r--r--@ 1 jacquesmenu staff 14665 Jun 28 20:52 LaTeXShortcuts.tex
15 8 -rw-r--r-- 1 jacquesmenu staff 2171 Jun 28 20:52 LaTeXTablesAndLists.tex

```

For example:

```

1 \newcommand{\CLI}{command line\index[Main]{command line}}

```

```

1 \newcommand{\musicXmlMarkup}[1]{%
2 {\tt <#1/>}\index[Main]{\tt $<$#1 /$>$}\index[MusicXML]{\tt #1 $<$>$}}%
3 }
4 \newcommand{\musicXmlAttribute}[1]{%
5 {\tt "#1"}\index[Main]{\tt $<$#1 /$>$}\index[MusicXML]{\tt #1 ""}}%
6 }

```

```

1 \newcommand{\Main}[1]{%
2 #1\index[Main]{#1}%
3 }
4 \newcommand{\MainName}[1]{%
5 \index[Main]{#1}%
6 }

```

```

7
8 \newcommand{\code}[1]{%
9 {\tt #1}\index[Main]{\tt #1}}%
10 }

```

Some command exist in two forms, differing in the capitalization of the first character:

```

1 \newcommand{\enumType}{enumeration type\index[Main]{enumeration type}}
2 \newcommand{\EnumType}{Enumeration type\index[Main]{enumeration type}}

```

Some command names are of the form **\*Both\***:

```

1 \newcommand{\fileName}[1]{%
2 {\tt #1}\index[Main]{\tt #1}\index[Files]{\tt #1}}%
3 }
4 \newcommand{\fileNameBoth}[1]{%
5 {\textcolor{brown}{\tt *#1.h/.cpp}}\index[Main]{#1.h/.cpp@\tt *#1.h/.cpp}}\index[Files
6 ]{#1.h/.cpp@\tt *#1.h/.cpp}}%

```

```

1 \newcommand{\msrToMsr}[1]{%
2 {\textcolor{brown}{\tt src/passes/msr2msr/#1}}%
3 }
4 \newcommand{\msrToMsrBoth}[1]{%
5 {\textcolor{brown}{\tt src/passes/msr2msr/#1.h/.cpp}}%
6 }

```

Some command names are of the form **star\***:

```

1 \newcommand{\methodName}[1]{%
2 {\tt #1~()}\index[Main]{\tt #1~()}\index[MethodsAndFields]{\tt #1~()}}%
3 }
4 \newcommand{\starMethodName}[1]{%
5 {\tt *#1~()}\index[Main]{#1~()@\tt *#1}}\index[MethodsAndFields]{*#1~()@\tt *#1~()}}%
6 }

```

Some commands have a variant of the form **\*Name\*** to produce only their arguments, with no additional text:

```

1 \newcommand{\file}[1]{%
2 file {\tt #1}\index[Main]{\tt #1}\index[Files]{\tt #1}}%
3 }
4 \newcommand{\File}[1]{%
5 File {\tt #1}\index[Main]{\tt #1}\index[Files]{\tt #1}}%
6 }
7
8 \newcommand{\fileName}[1]{%
9 {\tt #1}\index[Main]{\tt #1}\index[Files]{\tt #1}}%
10 }
11 \newcommand{\fileNameBoth}[1]{%
12 {\textcolor{brown}{\tt *#1.h/.cpp}}\index[Main]{#1.h/.cpp@\tt *#1.h/.cpp}}\index[Files
13 ]{#1.h/.cpp@\tt *#1.h/.cpp}}%

```

Some commands are in the form **\*Repr** : the designate the name of a representation, such as:

```

1 \newcommand{\msrRepr}{MSR\index[Main]{MSR}}

```



## 2.2 About this document

This document provides cross views of the information needed for MusicFormats maintainance. It is organized in a number of parts:

- the first part provides an overview of the library, together with the concepts it uses;
- then the two-phase visitors pattern, which is central to MusicFormats, is presented;
- the third part is dedicated to the programming style and conventions used throughout the code base;
- the OAH (Options and help), a pervasive feature in MusicFormats, is detailed;
- the fifth part details the formats provided by the library;
- the following parts are dedicated to passes, generators and converters, respectively;
- the ninth part presents the interfaces to the formats, passes and converters;
- the tenth part provides a longitudinal view of the handling of selected music score contents elements, grouped by such elements such as staves, tuplets and harmonies;
- and finally, the last part is dedicated to the implementation of the MSDDL language.

## 2.3 The MusicFormats architecture

## 2.4 User guide

[documentation/MusicFormatsUserGuide/MusicFormatsUserGuide.pdf](#) is the usual user guide. It presents the use of MusicFormats with the command line for the time being.

## 2.5 API guide

[documentation/MusicFormatsAPIGuide/MusicFormatsAPIGuide.pdf](#) presents the use of MusicFormats through the APIs. The latter are used internally by the command line tools, and can be used from applications at will, such as in a Web site.

## 2.6 Maintainance guide

[documentation/MusicFormatsMaintainanceGuide/MusicFormatsMaintainanceGuide.pdf](#) describes the internals of MusicFormats from a maintainer's point of view. It contains a detailed presentation of the various types used, and a part dedicated to selected topics: this is to have a longitudinal view of how various music elements are handled in the various passes.

## Chapter 3

# Building MusicFormats

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

- a C++17 compiler;
- the `cmake` tool.

The supported operating systems both 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.

The C++17 language is needed because MusicFormats uses `<std::regex>` and the `auto` keyword. More recent versions should not be a problem.

### 3.1 Cloning the repository

Commands such as the following can be used to clone the master and version branches, respectively:

```
1 MUSIC_FORMATS_DIR=${HOME}/musicformats-git-dev
2 git clone https://github.com/jacques-menu/musicformats.git ${MUSIC_FORMATS_DIR}
3 cd ${MUSIC_FORMATS_DEV}
```

```
1 VERSION_BRANCH=v0.9.59
2 MUSIC_FORMATS_DIR=${HOME}/musicformats-git-${VERSION_BRANCH}
3 git clone -b ${VERSION_BRANCH} https://github.com/jacques-menu/musicformats.git ${MUSIC_FORMATS_DIR}
4 cd ${MUSIC_FORMATS_DIR}
```

### 3.2 One-shot partial make runs

Some parts of the source code base have to be created by their own `make` file once and for all. This is the case of:

- the constants and classes generated for MXSR from the MusicXML DTD by `libmusicxml/src/elements/templ`
- the MFSL scanner and parser in the `mfs1/` directory, generated by `flex` and `bison` respectively.

### 3.3 cmake configuration

This configuration is in `build/CMakeLists.txt`.

### 3.4 Library Makefile

This Makefile is `build/Makefile`.

### 3.5 CLI amplex Makefile

This Makefile is `src/clisamples/Makefile`.

### 3.6 Building MusicFormats in practise

Once in the local repository clone, just execute:

```
1 cd build
2 make
```

The resulting executables are in `build/bin`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > ll build/bin
2 total 754368
3      0 drwxr-xr-x@ 26 jacquesmenu  staff      832 Sep 27 00:05:02 2021 ./
4      0 drwxr-xr-x  11 jacquesmenu  staff      352 Aug  1 18:32:54 2021 ../
5 72072 -rwxr-xr-x   1 jacquesmenu  staff 36899440 Sep 27 00:04:52 2021 LilyPondIssue34*
6 72080 -rwxr-xr-x   1 jacquesmenu  staff 36902528 Sep 27 00:04:54 2021
7      Mikrokosmos3Wandering*
8 8504 -rwxr-xr-x   1 jacquesmenu  staff 4350480 Sep 27 00:04:49 2021 MusicAndHarmonies*
9 8504 -rwxr-xr-x   1 jacquesmenu  staff 4350464 Sep 27 00:05:00 2021 RandomChords*
10 8504 -rwxr-xr-x   1 jacquesmenu  staff 4350448 Sep 27 00:05:01 2021 RandomMusic*
11 8696 -rwxr-xr-x   1 jacquesmenu  staff 4450928 Sep 27 00:04:56 2021 countnotes*
12 63904 -rwxr-xr-x   1 jacquesmenu  staff 32717248 Sep 27 00:04:57 2021
13      libMultipleInitsTest*
14 76696 -rwxr-xr-x   1 jacquesmenu  staff 39266928 Sep 27 00:05:01 2021 msdlconverter*
15 144 -rwxr-xr-x   1 jacquesmenu  staff    70480 Sep 27 00:04:55 2021 musicformatsversion
16      *
17 12616 -rwxr-xr-x   1 jacquesmenu  staff 6455376 Sep 27 00:04:59 2021 partsummary*
18 8920 -rwxr-xr-x   1 jacquesmenu  staff 4564864 Sep 27 00:04:59 2021 readunrolled*
19 81048 -rwxr-xr-x   1 jacquesmenu  staff 41496208 Sep 27 00:04:49 2021 xml2Any*
20 61232 -rwxr-xr-x   1 jacquesmenu  staff 31347456 Sep 27 00:04:53 2021 xml2brl*
21 63704 -rwxr-xr-x   1 jacquesmenu  staff 32615072 Sep 27 00:04:47 2021 xml2gmn*
22 17368 -rwxr-xr-x   1 jacquesmenu  staff 8891744 Sep 27 00:04:56 2021 xml2guido*
23 63896 -rwxr-xr-x   1 jacquesmenu  staff 32713936 Sep 27 00:04:50 2021 xml2ly*
24 12512 -rwxr-xr-x   1 jacquesmenu  staff 6403968 Sep 27 00:04:55 2021 xml2midi*
25 56384 -rwxr-xr-x   1 jacquesmenu  staff 28865024 Sep 27 00:04:59 2021 xml2xml*
26 9176 -rwxr-xr-x   1 jacquesmenu  staff 4695472 Sep 27 00:04:55 2021 xmlclone*
27 9320 -rwxr-xr-x   1 jacquesmenu  staff 4771024 Sep 27 00:05:00 2021 xmlfactory*
28 8912 -rwxr-xr-x   1 jacquesmenu  staff 4559072 Sep 27 00:04:57 2021 xmliter*
29 8752 -rwxr-xr-x   1 jacquesmenu  staff 4478336 Sep 27 00:04:55 2021 xmlread*
30 12104 -rwxr-xr-x   1 jacquesmenu  staff 6193216 Sep 27 00:04:54 2021 xmltranspose*
31 9320 -rwxr-xr-x   1 jacquesmenu  staff 4770128 Sep 27 00:05:02 2021 xmlversion*
```

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

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > ll build/lib
2 total 1888712
3      0 drwxr-xr-x  10 jacquesmenu  staff      320 Sep 27 00:04:46 2021 ./
4      0 drwxr-xr-x  11 jacquesmenu  staff      352 Aug  1 18:32:54 2021 ../
5 104904 -rwxr-xr-x   1 jacquesmenu  staff 53707712 Sep 27 00:04:46 2021 libmusicxml2
6      .3.2.0.dylib*
7      0 lrwxr-xr-x   1 jacquesmenu  staff      24 Sep 27 00:04:45 2021 libmusicxml2.3.
8      dylib@ -> libmusicxml2.3.2.0.dylib
9 1055040 -rw-r--r--   1 jacquesmenu  staff 532838416 Sep 27 00:04:41 2021 libmusicxml2.a
10 591776 -rw-r--r--   1 jacquesmenu  staff 302989312 Sep 21 09:05:55 2021 libmusicxml2.a.
11      A93i4n
12 57056 -rw-r--r--   1 jacquesmenu  staff 29212672 Sep 21 09:01:27 2021 libmusicxml2.a.
13      KHrJT0
14 39968 -rw-r--r--   1 jacquesmenu  staff 20463616 Sep 21 09:11:20 2021 libmusicxml2.a.
15      gZfmqe
16 39968 -rw-r--r--   1 jacquesmenu  staff 20463616 Sep 21 09:09:22 2021 libmusicxml2.a.
17      tndUAV
18      0 lrwxr-xr-x   1 jacquesmenu  staff      20 Sep 27 00:04:45 2021 libmusicxml2.
19      dylib@ -> libmusicxml2.3.dylib

```

## Chapter 4

# Code base structure

The code base of the MusicFormats library contains:

- `build` : a set of files to build the library in various environments with `make`
- `doc` : the documentation in  $\text{\LaTeX}$  source and PDF formats
- `files` : a set of sample files for MusicXML and MSDL
- `javascript` : a set of files for the generation of Java Script, to allow the use of MusicFormats in Web sites
- `libmusicxml` : an embedded copy of the `libmusicxml2` code base
- `packages` : a set of files to create installable versions of the library, not yet operational
- `samples` : the main programs for examples supplied with MusicFormats, such as generators and converters
- `schemas` : a set of files defining the input languages, currently MusicXML, BMML and MEI, together with scripts to generate the set of classes definitions for analyzing them
- `src` the library code base, detailed below
- `validation` : a set of files including a `Makefile` for the validation of the library using the contents of files
- `win32` : Windows related support

### 4.1 The `libmusicxml` folder

This folder contains a version of Grame's `libmusicxml2` library, available at <https://github.com/grame-cncm/libmusicxml>. It is used by MusicFormats, to avoid the need for installing it separately.

The only possible annoyance when installing both libraries is that the executables in `libmusicxml/build/bin` such as `countnotes` and `xml2guido` are installed twice: choosing which one to use can be handled in the `PATH` and `LD_LIBRARY_PATH` environment variables or their equivalent.

### 4.1.1 Embedding libmusicxml in MusicFormats

libmusicxml2 was cloned initially like this:

```
1 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev > git clone https://github.com/
   grame-cncm/libmusicxml -b dev
2 Cloning into 'libmusicxml'...
3 remote: Enumerating objects: 56386, done.
4 remote: Counting objects: 100% (4692/4692), done.
5 remote: Compressing objects: 100% (777/777), done.
6 remote: Total 56386 (delta 3917), reused 4671 (delta 3904), pack-reused 51694
7 Receiving objects: 100% (56386/56386), 105.98 MiB | 15.91 MiB/s, done.
8 Resolving deltas: 100% (46834/46834), done.
```

libmusicxml2 is quite stable, and it can be upgraded if needed with:

```
1 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml > git pull
2 Already up to date.
```

There is no need to build libmusicxml2 manually, since its code is taken into account by the MusicFormats Makefile.

### 4.1.2 Upgrading the supported MusicXML version

It may be necessary to upgrade the MusicXML DTD in `libmusicxml/elements` to keep up with evolutions if libmusicxml2 is not up to date yet.

To upgrade from MusicXML 3.1 to MusicXML 4.0, the following has been done. `Makefile_ORIGINAL` is a symbolic link to the `Makefile` provided by libmusicxml2 for 3.1 at the time of this writing:

```
1 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml/schema > ls -sal
2 total 208
3 0 drwxr-xr-x 14 jacquesmenu staff 448 Jul 30 05:59 .
4 0 drwxr-xr-x 22 jacquesmenu staff 704 Jul 29 07:19 ..
5 0 drwxr-xr-x 6 jacquesmenu staff 192 Jul 29 07:19 2.0
6 0 drwxr-xr-x 6 jacquesmenu staff 192 Jul 29 07:19 3.0
7 0 drwxr-xr-x 6 jacquesmenu staff 192 Jul 29 07:19 3.1
8 0 drwxr-xr-x 8 jacquesmenu staff 256 Jul 29 08:04 4.0
9 8 -rw-r--r-- 1 jacquesmenu staff 1215 Jul 29 07:19 Makefile
10 8 -rw-r--r-- 1 jacquesmenu staff 1215 Jul 29 07:19 Makefile_3.1
11 0 lrwxr-xr-x 1 jacquesmenu staff 8 Jul 30 05:59 Makefile_ORIGINAL -> Makefile
```

First, create the `Makefile` for version 4.0:

```
1 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml/schema > sed 's
   /3.1/4.0/g' Makefile_3.1 > Makefile_4.0
```

Then use it to create the C++ files containing the constants and types to be used by MXSR, such as `k_accidental`, `S_accidental`:

```
1 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml/schema > make -f
   Makefile_4.0
2 grep "<xs:element" 4.0/musicxml.xsd | sed -e 's/^.*name="//' | sed -e 's/"..*//' | sort -u
   > elements.txt
3 ../src/elements/templates/elements.bash elements.txt ../src/elements/templates constants >
   elements.h || rm -f elements.h
4 ../src/elements/templates/elements.bash elements.txt ../src/elements/templates types >
   typedefs.h || rm -f typedefs.h
5 ../src/elements/templates/elements.bash elements.txt ../src/elements/templates map >
   factory.cpp || rm -f factory.cpp
```

The resulting files are the following, where `elements.txt` contains an alphabetic list of the MusicXML markups found in the DTD:

```

1 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml/schema > ls -sal
2 total 208
3  0 drwxr-xr-x  13 jacquesmenu  staff    416 Jul 29 08:27 .
4  0 drwxr-xr-x  22 jacquesmenu  staff    704 Jul 29 07:19 ..
5  0 drwxr-xr-x   6 jacquesmenu  staff    192 Jul 29 07:19 2.0
6  0 drwxr-xr-x   6 jacquesmenu  staff    192 Jul 29 07:19 3.0
7  0 drwxr-xr-x   6 jacquesmenu  staff    192 Jul 29 07:19 3.1
8  0 drwxr-xr-x   8 jacquesmenu  staff    256 Jul 29 08:04 4.0
9  8 -rw-r--r--   1 jacquesmenu  staff   1215 Jul 29 07:19 Makefile_3.1
10 8 -rw-r--r--@  1 jacquesmenu  staff   1215 Jul 29 08:26 Makefile_4.0
11 8 -rw-r--r--   1 jacquesmenu  staff   1215 Jul 29 07:19 Makefile_ORIGINAL
12 16 -rw-r--r--   1 jacquesmenu  staff   7130 Jul 29 08:27 elements.h
13 16 -rw-r--r--   1 jacquesmenu  staff   4561 Jul 29 08:27 elements.txt
14 96 -rw-r--r--   1 jacquesmenu  staff  46341 Jul 29 08:27 factory.cpp
15 56 -rw-r--r--   1 jacquesmenu  staff  24604 Jul 29 08:27 typedefs.h

```

The mapping between the makups text and the types that describes them is done with:

```

1 typedef SMARTP<musicxml<k_accidental> >    S_accidental;
2
3 fMap["accidental"] = new newElementFunctor<k_accidental>;

```

Finally, copy the new C++ files to the `libmusicxml/elements` folder:

```

1 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml/schema > cp -p
   elements.h factory.cpp typedefs.h ../src/elements

```

Now, rebuild MusicFormats, for it to use the new MusicXML DTD: `/libdir/CMakeCache.txt`

```

1 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/build > rm libdir/CMakeCache.txt
2 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/build > make

```

It may happen that error messages regarding the new markups are issued:

```

1 Undefined symbols for architecture x86_64:
2  "MusicFormats::mxsr2msrTranslator::visitStart(MusicXML2::SMARTP<MusicXML2::musicxml<241>
   >&)", referenced from:
3  "MusicFormats::mxsr2msrTranslator::visitStart(MusicXML2::SMARTP<MusicXML2::musicxml<266>
   >&)", referenced from:
4  "MusicFormats::mxsr2msrTranslator::visitStart(MusicXML2::SMARTP<MusicXML2::musicxml<284>
   >&)", referenced from:
5  "MusicFormats::mxsr2msrTranslator::visitStart(MusicXML2::SMARTP<MusicXML2::musicxml<29>
   >&)", referenced from:
6  "non-virtual thunk to MusicFormats::mxsr2msrTranslator::visitStart(MusicXML2::SMARTP<
   MusicXML2::musicxml<284> >&)", referenced from:
7  "non-virtual thunk to MusicFormats::mxsr2msrTranslator::visitStart(MusicXML2::SMARTP<
   MusicXML2::musicxml<29> >&)", referenced from:
8 clang: error: linker command failed with exit code 1 (use -v to see invocation)
9 ** BUILD FAILED **
10 make[1]: *** [macos] Error 65
11 make: *** [all] Error 2

```

In that case, the corresponding constants can be found in `elements.h`, at line `'26+numericalValue'`. For example, with MusicXML version 4.0, 241 is the numerical value of `k_notations`, describing markup `<"notations"/>`:

```

1 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml/src/elements > grep -
  n kNoElement elements.h
2 26: kNoElement,
3 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml/src/elements > grep -
  n k_notations elements.h
4 267: k_notations,
5 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml/src/elements > grep -
  n k_other_notation elements.h
6 292: k_other_notation,
7 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml/src/elements > grep -
  n k_part_name elements.h
8 310: k_part_name,
9 311: k_part_name_display,
10 jacquesmenu@mac-mini-de-jacques-1:~/musicformats-git-dev/libmusicxml/src/elements > grep -
  n k_bass elements.h
11 55: k_bass,
12 56: k_bass_alter,
13 57: k_bass_separator,
14 58: k_bass_step,

```

The incriminated MXSR elements are thus `k_notations`, `k_other_notation`, `k_part_name` and `k_bass`.

The first 4 error messages above mean that the corresponding `visitStart ()` methods are declared alright, but are not defined in `src/passes/mxsr2msr/mxsr2msrTranslator.cpp`.

The following methods definitions are thus missing:

- `method mxsr2msrTranslator::visitStart ( S_notations& elt)`
- `method mxsr2msrTranslator::visitStart ( S_other_notation& elt)`
- `method mxsr2msrTranslator::visitStart ( S_part_name& elt)`
- `method mxsr2msrTranslator::visitStart ( S_bass& elt)`

Type `S_part_name` is there by mistake (some typing was done before the upgrade to 4.0), since it is handled in class `mxsr2msrSkeletonBuilder`, and the other 3 are new in MusicXML 4.0.

## 4.2 The doc folder

This folder contains `LaTeXCommonSettings.tex`, included by the various  $\text{\LaTeX}$  documents whose code is in the respective folders, together with the PDF files:

```

1 jacquesmenu@macmini:~/musicformats-git-dev/documentation > ll
2 total 32
3  0 drwxr-xr-x  11 jacquesmenu  staff    352 Nov  3 09:59:31 2022 ./
4  0 drwxr-xr-x  34 jacquesmenu  staff   1088 Nov  3 16:17:01 2022 ../
5 32 -rw-r--r--@   1 jacquesmenu  staff  14340 Nov  3 14:15:54 2022 .DS_Store
6  0 drwxr-xr-x  25 jacquesmenu  staff    800 Nov  3 09:57:22 2022 CommonLaTeXFiles/
7  0 drwxr-xr-x  32 jacquesmenu  staff   1024 Jun 28 20:52:12 2022 IntroductionToMusicXML/
8  0 drwxr-xr-x  48 jacquesmenu  staff   1536 Nov  1 14:24:27 2022 MusicFormatsAPIGuide/
9  0 drwxr-xr-x   9 jacquesmenu  staff    288 Nov  3 09:47:22 2022 MusicFormatsFigures/
10 0 drwxr-xr-x 106 jacquesmenu  staff   3392 Nov  3 10:58:57 2022
   MusicFormatsMaintenanceGuide/
11 0 drwxr-xr-x  55 jacquesmenu  staff   1760 Nov  1 14:24:15 2022 MusicFormatsUserGuide/
12 0 drwxr-xr-x  46 jacquesmenu  staff   1472 Aug 31 11:09:53 2022 graphics/
13 0 drwxr-xr-x   5 jacquesmenu  staff    160 Jun 28 20:52:12 2022 libmusicxml2Presentation
   /

```

`common` contains a set of files used by the various documents and various stuff:



```

1 jacquesmenu@macmini: ~/musicformats-git-dev/documentation/common > ll
2 total 776
3  0 drwxr-xr-x  9 jacquesmenu  staff      288 Jan  4 17:23:41 2022 ./
4  0 drwxr-xr-x  9 jacquesmenu  staff      288 Jan  4 17:23:41 2022 ../
5 624 -rw-r--r--  1 jacquesmenu  staff  318497 Apr 22 15:48:40 2021 INSIDE.pdf
6  8 -rw-r--r--@  1 jacquesmenu  staff      321 Sep  8 18:15:51 2021 MusicFormats.lst
7 48 -rw-r--r--@  1 jacquesmenu  staff  21751 Dec 28 18:45:25 2021
   MusicFormatsArchitecturePicture.tex
8 80 -rw-r--r--@  1 jacquesmenu  staff  39133 Jan  4 17:18:28 2022 LaTeXCommonSettings.tex
9  8 -rwxr-xr-x@  1 jacquesmenu  staff      157 Jan  4 09:43:30 2022
   createCurrentVersionNumberString.bash*
10 0 drwxr-xr-x 12 jacquesmenu  staff      384 Apr 22 15:48:41 2021 images_KEEP/
11 8 -rw-r--r--@  1 jacquesmenu  staff        7 Jan  4 09:25:02 2022
   MusicFormatsVersionNumber.txt

```

The `presentation` sub-folder contains the documentation of the `libmusicxml2` library, written by Dominique Fober:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/documentation/libmusicxml2Presentation > ll
2 total 416
3  0 drwxr-xr-x  5 jacquesmenu  staff      160 Jun 28 20:52:12 2022 ./
4  0 drwxr-xr-x 11 jacquesmenu  staff      352 Nov  3 09:59:31 2022 ../
5  0 drwxr-xr-x  6 jacquesmenu  staff      192 Jun 28 20:52:12 2022 imgs/
6 392 -rw-r--r--@  1 jacquesmenu  staff  200524 Jun 28 20:52:12 2022 libmusicxml2.pdf
7 24 -rw-r--r--  1 jacquesmenu  staff   11017 Jun 28 20:52:12 2022 libmusicxml2.tex

```

### 4.3 The schemas folder

This folder contains the definitions used to create the classes definitions to analyze textual data in the MusicXML, MEI and BMML formats.

In the `scripts` folder, `elements.bash` compiles the definitions of MusicXML markups into the C++ code files containing the corresponding C++ classes:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/schemas > ll
2 total 2576
3  0 drwxr-xr-x  9 jacquesmenu  staff      288 May 21 18:30:08 2021 ./
4  0 drwxr-xr-x 22 jacquesmenu  staff      704 May 25 17:19:16 2021 ../
5 16 -rw-r--r--  1 jacquesmenu  staff   6148 May 21 18:30:08 2021 .DS_Store
6  0 drwxr-xr-x  4 jacquesmenu  staff      128 Apr 22 15:49:14 2021 BMML/
7  0 drwxr-xr-x  5 jacquesmenu  staff      160 May 21 18:30:08 2021 MEI/
8  8 -rw-r--r--  1 jacquesmenu  staff   2502 Apr 22 15:49:15 2021 Makefile
9  0 drwxr-xr-x  6 jacquesmenu  staff      192 May 21 18:30:08 2021 MusicXML/
10 2552 -rw-r--r--  1 jacquesmenu  staff  1305905 Apr 22 15:49:13 2021 mei-CMN.rng
11  0 drwxr-xr-x  3 jacquesmenu  staff        96 Apr 22 15:49:08 2021 scripts/

```

### 4.4 The src folder

The `src` folder has the following structure:

- `clisamples` : the `main ()` functions of the various command line executables provided by MusicFormats:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > ll clisamples/
2 total 320
3 0 drwxr-xr-x 16 jacquesmenu staff 512 May 24 10:58:19 2021 ./
4 0 drwxr-xr-x 22 jacquesmenu staff 704 May 25 17:19:16 2021 ../
5 16 -rw-r--r-- 1 jacquesmenu staff 6148 May 21 18:30:07 2021 .DS_Store
6 8 -rw-r--r-- 1 jacquesmenu staff 116 Apr 22 15:49:06 2021 .gitignore
7 40 -rw-r--r--@ 1 jacquesmenu staff 20239 May 24 11:17:46 2021 LilyPondIssue34.cpp
8 8 -rw-r--r-- 1 jacquesmenu staff 1615 Apr 22 15:49:15 2021 Makefile
9 40 -rw-r--r--@ 1 jacquesmenu staff 20239 May 24 11:07:21 2021
   Mikrokosmos3Wandering.cpp
10 24 -rw-r--r-- 1 jacquesmenu staff 9941 May 21 18:30:07 2021 MusicAndHarmonies.
   cpp
11 8 -rw-r--r-- 1 jacquesmenu staff 3114 Apr 22 15:49:15 2021 libMultipleInitsTest
   .cpp
12 48 -rw-r--r-- 1 jacquesmenu staff 23061 May 21 18:30:07 2021 msdl.cpp
13 8 -rw-r--r-- 1 jacquesmenu staff 895 May 21 18:30:07 2021 musicformatsversion.
   cpp
14 24 -rw-r--r-- 1 jacquesmenu staff 10492 Apr 22 15:49:14 2021 xml2Any.cpp
15 24 -rw-r--r-- 1 jacquesmenu staff 10076 May 21 18:30:07 2021 xml2brl.cpp
16 24 -rw-r--r-- 1 jacquesmenu staff 10515 May 21 18:30:07 2021 xml2gmn.cpp
17 24 -rw-r--r-- 1 jacquesmenu staff 10309 May 21 18:30:07 2021 xml2ly.cpp
18 24 -rw-r--r-- 1 jacquesmenu staff 10463 May 21 18:30:08 2021 xml2xml.cpp

```

- **converters** : the multi-pass converter combining those in passes

```

- msdl2braille
- msdl2guido
- msdl2lilypond
- msdl2musicxml
- msdlconverter

- msr2braille
- msr2guido
- msr2lilypond
- msr2musicxml

- musicxml2braille
- musicxml2guido
- musicxml2lilypond
- musicxml2musicxml

```

- **generators** :

```

- LilyPondIssue34
- Mikrokosmos3Wandering

```

- **components** : the MusicFormats components formats, including versions numbering and history:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll components
2 total 168
3 0 drwxr-xr-x 7 jacquesmenu staff 224 Oct 22 08:53:06 2021 ./
4 0 drwxr-xr-x 19 jacquesmenu staff 608 Oct 22 05:29:29 2021 ../
5 8 -rw-r--r--@ 1 jacquesmenu staff 1106 Oct 22 09:16:21 2021 mfcComponents.h
6 96 -rw-r--r--@ 1 jacquesmenu staff 46691 Nov 8 12:59:57 2021 mfcBasicTypes.cpp
7 40 -rw-r--r--@ 1 jacquesmenu staff 20121 Nov 8 12:59:43 2021 mfcBasicTypes.h

```

```

8 16 -rw-r--r--@ 1 jacquesmenu staff 4950 Nov 8 12:59:08 2021 mfcLibraryComponent.
   cpp
9 8 -rw-r--r--@ 1 jacquesmenu staff 605 Oct 22 10:36:30 2021 mfcLibraryComponent.
   h

```

- **mfutilities** : various utilities, including indented text output streams, and version history support:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll mfutilities/
2 total 200
3 0 drwxr-xr-x 15 jacquesmenu staff 480 Oct 22 06:25:57 2021 ./
4 0 drwxr-xr-x 19 jacquesmenu staff 608 Oct 22 05:29:29 2021 ../
5 8 -rw-r--r--@ 1 jacquesmenu staff 3255 Oct 18 20:22:16 2021 mfBool.cpp
6 16 -rw-r--r--@ 1 jacquesmenu staff 4917 Oct 18 19:56:51 2021 mfBool.h
7 8 -rw-r--r--@ 1 jacquesmenu staff 1336 Oct 15 18:48:10 2021 mfEnumAll.h
8 16 -rw-r--r--@ 1 jacquesmenu staff 7182 Nov 8 13:08:51 2021 mfIndentedTextOutput
   .cpp
9 16 -rw-r--r--@ 1 jacquesmenu staff 7715 Nov 8 13:08:40 2021 mfIndentedTextOutput
   .h
10 8 -rw-r--r--@ 1 jacquesmenu staff 889 Oct 15 20:34:47 2021
   mfMusicformatsErrorKind.cpp
11 8 -rw-r--r--@ 1 jacquesmenu staff 629 Oct 15 20:34:47 2021 mfMusicformatsErrors
   .h
12 8 -rw-r--r--@ 1 jacquesmenu staff 2541 Nov 5 11:29:25 2021 oahOptionsVector.cpp
13 8 -rw-r--r--@ 1 jacquesmenu staff 972 Oct 15 20:16:51 2021 oahBasicTypes.h
14 64 -rw-r--r--@ 1 jacquesmenu staff 29773 Oct 15 18:48:10 2021 mfStringsHandling.
   cpp
15 16 -rw-r--r--@ 1 jacquesmenu staff 6269 Oct 15 18:55:46 2021 mfStringsHandling.h
16 16 -rw-r--r--@ 1 jacquesmenu staff 5028 Oct 7 20:03:27 2021 mfTiming.cpp
17 8 -rw-r--r--@ 1 jacquesmenu staff 3726 Oct 8 08:21:09 2021 mfTiming.h

```

- **oah** : object-oriented Options And Help support

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll oah
2 total 1456
3 0 drwxr-xr-x 34 jacquesmenu staff 1088 Nov 16 08:12:11 2021 ./
4 0 drwxr-xr-x 20 jacquesmenu staff 640 Nov 16 08:12:03 2021 ../
5 48 -rw-r--r--@ 1 jacquesmenu staff 23743 Nov 16 08:16:55 2021 basicOah2manPage.
   cpp
6 16 -rw-r--r--@ 1 jacquesmenu staff 5202 Nov 15 12:56:16 2021 basicOah2manPage.h
7 8 -rw-r--r--@ 1 jacquesmenu staff 539 Jun 6 06:38:55 2021
   enableHarmoniesExtraOahIfDesired.h
8 8 -rw-r--r--@ 1 jacquesmenu staff 526 Oct 11 11:56:29 2021
   oahEnableTracingIfDesired.h
9 72 -rw-r--r--@ 1 jacquesmenu staff 34280 Nov 16 08:16:55 2021 harmoniesExtraOah.
   cpp
10 24 -rw-r--r--@ 1 jacquesmenu staff 11848 Nov 15 12:56:16 2021 harmoniesExtraOah.
   h
11 16 -rw-r--r--@ 1 jacquesmenu staff 5154 Nov 16 08:16:55 2021
   harmoniesExtraOah2manPage.cpp
12 8 -rw-r--r--@ 1 jacquesmenu staff 1689 Nov 15 12:56:16 2021
   harmoniesExtraOah2manPage.h
13 8 -rw-r--r--@ 1 jacquesmenu staff 918 Nov 16 08:16:55 2021 oah2manPage.cpp
14 8 -rw-r--r--@ 1 jacquesmenu staff 912 Nov 15 12:56:16 2021 oah2manPage.h
15 344 -rw-r--r--@ 1 jacquesmenu staff 175094 Nov 16 08:16:55 2021 oahAtomsCollection
   .cpp
16 176 -rw-r--r--@ 1 jacquesmenu staff 87460 Nov 15 12:56:16 2021 oahAtomsCollection
   .h
17 336 -rw-r--r--@ 1 jacquesmenu staff 168969 Nov 16 08:16:55 2021 oahBasicTypes.cpp
18 96 -rw-r--r--@ 1 jacquesmenu staff 47228 Nov 15 12:56:16 2021 oahBasicTypes.h
19 8 -rw-r--r--@ 1 jacquesmenu staff 3258 Nov 16 08:16:55 2021 oahBrowsers.h
20 32 -rw-r--r--@ 1 jacquesmenu staff 14030 Nov 16 08:16:55 2021 oahElements.cpp
21 24 -rw-r--r--@ 1 jacquesmenu staff 10381 Nov 15 12:56:16 2021 oahElements.h
22 8 -rw-r--r--@ 1 jacquesmenu staff 2577 Nov 16 08:16:55 2021 oahInsiderHandlers
   .cpp
23 8 -rw-r--r--@ 1 jacquesmenu staff 2982 Nov 15 12:56:16 2021 oahInsiderHandlers
   .h

```

```

24 56 -rw-r--r--@ 1 jacquesmenu staff 25901 Nov 16 08:16:55 2021 oah0ah.cpp
25 32 -rw-r--r--@ 1 jacquesmenu staff 13849 Nov 16 08:16:55 2021 oah0ah.h
26 8 -rw-r--r--@ 1 jacquesmenu staff 1966 Nov 16 08:16:55 2021 oah0ah2manPage.cpp
27 8 -rw-r--r--@ 1 jacquesmenu staff 1021 Nov 15 12:56:16 2021 oah0ah2manPage.h
28 24 -rw-r--r--@ 1 jacquesmenu staff 8831 Nov 16 08:16:55 2021 oahRegularHandlers
   .cpp
29 8 -rw-r--r--@ 1 jacquesmenu staff 3855 Nov 15 12:56:16 2021 oahRegularHandlers
   .h
30 8 -rw-r--r--@ 1 jacquesmenu staff 568 Nov 15 12:56:16 2021 oahVisitor.cpp
31 8 -rw-r--r--@ 1 jacquesmenu staff 894 Nov 15 12:56:16 2021 oahVisitor.h
32 16 -rw-r--r--@ 1 jacquesmenu staff 5978 Nov 16 08:16:55 2021 outputFile0ah.cpp
33 8 -rw-r--r--@ 1 jacquesmenu staff 3593 Nov 15 12:56:16 2021 outputFile0ah.h
34 8 -rwxr--r--@ 1 jacquesmenu staff 236 Oct 23 12:02:12 2021 zsh_test.zsh*

```

- `formatsgeneration` : support for various output kinds

- `brailleGeneration`
- `guidoGeneration`
- `lilypondGeneration`
- `msrGeneration`
- `multiGeneration`
- `mxsrGeneration`

- `passes` : code for the individual passes

- `bsr2braille`
- `bsr2bsr`
- `lpsr2lilypond`
- `msr2bsr`
- `msr2lpsr`
- `msr2msr`
- `msr2mxsr`
- `mxsr2guido`
- `mxsr2msr`
- `mxsr2musicxml`

- `formats` : the various internal representations used by `MusicFormats`

- `bsr`
- `lpsr`
- `msdl`
- `msdr`
- `msr`
- `msrapi`
- `mxsr`

- `wae` : multilingual Warnings And Errors support, including exceptions handling

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll wae/
2 total 104
3  0 drwxr-xr-x   9 jacquesmenu  staff    288 Oct 15 20:23:46 2021 ./
4  0 drwxr-xr-x  20 jacquesmenu  staff    640 Nov 16 08:12:03 2021 ../
5  8 -rw-r--r--@   1 jacquesmenu  staff    680 Jun  6 06:35:19 2021
   waeEnableAbortToDebugErrors.h
6  8 -rw-r--r--@   1 jacquesmenu  staff    602 Nov 15 12:56:18 2021 waeExceptions.cpp
7 24 -rw-r--r--@   1 jacquesmenu  staff  11514 Nov 15 12:56:18 2021 waeExceptions.h
8  8 -rw-r--r--@   1 jacquesmenu  staff   1393 Nov 16 08:16:55 2021 waeHandlers.cpp
9  8 -rw-r--r--@   1 jacquesmenu  staff   1550 Nov 15 12:56:18 2021 waeHandlers.h
10 32 -rw-r--r--@   1 jacquesmenu  staff  16317 Nov 15 12:56:18 2021 wae.cpp
11 16 -rw-r--r--@   1 jacquesmenu  staff   5794 Nov 15 12:56:18 2021 waeInterface.h

```

## 4.5 The validation folder

This folder contains a `Makefile` to compile all the files in the `files` folder. `musicformatsversion.txt` contains a validation version number, without a priori relation to the actual version number of the library, for example:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/validation > cat musicformatsversion.txt
2 1.0.0

```

In this example, `make` will create a folder named `1.0.0` containing the converted files, including PDF files produced by `lilypond`.

This validation version number allows for comparisons between version to ease regression tests.

There is no `clean` target in the `Makefile`. Removing the `1.0.0` folder in this case does the equivalent, then we can run `make` again.

# Chapter 5

## Components

### 5.1 Components terminology

In compiler writing terminology:

- an external format
- an internal representation is a data structure representing the program being compiled;
- there are often several internal representations, to simplify the compiler internal workings or for optimisation purposes;
- the output of the compiler, such as binary code for some physical or emulated processor, is a last 'representation' of the program;
- a pass converts an internal representation into another one, in a single step;
- a multi-pass converter is a chain of passes, reading the input, converting it into a first internal representation, then a pass to convert it into another internal representation, and so on until the compiler output is produced.

MusicFormats maps exactly to this model, providing the following components:

- internal representations (formats for short) of the music score: MSR, LSPR, BSR and MXSR;
- several passes are available to convert such formats into others;
- a set of multi-pass converters are supplied, such as `xml2lily` `xml2xml` and MSDL converter.

In the MusicFormats user documentation, the term 'converter' is used because it is more meaningful for musicians.

MusicFormats provides high-level interfaces to its components as functions in **Interface** files:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > look Interface
2 ./formats/msr/msrInterface.cpp
3 ./formats/msr/msrInterface.h
4 ./formats/lpsr/lpsrInterface.cpp
5 ./formats/lpsr/lpsrInterface.h
6 ./formats/bsr/bsrInterface.cpp
7 ./formats/bsr/bsrInterface.h
8 ./passes/mxsr2musicxml/mxsr2musicxmlTranlatorInterface.h
9 ./passes/mxsr2musicxml/mxsr2musicxmlTranlatorInterface.cpp
10 ./passes/bsr2bsr/bsr2bsrFinalizerInterface.h
11 ./passes/bsr2bsr/bsr2bsrFinalizerInterface.cpp
12 ./passes/msr2mxsr/msr2mxsrInterface.cpp
13 ./passes/msr2mxsr/msr2mxsrInterface.h
14 ./passes/mxsr2msr/mxsr2msrSkeletonBuilderInterface.h
15 ./passes/mxsr2msr/mxsr2msrTranslatorInterface.cpp
16 ./passes/mxsr2msr/mxsr2msrTranslatorInterface.h
17 ./passes/mxsr2msr/mxsr2msrSkeletonBuilderInterface.cpp
18 ./passes/msr2msr/msr2msrInterface.h
19 ./passes/msr2msr/msr2msrInterface.cpp
20 ./passes/lpsr2lilypond/lpsr2lilypondInterface.h
21 ./passes/lpsr2lilypond/lpsr2lilypondInterface.cpp
22 ./passes/msr2lpsr/msr2lpsrInterface.cpp
23 ./passes/msr2lpsr/msr2lpsrInterface.h
24 ./passes/bsr2braille/bsr2brailleTranslatorInterface.h
25 ./passes/bsr2braille/bsr2brailleTranslatorInterface.cpp
26 ./passes/msr2bsr/msr2bsrInterface.h
27 ./passes/msr2bsr/msr2bsrInterface.cpp
28 ./passes/musicxml2mxsr/musicxml2mxsrInterface.h
29 ./passes/musicxml2mxsr/musicxml2mxsrInterface.cpp
30 ./passes/mxsr2guido/mxsr2guidoTranlatorInterface.h
31 ./passes/mxsr2guido/mxsr2guidoTranlatorInterface.cpp
32 ./converters/msr2guido/msr2guidoInterface.h
33 ./converters/msr2guido/msr2guidoInterface.cpp
34 ./converters/msr2braille/msr2brailleInterface.h
35 ./converters/msr2braille/msr2brailleInterface.cpp
36 ./converters/msdl2braille/msdl2brailleInterface.h
37 ./converters/msdl2braille/msdl2brailleInterface.cpp
38 ./converters/msdl2guido/msdl2guidoInterface.cpp
39 ./converters/msdl2guido/msdl2guidoInterface.h
40 ./converters/msdl2musicxml/msdl2musicxmlInterface.h
41 ./converters/msdl2musicxml/msdl2musicxmlInterface.cpp
42 ./converters/msdl2lilypond/msdl2lilypondInterface.h
43 ./converters/msdl2lilypond/msdl2lilypondInterface.cpp
44 ./converters/musicxml2braille/musicxml2brailleInterface.cpp
45 ./converters/musicxml2braille/musicxml2brailleInterface.h
46 ./converters/msr2lilypond/msr2lilypondInterface.cpp
47 ./converters/msr2lilypond/msr2lilypondInterface.h
48 ./converters/msr2musicxml/msr2musicxmlInterface.cpp
49 ./converters/msr2musicxml/msr2musicxmlInterface.h
50 ./converters/musicxml2musicxml/musicxml2musicxmlInterface.h
51 ./converters/musicxml2musicxml/musicxml2musicxmlInterface.cpp
52 ./converters/musicxml2lilypond/musicxml2lilypondInterface.h
53 ./converters/musicxml2lilypond/musicxml2lilypondInterface.cpp
54 ./converters/musicxml2guido/musicxml2guidoInterface.cpp
55 ./converters/musicxml2guido/musicxml2guidoInterface.h

```

The converters are implemented as functions as well as CLI tools that use the latter.

MusicFormats includes support for components versions numbering and history, see chapter ?? [MusicFormats components], page ??.

`src/components/mfcComponents.h` includes all the components's header files.

## 5.2 Formats

The formats are in `src/formats`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/
2 total 32
3 0 drwxr-xr-x 10 jacquesmenu staff 320 Jun 25 05:39:49 2021 ./
4 0 drwxr-xr-x 13 jacquesmenu staff 416 Jun 17 17:16:37 2021 ../
5 24 -rw-r--r-- 1 jacquesmenu staff 10244 Jun 19 07:58:55 2021 .DS_Store
6 0 drwxr-xr-x 60 jacquesmenu staff 1920 Jun 18 07:32:14 2021 bsr/
7 0 drwxr-xr-x 42 jacquesmenu staff 1344 May 26 08:20:55 2021 lpsr/
8 0 drwxr-xr-x 12 jacquesmenu staff 384 Apr 22 15:49:23 2021 msdl/
9 0 drwxr-xr-x 10 jacquesmenu staff 320 May 26 08:20:55 2021 msdr/
10 0 drwxr-xr-x 151 jacquesmenu staff 4832 Jun 20 09:58:00 2021 msr/
11 0 drwxr-xr-x 6 jacquesmenu staff 192 May 26 08:20:55 2021 mxsr/
```

The formats interfaces are in files with the format's name:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/bsr/bsr.*
2 8 -rw-r--r--@ 1 jacquesmenu staff 700 Jun 6 06:35:19 2021 formats/bsr/bsr.cpp
3 8 -rw-r--r--@ 1 jacquesmenu staff 1206 Jun 18 10:04:45 2021 formats/bsr/bsr.h
4
5 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/lpsr/lpsr.*
6 8 -rw-r--r--@ 1 jacquesmenu staff 703 Jun 6 06:35:19 2021 formats/lpsr/lpsr.cpp
7 8 -rw-r--r--@ 1 jacquesmenu staff 1004 Jun 6 06:35:19 2021 formats/lpsr/lpsr.h
8
9 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/msdl/msdl.*
10 8 -rw-r--r--@ 1 jacquesmenu staff 736 Jun 6 06:35:19 2021 formats/msdl/msdl.cpp
11 8 -rw-r--r--@ 1 jacquesmenu staff 643 Jun 6 06:35:19 2021 formats/msdl/msdl.h
12
13 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/msdr/msdr.*
14 8 -rw-r--r--@ 1 jacquesmenu staff 709 Jun 6 06:35:19 2021 formats/msdr/msdr.cpp
15 8 -rw-r--r--@ 1 jacquesmenu staff 531 Jun 6 06:35:19 2021 formats/msdr/msdr.h
16
17 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/msr/msr.*
18 8 -rw-r--r--@ 1 jacquesmenu staff 700 Jun 6 06:35:19 2021 formats/msr/msr.cpp
19 8 -rw-r--r--@ 1 jacquesmenu staff 2410 Jun 20 09:58:38 2021 formats/msr/msr.h
20
21 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/mxsr/mxsr.*
22 8 -rw-r--r--@ 1 jacquesmenu staff 3292 Jun 6 06:35:19 2021 formats/mxsr/mxsr.cpp
23 8 -rw-r--r--@ 1 jacquesmenu staff 1555 Jun 6 06:35:19 2021 formats/mxsr/mxsrGeneration.
    h
```

## 5.3 Representations

The representations are in `src/representations`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll representations/
2 total 24
3 0 drwxr-xr-x 11 jacquesmenu staff 352 Dec 30 17:25:10 2021 ./
4 0 drwxr-xr-x 18 jacquesmenu staff 576 Jan 16 16:50:25 2022 ../
5 24 -rw-r--r--@ 1 jacquesmenu staff 10244 Jan 6 17:40:44 2022 .DS_Store
6 0 drwxr-xr-x 8 jacquesmenu staff 256 Dec 30 10:26:26 2021 braille/
7 0 drwxr-xr-x 69 jacquesmenu staff 2208 Jan 4 07:52:14 2022 bsr/
8 0 drwxr-xr-x 4 jacquesmenu staff 128 Dec 30 10:27:01 2021 guido/
9 0 drwxr-xr-x 51 jacquesmenu staff 1632 Jan 4 07:52:36 2022 lpsr/
10 0 drwxr-xr-x 16 jacquesmenu staff 512 Jan 4 07:52:55 2022 msdl/
11 0 drwxr-xr-x 12 jacquesmenu staff 384 Jan 4 07:53:13 2022 msdr/
12 0 drwxr-xr-x 165 jacquesmenu staff 5280 Jan 4 07:53:34 2022 msr/
13 0 drwxr-xr-x 10 jacquesmenu staff 320 Jan 4 07:53:54 2022 mxsr/
```



## 5.4 Passes

The passs are in `src/passes`:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll passes
2 total 24
3  0 drwxr-xr-x  14 jacquesmenu  staff    448 Nov  24  16:29:20 2021 ./
4  0 drwxr-xr-x  20 jacquesmenu  staff    640 Nov  16  08:12:03 2021 ../
5 24 -rw-r--r--@  1 jacquesmenu  staff  10244 Nov  24  10:38:11 2021 .DS_Store
6  0 drwxr-xr-x   8 jacquesmenu  staff    256 Oct  22  07:19:11 2021 bsr2braille/
7  0 drwxr-xr-x   6 jacquesmenu  staff    192 Oct  22  07:20:34 2021 bsr2bsr/
8  0 drwxr-xr-x  10 jacquesmenu  staff    320 Nov  16  10:09:27 2021 lpsr2lilypond/
9  0 drwxr-xr-x  14 jacquesmenu  staff    448 Oct  22  07:22:09 2021 msdl2msr/
10 0 drwxr-xr-x   8 jacquesmenu  staff    256 Oct  22  07:24:35 2021 msr2bsr/
11 0 drwxr-xr-x   8 jacquesmenu  staff    256 Nov   1  16:31:34 2021 msr2lpsr/
12 0 drwxr-xr-x   8 jacquesmenu  staff    256 Nov   1  16:31:34 2021 msr2msr/
13 0 drwxr-xr-x   6 jacquesmenu  staff    192 Oct  22  07:27:46 2021 msr2mxsr/
14 0 drwxr-xr-x   4 jacquesmenu  staff    128 Oct  22  07:28:37 2021 mxsr2guido/
15 0 drwxr-xr-x  10 jacquesmenu  staff    320 Nov   1  16:31:34 2021 mxsr2msr/
16 0 drwxr-xr-x   4 jacquesmenu  staff    128 Oct  22  07:29:50 2021 mxsr2musicxml/

```

Some passes are named translators (converters could have been used), and others are not. In `src/passes/mxsr2msr/`, class `mxsr2msrSkeletonBuilder` does not translate MusicXML data to another full representation: it merely creates a skeleton containing voices, are are empty:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll passes/mxsr2msr/
2 total 1808
3  0 drwxr-xr-x   8 jacquesmenu  staff    256 Jun  25  05:47:41 2021 ./
4  0 drwxr-xr-x  16 jacquesmenu  staff    512 May  26  08:20:55 2021 ../
5 96 -rw-r--r--@  1 jacquesmenu  staff  48389 Jun  21  07:43:20 2021 mxsr2msr0ah.cpp
6 40 -rw-r--r--@  1 jacquesmenu  staff  20327 Jun  16  10:41:37 2021 mxsr2msr0ah.h
7 192 -rw-r--r--@  1 jacquesmenu  staff  97896 Jun  25  08:58:38 2021
   mxsr2msrSkeletonBuilder.cpp
8 48 -rw-r--r--@  1 jacquesmenu  staff  20942 Jun  25  07:36:29 2021
   mxsr2msrSkeletonBuilder.h
9 1280 -rw-r--r--@  1 jacquesmenu  staff 651474 Jun  25  07:49:52 2021 mxsr2msrTranslator.cpp
10 152 -rw-r--r--@  1 jacquesmenu  staff  77039 Jun  21  07:43:20 2021 mxsr2msrTranslator.h

```

The passes functionality is available as functions in `*Interface.*`:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > look Interface
2 ./representations/msr/msrInterface.cpp
3 ./representations/msr/msrInterface.h
4 ./representations/lpsr/lpsrInterface.cpp
5 ./representations/lpsr/lpsrInterface.h
6 ./representations/bsr/bsrInterface.h
7 ./representations/bsr/bsrInterface.cpp
8 ./passes/mxsr2musicxml/mxsr2musicxmlTranlatorInterface.h
9 ./passes/mxsr2musicxml/mxsr2musicxmlTranlatorInterface.cpp
10 ./passes/bsr2bsr/bsr2bsrFinalizerInterface.h
11 ./passes/bsr2bsr/bsr2bsrFinalizerInterface.cpp
12 ./passes/msr2mxsr/msr2mxsrInterface.cpp
13 ./passes/msr2mxsr/msr2mxsrInterface.h
14 ./passes/mxsr2msr/mxsr2msrSkeletonBuilderInterface.h
15 ./passes/mxsr2msr/mxsr2msrTranslatorInterface.cpp
16 ./passes/mxsr2msr/mxsr2msrTranslatorInterface.h
17 ./passes/mxsr2msr/mxsr2msrSkeletonBuilderInterface.cpp
18 ./passes/msr2msr/msr2msrInterface.h
19 ./passes/msr2msr/msr2msrInterface.cpp
20 ./passes/lpsr2lilypond/lpsr2lilypondInterface.h
21 ./passes/lpsr2lilypond/lpsr2lilypondInterface.cpp
22 ./passes/msr2lpsr/msr2lpsrInterface.cpp
23 ./passes/msr2lpsr/msr2lpsrInterface.h
24 ./passes/bsr2braille/bsr2brailleTranslatorInterface.h
25 ./passes/bsr2braille/bsr2brailleTranslatorInterface.cpp

```

```

26 ./passes/msr2bsr/msr2bsrInterface.h
27 ./passes/msr2bsr/msr2bsrInterface.cpp
28 ./passes/musicxml2mxsr/musicxml2mxsrInterface.h
29 ./passes/musicxml2mxsr/musicxml2mxsrInterface.cpp
30 ./passes/mxsr2guido/mxsr2guidoTranlatorInterface.h
31 ./passes/mxsr2guido/mxsr2guidoTranlatorInterface.cpp
32 ./converters/msr2guido/msr2guidoInterface.h
33 ./converters/msr2guido/msr2guidoInterface.cpp
34 ./converters/msr2braille/msr2brailleInterface.h
35 ./converters/msr2braille/msr2brailleInterface.cpp
36 ./converters/msdl2braille/msdl2brailleInterface.h
37 ./converters/msdl2braille/msdl2brailleInterface.cpp
38 ./converters/msdl2guido/msdl2guidoInterface.cpp
39 ./converters/msdl2guido/msdl2guidoInterface.h
40 ./converters/msdl2musicxml/msdl2musicxmlInterface.h
41 ./converters/msdl2musicxml/msdl2musicxmlInterface.cpp
42 ./converters/msdl2lilypond/msdl2lilypondInterface.h
43 ./converters/msdl2lilypond/msdl2lilypondInterface.cpp
44 ./converters/musicxml2braille/musicxml2brailleInterface.cpp
45 ./converters/musicxml2braille/musicxml2brailleInterface.h
46 ./converters/msr2lilypond/msr2lilypondInterface.cpp
47 ./converters/msr2lilypond/msr2lilypondInterface.h
48 ./converters/msr2musicxml/msr2musicxmlInterface.cpp
49 ./converters/msr2musicxml/msr2musicxmlInterface.h
50 ./converters/musicxml2musicxml/musicxml2musicxmlInterface.h
51 ./converters/musicxml2musicxml/musicxml2musicxmlInterface.cpp
52 ./converters/musicxml2lilypond/musicxml2lilypondInterface.h
53 ./converters/musicxml2lilypond/musicxml2lilypondInterface.cpp
54 ./converters/musicxml2guido/musicxml2guidoInterface.cpp
55 ./converters/musicxml2guido/musicxml2guidoInterface.h

```

## 5.5 Generators

A generator is a multi-pass command line tool that creates an output from scratch, without reading anything. All of them use `src/formatsgeneration/multiGeneration/multiGeneration.h/.cpp` to offer a set of output formats:

- `src/clisamples/Mikrokosmos3Wandering.cpp`  
creates a score for this Bartok piece in various forms, depending on the options. It has been used to check the MSR API's:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formatsgeneration/
   multiGeneration/
2 total 56
3  0 drwxr-xr-x  4 jacquesmenu  staff    128 Apr 22 15:49:16 2021 ./
4  0 drwxr-xr-x 10 jacquesmenu  staff    320 May 26 08:20:55 2021 ../
5 40 -rw-r--r--@ 1 jacquesmenu  staff  16774 Jun  6 06:38:55 2021 multiGeneration0ah.
   cpp
6 16 -rw-r--r--@ 1 jacquesmenu  staff   6750 Jun  6 06:38:55 2021 mfMultiGeneration0ah
   .h

```

For example:

```

1 jacquesmenu@macmini: ~ > Mikrokosmos3Wandering -lilypond -a
2 What LilyPondIssue34 does:
3
4 This multi-pass generator creates a textual representation
5 of the LilyPondIssue34 score.
6 It basically performs 4 passes when generating LilyPond output output:
7
8 Pass 1: generate a first MSR for the LilyPondIssue34 score
9 Pass 2: converts the first MSR a second MSR;

```

```

10      Pass 3:  converts the second MSR into a
11              LilyPond Score Representation (LPSR);
12      Pass 4:  converts the LPSR to LilyPond code
13              and writes it to standard output.
14
15      Other passes are performed according to the options, such as
16      displaying views of the internal data or printing a summary of the score.
17
18      The activity log and warning/error messages go to standard error.

```

- `src/clisamples/LilyPondIssue34.cpp` creates a score for the LilyPond issue #34 issue, also in various forms;

```

1 jacquesmenu@macmini: ~ > LilyPondIssue34 -musicxml -a
2 What LilyPondIssue34 does:
3
4      This multi-pass generator creates a textual representation
5      of the LilyPondIssue34 score.
6      It basically performs 4 passes when generating MusicXML output output:
7
8      Pass 1:  generate a first MSR for the LilyPondIssue34 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.

```

## 5.6 Converters

The MusicFormats converters chain passes into a sequence, each pass reading the input or the format produced by the preceeding one:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll converters/
2 total 32
3 0 drwxr-xr-x 17 jacquesmenu staff 544 May 26 08:20:55 2021 ./
4 0 drwxr-xr-x 13 jacquesmenu staff 416 Jun 17 17:16:37 2021 ../
5 24 -rw-r--r-- 1 jacquesmenu staff 10244 Jun 18 10:34:45 2021 .DS_Store
6 0 drwxr-xr-x 8 jacquesmenu staff 256 May 26 08:20:55 2021 msdl2braille/
7 0 drwxr-xr-x 8 jacquesmenu staff 256 May 26 08:20:55 2021 msdl2guido/
8 0 drwxr-xr-x 8 jacquesmenu staff 256 May 26 08:20:55 2021 msdl2lilypond/
9 0 drwxr-xr-x 8 jacquesmenu staff 256 May 26 08:20:55 2021 msdl2musicxml/
10 0 drwxr-xr-x 8 jacquesmenu staff 256 May 26 08:20:55 2021 msdlconverter/
11 0 drwxr-xr-x 8 jacquesmenu staff 256 May 26 08:20:55 2021 msr2braille/
12 0 drwxr-xr-x 8 jacquesmenu staff 256 May 26 08:20:55 2021 msr2guido/
13 0 drwxr-xr-x 8 jacquesmenu staff 256 May 26 08:20:55 2021 msr2lilypond/
14 0 drwxr-xr-x 8 jacquesmenu staff 256 May 26 08:20:55 2021 msr2musicxml/
15 0 drwxr-xr-x 4 jacquesmenu staff 128 May 26 08:20:55 2021 musicxml2braille/
16 0 drwxr-xr-x 4 jacquesmenu staff 128 May 26 08:20:55 2021 musicxml2guido/
17 0 drwxr-xr-x 4 jacquesmenu staff 128 May 26 08:20:55 2021 musicxml2lilypond/
18 0 drwxr-xr-x 4 jacquesmenu staff 128 May 26 08:20:55 2021 musicxml2musicxml/

```

## 5.7 Running a service

When a MusicFormats service is *run* from the command line or through an API function, an instance of class `mfServiceRunData` is created.

This class is defined in `src/mflibrary/mfServiceRunData.h/.cpp` to hold data specific to the run. They are global data, but don't belong to the regular, invariant data contained in the library, such as the notes pitches in various languages:

```

1 class EXP mfServiceRunData : public smartable
2 {
3     public:
4
5         // creation
6         // -----
7
8         static SMARTP<mfServiceRunData> create (const std::string& serviceName);
9
10        static SMARTP<mfServiceRunData> create (
11            const std::string& serviceName,
12            int                argc,
13            char*              argv[]);
14
15        static SMARTP<mfServiceRunData> create (
16            const std::string&      serviceName,
17            mfOptionsAndArguments& optionsAndArguments);
18
19    public:
20
21        // constructors/destructor
22        // -----
23
24        mfServiceRunData (const std::string& serviceName);
25
26        mfServiceRunData (
27            const std::string& serviceName,
28            int                argc,
29            char*              argv[]);
30
31        mfServiceRunData (
32            const std::string&      serviceName,
33            mfOptionsAndArguments& optionsAndArguments);
34
35        virtual ~mfServiceRunData ();
36
37        // .. .. .
38
39    private:
40
41        // private fields
42        // -----
43
44        // service name
45        std::string      fServiceName;
46
47        // conversion date
48        std::string      fRunDateFull;
49        std::string      fRunDateYYYYMMDD;
50
51        // conversion command
52        std::string      fCommandAsSupplied;
53
54        std::string      fCommandWithLongOptionsNames;
55        std::string      fCommandWithShortOptionsNames;
56
57        // options and arguments
58        mfOptionsAndArguments
59            fOptionsAndArguments;
60
61        // command line
62        std::string      fCommandLineAsSupplied;
63

```

```

64 // input source
65 std::string      fInputSourceName;
66 };

```

The various constructors are used depending on the way the service is run.

For example, if is created this way in `src/clisamples/xml2ly.cpp`:

```

1 int main (int argc, char* argv[])
2 {
3     // setup signals catching
4     // -----
5
6     // JMI catchSignals ();
7
8     // the service name
9     // -----
10
11     std::string serviceName = argv [0];
12
13     // create the global output and log indented streams
14     // -----
15
16     createTheGlobalIndentedOstreams (std::cout, std::cerr);
17
18     // create the global run data
19     // -----
20
21     gGlobalServiceRunData =
22         mfServiceRunData::create (serviceName);
23
24     // ... ..
25 }

```

Then the various run data can be accessed easily:

```

1 std::string
2     inputSourceName =
3     gGlobalServiceRunData->getInputSourceName ();

```

The run data is used for example in class `lpsrScore`, defined in `src/formats/lpsr//lpsrScores.h/.cpp`:

```

1 lpsrScore::lpsrScore (
2     int          inputLineNumber,
3     const S_msrScore&   theMsrScore,
4     const S_mfcMultiComponent& multiComponent)
5     : lpsrElement (inputLineNumber)
6 {
7     // ...
8
9     fMultiComponent = multiComponent;
10
11     // should the initial comments about the service and the options used
12     // be generated?
13     if (gGlobalLpsr2lilypondOahGroup->getXml2lyInfos ()) {
14         // create the 'generated by' comment
15         {
16             std::stringstream s;
17
18             s <<
19                 "Generated by " <<
20                 gGlobalOahOahGroup->getOahOahGroupServiceName () <<
21                 ' ' <<
22                 getGlobalMusicFormatsVersionNumberAndDate () <<
23                 std::endl <<
24

```

```
25     "% on " <<
26     gGlobalServiceRunData->getRunDateFull () <<
27     std::endl <<
28
29     "% from ";
30
31     std::string inputSourceName =
32     gGlobalServiceRunData->getInputSourceName ();
33
34     if (inputSourceName == "-") {
35         s << "standard input";
36     }
37     else {
38         s << "\"" << inputSourceName << "\"";
39     }
40
41     fInputSourceNameComment =
42     lpsrComment::create (
43         inputLineNumber,
44         s.str (),
45         lpsrCommentGapAfterwardsKind::kCommentGapAfterwardsYes);
46 }
47 }
48
49 // ...
50 }
```

## Chapter 6

# Command line samples

The `src/clisamples` folder contains example of the use of MusicFormats in CLI tools. They are out of the library proper, and built with a specific Makefile:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > ll samples/
2 total 320
3  0 drwxr-xr-x  16 jacquesmenu  staff    512 Jun 29 09:59:07 2021 ./
4  0 drwxr-xr-x  28 jacquesmenu  staff    896 Jul  1 05:37:35 2021 ../
5 16 -rw-r--r--   1 jacquesmenu  staff   6148 May 26 08:20:55 2021 .DS_Store
6  8 -rw-r--r--   1 jacquesmenu  staff    116 Apr 22 15:49:06 2021 .gitignore
7 40 -rw-r--r--@  1 jacquesmenu  staff  18344 Jun 29 11:05:18 2021 LilyPondIssue34.cpp
8  8 -rw-r--r--@  1 jacquesmenu  staff   2101 Jun 29 10:00:56 2021 Makefile
9 40 -rw-r--r--@  1 jacquesmenu  staff  18362 Jun 29 11:05:10 2021 Mikrokosmos3Wandering.cpp
10 24 -rw-r--r--@  1 jacquesmenu  staff  10017 May 31 11:12:12 2021 MusicAndHarmonies.cpp
11  8 -rw-r--r--@  1 jacquesmenu  staff   3117 May 31 11:17:27 2021 libMultipleInitsTest.cpp
12 48 -rw-r--r--@  1 jacquesmenu  staff  21459 Jun 29 11:05:02 2021 msdlconverter.cpp
13  8 -rw-r--r--@  1 jacquesmenu  staff    898 May 31 11:15:59 2021 musicformatsversion.cpp
14 24 -rw-r--r--@  1 jacquesmenu  staff   8642 Jun 28 07:42:57 2021 xml2Any.cpp
15 24 -rw-r--r--@  1 jacquesmenu  staff  10085 Jul  1 06:22:13 2021 xml2brl.cpp
16 24 -rw-r--r--@  1 jacquesmenu  staff  10519 Jul  1 06:22:50 2021 xml2gmn.cpp
17 24 -rw-r--r--@  1 jacquesmenu  staff  10320 Jul  1 00:09:51 2021 xml2ly.cpp
18 24 -rw-r--r--@  1 jacquesmenu  staff  10473 Jul  1 06:21:10 2021 xml2xml.cpp
```

All the `*.cpp` files contain a `main ()` function using the interfaces for their purpose. Among them:

- `libMultipleInitsTest.cpp` is a maintenance tool to check that the MusicFormats library is not initialized more than once;
- `MusicAndHarmonies.cpp` creates a score at random with harmonies in it;
- `Mikrokosmos3Wandering.cpp` and `LilyPondIssue34.cpp` are generators;
- `xml2Any.cpp` uses the `oahOptionsVector` way to supply arguments instead of `arg/argv`;
- `xml2ly`, `xml2brl`, `xml2xml` and `xml2gmn` are converters from MusicXML to other formats;
- `msdlconverter.cpp` is the MSDL converter.

## Chapter 7

# Warning and errors (WAE)

Warning and errors in MusicFormats are handled with a set of functions defined in the `wae` folder.

`mfException` and context-specific exceptions are defined in `src/wae/waeExceptions`, such as:

```
1 // -----
2 class EXP mf0ahException: public mfException
3 {
4     public:
5         mf0ahException (std::string const& exceptionDescription) throw ()
6             : mfException (exceptionDescription)
7         {}
8 };
9 typedef SMARTP<mf0ahException> S_mf0ahException;
```

A typical use of exceptions in `src/passes/lpsr2lilypond/lpsr2lilypondInterface.cpp` is:

```
1 // convert the LPSR score to LilyPond code
2 try {
3     translateLpsrToLilypond (
4         theLpsrScore,
5         gGlobalMsr0ahGroup,
6         gGlobalLpsr0ahGroup,
7         passNumber,
8         passDescription,
9         lilypondStandardOutputStream);
10 }
11 catch (lpsr2lilypondException& e) {
12     mfDisplayException (e, gOutputStream);
13     return;
14 }
15 catch (std::exception& e) {
16     mfDisplayException (e, gOutputStream);
17     return;
18 }
```

One finds in `src/wae/waeEnableAbortToDebugErrors.h` the `ABORT_TO_DEBUG_ERRORS` macro to help debugging the code base:

```
1 // comment the following definition if abort on internal errors is desired
2 // CAUTION: DON'T USE THIS IN PRODUCTION CODE,
3 // since that could kill a session on a \Web\ server, for example
4
5 #ifndef ABORT_TO_DEBUG_ERRORS
6     #define ABORT_TO_DEBUG_ERRORS
7 #endif
```



## Chapter 8

# The trace facility

MusicFormats is instrumented with an optionnal, full-fledged trace facility, with numerous options to display what is going on when using the library. One can build the library with or without trace, which applies to the whole code base.

### 8.1 Activating the trace

Tracing is controlled by `TRACING_IS_ENABLED`, defined or nor in `src/oah/oahEnableTracingIfDesired.h`:

```
1 #ifndef __enableTracingIfDesired__
2 #define __enableTracingIfDesired__
3
4 #ifndef TRACING_IS_ENABLED
5     // comment the following definition if no tracing is desired
6     #define TRACING_IS_ENABLED
7 #endif
8
9 #endif
```

This file should be included when the trace facility is used:

```
1 #include "oahEnableTracingIfDesired.h"
2 #ifdef TRACING_IS_ENABLED
3     #include "tracingOah.h"
4 #endif
```

The files `src/oah/tracingOah.h/.cpp` contain the options to the trace facility itself.

Be sure to build MusicFormats with `TRACING_IS_ENABLED` both active and commented out before creating a new `v*` version branch, to check that variables scopes are fine.

For example, `xml2ly -insider -help-tracexml2lyoption -insider` produces:

```
1 menu@macbookprojm > xml2ly -insider -help-trace
2 --- Help for group "OAH Trace" ---
3 OAH Trace (-ht, -help-trace) (use this option to show this group)
4     There are trace options transversal to the successive passes,
5     showing what's going on in the various translation activities.
6     They're provided as a help for the maintenance of MusicFormats,
7     as well as for the curious.
8     The options in this group can be quite verbose, use them with small input data!
9     All of them imply '-trace-passes, -tpasses'.
10 -----
11 Options handling trace      (-htoh, -help-trace-options-handling):
12 -toah, -trace-oah
```

```

13         Write a trace of options and help handling to standard error.
14     -toahd, -trace-oah-details
15         Write a trace of options and help handling with more details to standard error
16     .
17     Score to voices          (-htstv, -help-trace-score-to-voices):
18     -t<SHORT_NAME>, -trace-<LONG_NAME>
19         Trace SHORT_NAME/LONG_NAME in books to voices.
20     The 10 known SHORT_NAMES are:
21         book, scores, pgroups, pgroupsd, parts, staves, st, schanges,
22     .
23     The 10 known LONG_NAMES are:
24     -books, -scores, -part-groups, -part-groups-details,
25     -parts, -staves, -staff-details, -staff-changes, -voices and
26     -voices-details.
27 ... ..

```

## 8.2 Trace categories

## 8.3 Using traces in practise

In `src/passes/lpsr2lilypond/lpsr2lilypondTranslator.cpp`, the trace for the generation of LilyPond code for a regular note in a measure is produced by:

```

1 void lpsr2lilypondTranslator::generateCodeForNoteRegularInMeasure (
2     const S_msrNote& note)
3 {
4     int inputLineNumber =
5         note->getInputLineNumber ();
6
7     #ifdef TRACING_IS_ENABLED
8     if (gGlobalTracingOahGroup->getTraceNotes ()) {
9         std::stringstream s;
10
11         s <<
12         std::endl <<
13         "% --> generating code for noteRegularInMeasure " <<
14         note->asString () <<
15         ", line " << inputLineNumber <<
16         std::endl;
17
18         gLogStream          << s.str ();
19         fLilypondCodeStream << s.str ();
20     }
21     #endif

```

## 8.4 Debugging traces handling

If case there is a null pointer in a case such as:

```

1 gGlobalMsrOahGroup->getUseFilenameAsWorkCreditTypeTitle ()

```

the way to go is to:

- locate `gGlobalMxsr2msrOahGroup` in the `*.h` it is declared in;
- check that the creation method in the same, such as `createGlobalMxsr2msrOahGroup ()`, is called in the `*InsiderHandler.cpp` file for the tool that crashed, which may require including that `.h` header in `InsiderHandler.cpp`.

## Chapter 9

# Multi-lingual support

MusicFormats supports multiple languages in various areas:

- note pitches names in MSR, LPSR and generated LilyPond code;
- LilyPond chord names;
- reserved keywords in MSDL.

## Chapter 10

# Textual input and output

### 10.1 Indented output streams

To meet the need of indented output to produce , we got inspiration from <https://stackoverflow.com/questions/2212776/overload-handling-of-stdendl>.

This leads to class `mfIndentedOstream`, defined in `src/utilities/mfIndentedTextOutput.h/.cpp`:

```
1 class EXP mfIndentedOstream: public std::ostream, public smartable
```

The key to this lies in the overloaded method `mfIndentedStreamBuf::sync ()`:

```
1 // -----
2 int mfIndentedStreamBuf::sync ()
3 {
4     // When we sync the stream with fOutputStream:
5     // 1) output the indentation then the buffer
6     // 2) reset the buffer
7     // 3) flush the actual output stream we are using.
8
9     unsigned int strSize = str ().size ();
10
11     // fetch the last non-space character in the buffer
12     // caution: the '\n' is present as the last character!
13     size_t found = str ().find_last_not_of ( ' ', strSize - 2);
14
15     // this can be uncommented to see low level informations
16     // fOutputStream << "% strSize: " << strSize << ", found: " << found << '\n';
17
18     // output the indenter
19     fOutputStream << fOutputIndenter;
20
21     // output the buffer
22     if (found == strSize - 3) {
23         // don't output the trailing spaces, but output the end of line
24         fOutputStream << str ().substr (0, found + 1) << '\n';
25     }
26     else {
27         // output the whole buffer
28         fOutputStream << str ();
29     }
30
31     // reset the buffer
32     str ("");
33
34     // flush the output stream
35     fOutputStream.flush ();
36
```

```

37     return 0;
38 }

```

Such indented output streams are used for nearly all of the output produced by MusicFormats, except for Braille which follows its own rules for indentation of cell lines.

## 10.2 Creating indented output streams

Such indented output streams are passed by reference to various methods which only know of `ostream`, among them:

```

1 void print (std::ostream& os) const override;

```

All those methods manipulate `mfIndentedOstream` instances seamlessly, not knowing their actual type. This is needed for the use of MusicFormats by applications through the API and not as a service. In particular, Web sites also know only of regular output streams.

So when and where are `mfIndentedOstream` instances created?

Functions `translateLpsrToLilypondWithHandler ()`, in `src/passes/lpsr2lilypond/lpsr2lilypondInterface`, creates one depending on whether it writes the LilyPond code to standard output or to a file.

The parameters to this function are:

```

1 EXP void translateLpsrToLilypondWithHandler (
2     const S_lpsrScore& theLpsrScore,
3     const S_msrOahGroup& msrOpts,
4     const S_lpsrOahGroup& lpsrOpts,
5     const std::string& passNumber,
6     const std::string& passDescription,
7     S_oahHandler handler,
8     std::ostream& out,
9     std::ostream& err)

```

In order to have a global current indentation, MusicFormats uses global variable `gIndenter`, because it should otherwise be passed over to many methods throughout the code base. It is defined in `src/mfutilities/mfIndentedOstream`.

```

1 #define gIndenter mfOutputIndenter::gGlobalOStreamIndenter

```

When writing to standard output, the indented output stream is constructed above the caller-supplied `out`:

```

1 // create an indented output stream for the LilyPond code
2 // to be written to outputFileStream
3 mfIndentedOstream
4     lilypondStandardOutputStream (
5         out,
6         gIndenter);
7
8 // convert the LPSR score to LilyPond code
9 try {
10     translateLpsrToLilypond (
11         theLpsrScore,
12         gGlobalMsrOahGroup,
13         gGlobalLpsrOahGroup,
14         passNumber,
15         passDescription,
16         lilypondStandardOutputStream);
17 }

```

When writing to a file, an `std::ofstream` is instantiated to write to the file given by its name, and the indented output stream is constructed above the latter:

```

1      std::ofstream
2      outputFileStream (
3          outputFileName.c_str (),
4          std::ofstream::out);
5
6      // create an indented output stream for the LilyPond code
7      // to be written to outputFileStream
8      mfIndentedOstream
9      lilypondFileOutputStream (
10         outputFileStream,
11         gIndenter);
12
13     // convert the LPSR score to LilyPond code
14     try {
15         translateLpsrToLilypond (
16             theLpsrScore,
17             gGlobalMsrOahGroup,
18             gGlobalLpsrOahGroup,
19             passNumber,
20             passDescription,
21             lilypondFileOutputStream);
22     }

```

The code that uses MusicFormats thus does not have to care for indented streams instantiation: this is done behind the scene by the library.

## 10.3 Indenting the output

Indenting the output is handled with a single variable defined in `src/mfutilities/mfIndentedTextOutput.h`. This sharing of a global variable is needed to produce orderly output, since many parts of the MusicFormats library can contribute to it:

```

1  // useful shortcut macros
2  #define gIndenter mfOutputIndenter::gGlobalOStreamIndenter
3  #define gTab      mfOutputIndenter::gGlobalOStreamIndenter.getSpacer ()

```

A typical sequence to produce indented output is:

```

1  void msrTransposition::print (std::ostream& os) const
2  {
3      const int fieldWidth = 22;
4
5      os <<
6          "Transposition" <<
7          ", line " << fInputLineNumber <<
8          std::endl;
9
10     ++gIndenter;
11
12     os << std::left <<
13         std::setw (fieldWidth) <<
14         "fTranspositionDiatonic" << " = " << fTranspositionDiatonic <<
15         std::endl <<
16         std::setw (fieldWidth) <<
17         "fTranspositionChromatic" << " = " << fTranspositionChromatic <<
18         std::endl <<
19         std::setw (fieldWidth) <<
20         "fTranspositionOctaveChange" << " = " << fTranspositionOctaveChange <<
21         std::endl <<
22         std::setw (fieldWidth) <<

```

```

23     "fTranspositionDouble" << " = " << fTranspositionDouble <<
24     std::endl << std::endl;
25
26     --gIndenter;
27 }

```

Note that the new value of `gIndenter` after `++gIndenter` and `--gIndenter` is taken into account only at the *next* end-of-line – the output of lines 5 to 8 above is indented one level less than the various field values output afterwards.

There can be indentation issues, in which the user gets messages like:

```

1 % ### Indentation has become negative: -1

```

To debug this:

- activate the debugging for the indenter, uncommenting this definition in `src/components/mfIndentedTextOutput`

```

1 //-----
2 // #define DEBUG_INDENTER

```

- activate abortion on errors in `src/wae/waeEnableAbortToDebugErrors.h`:

```

1 // comment the following definition if abort on internal errors is desired
2 // CAUTION: DON'T USE THIS IN PRODUCTION CODE,
3 // since that could kill a session on a web server, for example
4
5 #ifndef ABORT_TO_DEBUG_ERRORS
6     // #define ABORT_TO_DEBUG_ERRORS
7 #endif

```

## 10.4 Printing descriptions

There is a standard set of methods to print the contents of the descriptions in `MusicFormats` to standard output, depending on the granularity of the information to be displayed:

```

1 void print (std::ostream& os) const override;
2
3 std::string asString () const override;
4 std::string asStringShort () const override;

```

There are also more specific methods such as:

```

1 void printFull (std::ostream& os) const override;
2
3 void printSummary (std::ostream& os) const override;

```

Note that:

- virtual method `asString ()` produces a rather condensed view of the data to be displayed as part of a single line;
- virtual method `print ()` may produce its output on multiples lines, which always ends with an end of line.

Most classes in `MusicFormats` can be printed with the `<<` operator. Since `MusicFormats` is a large, a test is done for nullity, which is safer and easier for the applications that use it:

```

1 std::ostream& operator << (std::ostream& os, const S_msrElement& elt)
2 {
3     if (elt) {
4         elt->print (os);
5     }
6     else {
7         os << "[NONE]" << std::endl;
8     }
9
10    return os;
11 }

```

In simple cases, virtual method `print ()` merely calls virtual method `asString ()`:

```

1 void msrElement::print (std::ostream& os) const
2 {
3     os << asString () << std::endl;
4 }

```

All `asString ()` methods produce an output of the form [...], in order to facilitate selecting the whole with a double click to help the user, since such output can be nested:

```

1 std::string msrTransposition::asString () const
2 {
3     std::stringstream s;
4
5     s <<
6     "[Transposition" <<
7     ", fTranspositionDiatonic = " << fTranspositionDiatonic <<
8     ", fTranspositionChromatic = " << fTranspositionChromatic <<
9     ", fTranspositionOctaveChange = " << fTranspositionOctaveChange <<
10    ", fTranspositionDouble = " << fTranspositionDouble <<
11    ", line " << fInputLineNumber <<
12    ']' ;
13
14    return s.str ();
15 }

```

A typical sequence to produce indented output is:

```

1 void msrTransposition::print (std::ostream& os) const
2 {
3     const int fieldWidth = 22;
4
5     os <<
6     "Transposition" <<
7     ", line " << fInputLineNumber <<
8     std::endl;
9
10    ++gIndenter;
11
12    os << std::left <<
13    std::setw (fieldWidth) <<
14    "fTranspositionDiatonic" << " = " << fTranspositionDiatonic <<
15    std::endl <<
16    std::setw (fieldWidth) <<
17    "fTranspositionChromatic" << " = " << fTranspositionChromatic <<
18    std::endl <<
19    std::setw (fieldWidth) <<
20    "fTranspositionOctaveChange" << " = " << fTranspositionOctaveChange <<
21    std::endl <<
22    std::setw (fieldWidth) <<
23    "fTranspositionDouble" << " = " << fTranspositionDouble <<
24    std::endl << std::endl;
25 }

```



```
26 |   --gIndenter;  
27 | }
```

The main indented output streams are:

```
1 | #define gOutputStream *gGlobalOutputIndentedOstream  
2 | #define gLogStream    *gGlobalLogIndentedOstream
```

## Chapter 11

# Binary data output

Binary data output is done for Braille

# Chapter 12

## CPU measurements

Option `-cpu` displays the time spent in the successive passes, such as:

Activity	Description	Kind	CPU (sec)
	Handle the options and arguments from <code>argc/argv</code>	mandatory	0.01187
Pass 1	Create an MXSR reading a MusicXML file	mandatory	0.00471
Pass 2a	Create an MSR skeleton from the MXSR	mandatory	0.00222
Pass 2b	Populate the MSR skeleton from MusicXML data	mandatory	0.00405
Pass 4	Convert the MSR into an LPSR	mandatory	0.00137
Pass 5	Convert the LPSR score to LilyPond code	mandatory	0.00136
Total (sec)	Mandatory    Optional		
0.02558	0.02558    0.00000		

These numbers are for the CPU only, not including input and output tasks. The time spent in options handling is roughly always the same on a given machine.

Class `mfTimingItemsList`, defined in `src/utilities/mfTiming.h/.cpp`, provides:

```
1 class EXP mfTimingItemsList {
2     // ... ..
3
4     public:
5
6         // global variable for general use
7         // -----
8
9         static mfTimingItemsList          gGlobalTimingItemsList;
10
11     public:
12
13         // public services
14         // -----
15
16         // add an item
17         void          appendTimingItem (
18                     std::string          activity,
19                     std::string          description,
20                     mfTimingItemKind kind,
21                     clock_t              startClock,
22                     clock_t              endClock);
23
24         // ... ..
25 }
```

Functions `translateMsrToLpsrScore ()` in `src/passes/msr2lpsr/msr2lpsrInterface.cpp` measures time to perform the conversion this way:

```

1 S_lpsrScore translateMsrToLpsr (
2     const S_msrScore&    originalMsrScore,
3     const S_msr0ahGroup& msrOpts,
4     const S_lpsr0ahGroup& lpsrOpts,
5     std::string          passNumber,
6     std::string          passDescription,
7     const S_mfcMultiComponent& multiComponent)
8 {
9     if (gGlobalLpsr2lilypond0ahGroup->getNoLilypondCode ()) {
10         gLogStream <<
11             "Option '-nolpc, -no-lilypond-code' is set, no LPSR is created" <<
12             std::endl;
13
14         return nullptr;
15     }
16
17     // sanity check
18     mfAssert (
19         __FILE__, __LINE__,
20         originalMsrScore != nullptr,
21         "originalMsrScore is null");
22
23     // start the clock
24     clock_t startClock = clock ();
25
26 #ifdef TRACING_IS_ENABLED
27     if (gGlobal0ahEarlyOptions.getEarlyTracePasses ()) {
28         std::string separator =
29             "%-----";
30
31         gLogStream <<
32             std::endl <<
33             separator <<
34             std::endl <<
35             gTab <<
36             passNumber << ": " << passDescription <<
37             std::endl <<
38             separator <<
39             std::endl;
40     }
41 #endif
42
43     // create an msr2lpsrTranslator
44     msr2lpsrTranslator
45         translator (
46             originalMsrScore);
47
48     // build the LPSR score
49     S_lpsrScore
50         resultingLpsr =
51         translator.translateMsrToLpsr (
52             originalMsrScore,
53             multiComponent);
54
55     clock_t endClock = clock ();
56
57     // register time spent
58     mfTimingItemsList::gGlobalTimingItemsList.appendTimingItem (
59         passNumber,
60         passDescription,
61         mfTimingItemKind::kMandatory,
62         startClock,
63         endClock);

```

---

## Part II

# Programming style and conventions

## Chapter 13

# Programming style and conventions

### 13.1 Files naming conventions

Most file names start with an identification of the component they belong to, such as 'oah', 'mxsr', 'msr', 'lpsr', 'lilypond', 'bsr', 'braille', 'xml2ly', 'xml2brl' and msdl.

The ancillary files such as `src/utilities/mfIndentedTextOutput.h/.cpp` follow this rule too, with an mf prefix.

The '\*Oah.\*' files handle the options and help for the corresponding component, such as `'src/passes/msr2msr/msr2msrOah.h/.cpp'`.

The `'src/oah/tracingOah.h/.cpp'`, `src/oah/musicxmlOah.h/.cpp` 'extra' and 'general' prefixes are about the corresponding help groups.

There are a couple of 'global' files not related to any particular component, placed in `src/mfutilities/` with an mf name prefix:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll mfutilities/
2 total 200
3  0 drwxr-xr-x  15 jacquesmenu  staff    480 Oct 22 06:25:57 2021 ./
4  0 drwxr-xr-x  19 jacquesmenu  staff    608 Oct 22 05:29:29 2021 ../
5  8 -rw-r--r--@  1 jacquesmenu  staff   3255 Oct 18 20:22:16 2021 mfBool.cpp
6 16 -rw-r--r--@  1 jacquesmenu  staff   4917 Oct 18 19:56:51 2021 mfBool.h
7  8 -rw-r--r--@  1 jacquesmenu  staff   1336 Oct 15 18:48:10 2021 mfEnumAll.h
8 16 -rw-r--r--@  1 jacquesmenu  staff   7182 Nov  8 13:08:51 2021 mfIndentedTextOutput.cpp
9 16 -rw-r--r--@  1 jacquesmenu  staff   7715 Nov  8 13:08:40 2021 mfIndentedTextOutput.h
10 8 -rw-r--r--@  1 jacquesmenu  staff    889 Oct 15 20:34:47 2021 mfMusicformatsErrorKind.
    cpp
11 8 -rw-r--r--@  1 jacquesmenu  staff    629 Oct 15 20:34:47 2021 mfMusicformatsErrors.h
12 8 -rw-r--r--@  1 jacquesmenu  staff   2541 Nov  5 11:29:25 2021 oahOptionsVector.cpp
13 8 -rw-r--r--@  1 jacquesmenu  staff    972 Oct 15 20:16:51 2021 oahBasicTypes.h
14 64 -rw-r--r--@  1 jacquesmenu  staff  29773 Oct 15 18:48:10 2021 mfStringsHandling.cpp
15 16 -rw-r--r--@  1 jacquesmenu  staff   6269 Oct 15 18:55:46 2021 mfStringsHandling.h
16 16 -rw-r--r--@  1 jacquesmenu  staff   5028 Oct  7 20:03:27 2021 mfTiming.cpp
17 8 -rw-r--r--@  1 jacquesmenu  staff   3726 Oct  8 08:21:09 2021 mfTiming.h
```

The files `*Elements.h/.cpp` contain base classes to variants, such as `src/formats/lpsr//lpsrElements.h/.cpp`, whose `lpsrElement` class is used in a number of other files:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public lpsrElement' *
2 formats/lpsr/lpsrStaves.h:29:class EXP lpsrNewStaffgroupBlock : public lpsrElement
3 formats/lpsr/lpsrStaves.h:87:class EXP lpsrNewStaffTuningBlock : public lpsrElement
4 formats/lpsr/lpsrStaves.h:150:class EXP lpsrNewStaffBlock : public lpsrElement
5 formats/lpsr/lpsrStaves.h:208:class EXP lpsrStaffBlock : public lpsrElement
6 formats/lpsr/lpsrVariables.h:27:class EXP lpsrVariableUseCommand : public lpsrElement
7 formats/lpsr/lpsrScores.h:35:class EXP lpsrScore : public lpsrElement
8 formats/lpsr/lpsrBarNumbers.h:26:class EXP lpsrBarNumberCheck : public lpsrElement
9 formats/lpsr/lpsrBarNumbers.h:85:class EXP lpsrBarCommand : public lpsrElement
10 formats/lpsr/lpsrLyrics.h:31:class EXP lpsrNewLyricsBlock : public lpsrElement
11 formats/lpsr/lpsrComments.h:25:class EXP lpsrComment : public lpsrElement
12 formats/lpsr/lpsrVoices.h:29:class EXP lpsrUseVoiceCommand : public lpsrElement
13 formats/lpsr/lpsrParts.h:27:class EXP lpsrPartBlock : public lpsrElement
14 formats/lpsr/lpsrPapers.h:32:class EXP lpsrPaper : public lpsrElement
15 formats/lpsr/lpsrPartGroups.h:29:class EXP lpsrPartGroupBlock : public lpsrElement
16 formats/lpsr/lpsrParallelMusic.h:28:class EXP lpsrParallelMusicBlock : public lpsrElement
17 formats/lpsr/lpsrLayouts.h:23:class EXP lpsrLayout : public lpsrElement
18 formats/lpsr/lpsrHeaders.h:27:class EXP lpsrHeader : public lpsrElement
19 formats/lpsr/lpsrScheme.h:29:class EXP lpsrSchemeVariable : public lpsrElement
20 formats/lpsr/lpsrScheme.h:140:class EXP lpsrSchemeFunction : public lpsrElement
21 formats/lpsr/lpsrBookBlockElements.h:35:class EXP lpsrBookBlockElement : public
    lpsrElement
22 formats/lpsr/lpsrBookBlockElements.h:237:class EXP lpsrBookBlock : public lpsrElement
23 formats/lpsr/lpsrContexts.h:30:class EXP lpsrContext : public lpsrElement
```

There are a number of self-explaning `*BasicTypes.h/.cpp` file names:

```

1 ./formats/msdl/msdlEnumTypes.h
2 ./formats/msdl/msdlEnumTypes.cpp
3 ./formats/msr/msrBasicTypes.cpp
4 ./formats/msr/msrBasicTypes.h
5 ./formats/lpsr/lpsrEnumTypes.cpp
6 ./formats/lpsr/lpsrEnumTypes.h
7 ./formats/bsr/bsrEnumTypes.h
8 ./formats/bsr/bsrEnumTypes.cpp
9 ./oah/oahBasicTypes.h
10 ./oah/oahBasicTypes.cpp
11 ./formatsgeneration/msrGeneration/msrGenerationBasicTypes.cpp
12 ./formatsgeneration/msrGeneration/msrGenerationBasicTypes.h

```

The files are grouped in the `src` folder according to the component they belong to:

- converters
- generators
- interfaces
- oah
- formatsgeneration
- passes
- formats
- utilities
- wae

## 13.2 Adding C++ files

Building MusicFormats relies on `build/CMakeLists.txt` to find the C++ files that should be compiled.

When building MusicFormats with:

```

1 cd build
2 make

```

a `cmake` cache is created in file `build/libdir/CMakeCache.txt`, containing the list of all the C++ files in the library, including those of the embedded `libmusicxml2`.

Adding individual files is fine, but adding new folders in the `src` hierarchy implies to update variable `SRC_FOLDERS` in `build/CMakeLists.txt` accordingly and to remove the `build/libdir/CMakeCache.txt` cache.

Care must be taken when adding a new file on a case insensitive file system: the type case in its name should be what is needed in the first place.

For example, renaming `src/formats/msr/msrBarlines.h` to `src/formats/msr/msrBarLines.h` (this author's experience) leads MusicFormats not to build successfully on Linux if you develop on Windows or MacOS:

- the latter two usually use case insensitive file names (even though one may choose to format as disk to be case-sensitive), but Linux does not;



- the renaming above is not pushed to the repository by `git push` on case insensitive file systems.

The best solution here, both for files and folders names, is to use `'git mv'` to do the renaming instead of the operating system tools:

- `git mv oldName newName`
- `git commit "..."` `-a`
- `git push`

### 13.3 Renaming C++ files

Renaming a C++ file causes `build/libdir/CMakeCache.txt` to be obsolete: it then has to be removed, and the library should be built anew.

For example, this author uses the `rmcache` bash alias to remove the cache:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > type rmcache
2 rmcache is aliased to 'rm /Users/jacquesmenu/musicformats-git-dev/build/libdir/CMakeCache.txt'
```

Running `make` will re-create this cache with the new file name.

Caution has to be taken when a file name case is changed in a case-insensitive development environment such as Windows or MacOS. Cloning MusicFormats in Linux will then fail to find the file under its new name.

In such a case, the following Git command has to be used to actually change the file name in MusicFormats repository:

```
1 git mv -f <old name> <new name>
```

Changing the name of a directory in `src/` should be propagated to `build/CMakeLists.txt`, since this is where the set of files to be compiled is determined, as in:

```
1 if (MSR)
2   set (SRC_FOLDERS ${SRC_FOLDERS} passes/musicxml2mxsr formats/mxsr passes/mxsr2msr
        formats/msr passes/msr2msr formatsgeneration/multiGeneration formatsgeneration/
        msrGeneration generators/Mikrokosmos3Wandering generators/LilyPondIssue34)
3 endif()
```

In such a case, the cache should be removed before building, see `rmcache` above.

### 13.4 Source code layout

The following text-editing conventions are used:

- tabs are not used before the first non-space character in a line, two spaces are used instead;
- the code is not tightly packed: declarations in classes have the members' names aligned vertically, with many spaces before them if needed, and empty lines are used to separate successive activities in methods.

## 13.5 Defensive programming

The code base of `xml2ly` is *defensive programming* oriented, which means that:

- identifiers are explicit and long if needed – only very local ones are short, such as iteration loops indexes;
- the code is organized in sections, with an initial comment documenting what the code does;
- the C++17's `auto` declaration feature is used only for enumeration type `s`, see below. Writing the explicit types in a large code base helps the maintainer mastering the code;
- function `mfAssert ()` is used to perform sanity checks, such as detect a null pointer prior to using it.

The few uses of `auto` declarations are in range-based `for` loops over enumeration type `s`. There the type of the index is explicit from the `Enum*` being enumerated over. For example, in `src/formats/msr/msrBasicTypes.cpp`:

```
1  for (auto e : mfEnumAll<msrHarmonyKind> ()) {  
2      // create the harmony structure  
3      S_msrHarmonyStructure  
4      harmonyStructure =  
5          msrHarmonyStructure::create (  
6              e);  
7  
8      // register it in the map  
9      gGlobalHarmonyStructuresMap [e] =  
10         harmonyStructure;  
11 } // for
```

Class `mfEnumAll` is defined in `src/mfutilities/mfEnumAll.h` as:

```

1  template< typename T >
2  class    mfEnumAll
3  {
4      public:
5
6          class    Iterator
7          {
8              public:
9
10             Iterator (int value)
11                 : fIterationIndex (value)
12                 {}
13
14             T operator* (void) const
15                 { return (T) fIterationIndex; }
16
17             void operator++ (void)
18                 { ++fIterationIndex; }
19
20             Bool operator != (Iterator rhs)
21                 { return fIterationIndex != rhs.fIterationIndex; }
22
23             private:
24
25                 int fIterationIndex;
26
27     };

```

## 13.6 Sanity checks

They are performed to ensure that the formats in `MusicFormats` are consistent, to avoid ugly crashes. An example is:

```

1  // get voice to insert harmonies into
2  S_msrVoice
3      voiceToInsertHarmoniesInto =
4          fCurrentPart->
5              getPartHarmoniesVoice ();
6
7  // sanity check
8  mfAssert (
9      __FILE__, __LINE__,
10      voiceToInsertHarmoniesInto != nullptr,
11      "voiceToInsertHarmoniesInto is null");

```

## 13.7 JMI comments

Comments containing JMI indicates that the code may have to be reconsidered in the future, should a problem arise. They are removed when it becomes obvious that the code is fine. JMI was the acronym for the author's activity as a software contractor long time ago.

## 13.8 Exported symbols

The classes and functions that need to be exported from the MusicFormats library in the Windows meaning are marked as such with an EXP specification:

```
1 class EXP smartable {
2     // ... ..
3 };
```

```
1 EXP S_mxsrOahGroup createGlobalMxsrOahGroup ();
```

EXP is defined in `libmusicxml/src/interface/exports.h` and is non-blank only when building on Windows™:

```
1 #if defined(WIN32) // && !defined (GCC)
2
3 # ifdef MSVC
4 #   pragma warning (disable : 4267)
5 #   pragma warning (disable : 4275)
6 #   pragma warning (disable : 4251)
7 #   pragma warning (disable : 4786)
8 #   pragma warning (disable : 4251)
9 #   pragma warning (disable : 4275)
10 # endif
11
12 # ifdef LIBMUSICXML_EXPORTS
13 #   define EXP __declspec(dllexport)
14
15 # elif defined(LIBMUSICXML_STATIC)
16 #   define EXP
17
18 # else
19 #   define EXP __declspec(dllimport)
20 # endif
21
22 #else
23
24 # ifdef LIBMUSICXML_EXPORTS
25 #   define EXP __attribute__((visibility("default")))
26 # else
27 #   define EXP
28 # endif
29
30 #endif
```

## 13.9 Smart pointers

`libmusicxml2` provides what Dominique Fober named smart pointers, because:

- a smart pointer is an instance of a class that contains the actual pointer in the usual C++ sense;
- the actual pointer is guaranteed to be initialized to `nullptr`;
- garbage collection is implicit, using reference counts.

The definitions are in `libmusicxml/src/lib!smartpointer.h`.

The reference counting is done in class `smartable`:

```

1 class EXP smartable {
2     private:
3         unsigned refCount;
4     public:
5         ///! gives the reference count of the object
6         unsigned refs() const { return refCount; }
7         ///! addReference increments the ref count and checks for refCount overflow
8         void addReference() { refCount++; assert(refCount != 0); }
9         ///! removeReference delete the object when refCount is zero
10        void removeReference() { if (--refCount == 0) delete this; }
11
12    protected:
13        smartable() : refCount(0) {}
14        smartable(const smartable&): refCount(0) {}
15        ///! destructor checks for non-zero refCount
16        virtual ~smartable() { assert (refCount == 0); }
17        smartable& operator=(const smartable&) { return *this; }
18    };
19
20 A smart pointer type is created with template class {\tt SMARTP}, for example:
21 template<class T> class SMARTP {
22     class EXP msrElement : public smartable
23     {
24         // ... ..
25     };
26     typedef SMARTP<msrElement> S_msrElement;

```

Smart pointer type name belonging to Dominique Fober's work can be told from those of MusicFormats by their prefix:

- in libmusicxml2, smart pointer type names start with an 'S', such as \$xmlelement;
- in MusicFormats, they start with 'S\_', such as S\_oahHandler.

Inheriting from class `smartable` is used to create smart pointer types, as in `src/wae/waeHandlers.h`:

```

1 class EXP waeHandler : public smartable
2 {
3     public:
4
5         // creation
6         // -----
7
8         static SMARTP<waeHandler> create ();
9
10    public:
11
12        // constructors/destructor
13        // -----
14
15                waeHandler ();
16
17        virtual    ~waeHandler ();
18
19    public:
20
21        // set and get
22        // -----
23
24    public:
25
26        // public services
27        // -----
28

```

```

29 public:
30
31     // print
32     // -----
33
34     std::string          asString () const;
35
36     void                print (std::ostream& os) const;
37
38 private:
39
40     // private fields
41     // -----
42 };
43 typedef SMARTP<waeHandler> S_waeHandler;
44 EXP std::ostream& operator << (std::ostream& os, const S_waeHandler& elt);

```

The creation of the instances in `src/wae/waeHandlers.cpp` is done with:

```

1 S_waeHandler waeHandler::create ()
2 {
3     waeHandler* o =
4         new waeHandler ();
5     assert (o != nullptr);
6     return o;
7 }

```

## 13.10 Files contents layout

Indentation is done by two spaces, avoiding TAB characters.

In `*.h` files, the classes declarations contain all of part of the following:

- public data types, usually enumeration type `s`, if any;
- public static class `create* ()` methods, except for pure virtual classes, in which case they are commented out;
- constructors and destructor;
- public `set* ()` and `get* ()` methods;
- public services if any;
- public visiting methods, i.e. `acceptIn ()`, `acceptOut ()` and `browseData ()`, if the class contains browsable data such as STL lists, vectors, maps and sets;
- public print methods, such as `asString ()` and `print ()`;
- private methods if any;
- private fields.
- private work methods if any;
- private work fields.

A work method is used internally by the class, while a work field is one that evolves as the class contents is populated.

Most class declarations are followed by a smart pointer type and a `operator <<`, such as:

```

1 typedef SMARTP<msrHarmonyDegree> S_msrHarmonyDegree;
2 EXP std::ostream& operator << (std::ostream& os, const S_msrHarmonyDegree& elt);

```

The same order for constructors, destructor and methods is followed in most .cpp files.

## 13.11 #define DEBUG\* code sections

Some sections of code in .cpp are controlled by such definitions:

- `//#define DEBUG_EARLY_OPTIONS::` in `src/oah/oahEarlyOptions.cpp`;
- `//#define DEBUG_INDENTER::` in `src/oah/mfIndentedTextOutput.cpp`;
- `//#define DEBUG_SPLITTING::` in `src/mfutilities/mfStringsHandling.cpp`

These can be uncommented to obtain development-time tracing information, without there being a need for such in MusicFormats library regular use.

## 13.12 Identifiers choice conventions

The following rules apply:

- all enumeration type names describing variants in classes end in 'Kind';
- all enumeration constants start with 'k'. common prefixes are used to help locate all occurrences of constants of the given type in a text editor, as:

```

1 enum class msrBassFigurePrefixKind {
2     kBassFigurePrefix_UNKNOWN,
3     kBassFigurePrefix_DoubleFlat, kBassFigurePrefix_Flat, kBassFigurePrefix_FlatFlat,
4     kBassFigurePrefix_Natural,
5     kBassFigurePrefix_SharpSharp, kBassFigurePrefix_Sharp, kBassFigurePrefix_DoubleSharp
6 };

```

- all classes names have a prefix indicating which part of MusicFormats there belong to, such as class `msrTimeSignature`, `oahAtomStoringAValue` and `msdlKeywordsLanguageAtom`;
- all classes member fields start with 'f';
- all global variables start with 'gGlobal';
- all variables private to methods start with 'pPrivate';
- some `K_*` constants are declared as static class constant members, such as in `src/formats/msr/msrParts.h`:

```

1 // -----
2 class EXP msrPart : public msrPartGroupElement
3 {
4     public:
5
6         // constants
7         // -----
8
9         static const int K_PART_HARMONIES_STAFF_NUMBER;
10        static const int K_PART_HARMONIES_VOICE_NUMBER;
11
12        static const int K_PART_FIGURED_BASS_STAFF_NUMBER;
13        static const int K_PART_FIGURED_BASS_VOICE_NUMBER;
14
15        // ... ..
16 };

```

with the definition in `src/formats/msr/msrParts.cpp`:

```

1 // -----
2 // constants
3 const int msrPart::K_PART_HARMONIES_STAFF_NUMBER = 10;
4 const int msrPart::K_PART_HARMONIES_VOICE_NUMBER = 11;
5
6 const int msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER = 20;
7 const int msrPart::K_PART_FIGURED_BASS_VOICE_NUMBER = 21;

```

When a field is an STL container, such a vector, list, map or set, this is indicated as part of the identifier, such as:

```

1 std::map<std::string, std::string>    fPartsRenamingMap;

```

or

```

1 fStringToDalSegnoKindMapVariable;

```

or

```

1 std::map<std::string, Sxmlelement>    fPartMeasureNumbersToElementsMap;

```

All `create* ()` methods create class instances, and are paired with an explicit constructor with the same parameters:

```

1 // creation from MusicXML
2 // -----
3
4 static SMARTP<msrHarmonyDegree> create (
5         int                inputLineNumber,
6         int                harmonyDegreeValue,
7         msrAlterationKind  harmonyDegreeAlterationKind,
8         msrHarmonyDegreeTypeKind harmonyDegreeTypeKind);
9
10 protected:
11
12 // constructors/destructor
13 // -----
14
15         msrHarmonyDegree (
16             int                inputLineNumber,
17             int                harmonyDegreeValue,
18             msrAlterationKind  harmonyDegreeAlterationKind,
19             msrHarmonyDegreeTypeKind harmonyDegreeTypeKind);
20
21 virtual ~msrHarmonyDegree ();

```



Some classes use private fields and methods for their internal working. A field in the form `fCurrent*` denotes something whose value is not permanent once set. Fields named `fPending*` contain values gathered to be used later, such as `fPendingHarmoniesList` in `src/passes/mxsr2msr/mxsr2msrTranslator.h/.cpp`.

## 13.13 Exceptions and warnings/errors reporting

MusicFormats defines exceptions for its needs in `src/wae/waeExceptions.h/.cpp`. These exceptions can be related to a format, a pass or a converter. Exceptions named `*Internal*` are raised when something that should not happen occurs: this to avoid ugly crashes later in the execution.

All the `std::exception` classes are derived from `mfException`, that contains:

```

1 class EXP mfException: public std::exception
2 {
3     public:
4
5         // constructors/destructor
6         // -----
7
8         mfException (
9             std::string const& exceptionDescription = "",
10             int exceptionNumber = 0,
11             int exceptionLevel = 0
12         ) throw ()
13         : fExceptionDescription (
14             "mfException: " + exceptionDescription),
15           fExceptionNumber (exceptionNumber),
16           fExceptionLevel (exceptionLevel)
17         {}
18
19         // ... ..
20
21     private:
22
23         std::string fExceptionDescription;
24
25         int fExceptionNumber;
26         int fExceptionLevel;
27 };

```

An example of `std::exception` is:

```

1 class EXP mxsr2msrException: public mfException
2 {
3     public:
4         mxsr2msrException (std::string const& exceptionDescription) throw ()
5         : mfException (exceptionDescription)
6         {}
7 };
8 typedef SMARTP<musicxmlException> S_musicxmlException;

```

There are warning and error reporting functions in `src/wae/waeInterface.h.h/.cpp`. Examples are:

```

1 void oahAtomExpectingAValue::applyElement (std::ostream& os)
2 {
3     std::stringstream s;
4
5     s <<
6     "Applying atom expecting a value '" <<
7     fetchNames () <<
8     "' without a value";
9
10    oahInternalError (s.str ());

```

```
11 }
```

and:

```
1  case msrPedalTypeKind::kPedalType_UNKNOWN:
2      {
3          // should not occur
4
5          std::stringstream s;
6
7          s <<
8              "msrPedal '" <<
9              elt->asShortString () <<
10             "' has no pedal type";
11
12         msrInternalError (
13             gGlobalServiceRunData->getInputSourceName (),
14             inputLineNumber,
15             __FILE__, __LINE__,
16             s.str ());
17     }
18     break;
```

Another one is:

```
1 void mxsr2msrTranslator::visitEnd ( S_accordion_registration& elt )
2 {
3     int inputLineNumber =
4         elt->getInputLineNumber ();
5
6     #ifdef TRACING_IS_ENABLED
7         if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
8             gLogStream <<
9                 "--> End visiting S_accordion_registration" <<
10                ", line " << inputLineNumber <<
11                std::endl;
12         }
13     #endif
14
15     // An accordion-registration element needs to have
16     // at least one of the child elements present
17
18     if (fCurrentAccordionNumbersCounter == 0) {
19         musicxmlWarning (
20             gGlobalServiceRunData->getInputSourceName (),
21             inputLineNumber,
22             "accordion-registration has 0 child element, ignoring it");
23     }
24
25     else {
26         // create the accordion registration
27         S_msrAccordionRegistration
28             accordionRegistration =
29             msrAccordionRegistration::create (
30                 inputLineNumber,
31                 fCurrentAccordionHigh,
32                 fCurrentAccordionMiddle,
33                 fCurrentAccordionLow);
34
35         // append it to the current part
36         fCurrentPart->
37             appendAccordionRegistrationToPart (
38                 accordionRegistration);
39     }
40 }
```

## 13.14 Exporting symbols for Windows DLLs

Windows needs export specifications for the symbols used by clients of a DLL.

<https://docs.microsoft.com/en-us/cpp/build/exporting-from-a-dll-using-declspec-dllexport?view=msvc-160> is titled "Exporting from a DLL Using `__declspec(dllexport)`". It states that:

- to export functions, the `__declspec(dllexport)` keyword must appear to the left of the calling-convention keyword, if a keyword is specified. For example:

```
1 __declspec(dllexport) void __cdecl Function1(void);
```

- to export all of the public data members and member functions in a class, the keyword must appear to the left of the class name as follows:

```
1 class __declspec(dllexport) CExampleExport : public CObject
2 { ... class definition ... };
```

MusicFormats uses symbol `EXP`, supplied by `libmusicxml/samples/`.

It is defined in `libmusicxml/src/elements!exports.h` to be empty except on Windows, where it is a default visibility attribute:

```
1 #ifndef __exports__
2 #define __exports__
3
4 #if defined(WIN32) // && !defined (GCC)
5
6 # ifdef MSVC
7 #  pragma warning (disable : 4267)
8 #  pragma warning (disable : 4275)
9 #  pragma warning (disable : 4251)
10 #  pragma warning (disable : 4786)
11 #  pragma warning (disable : 4251)
12 #  pragma warning (disable : 4275)
13 # endif
14
15 # ifdef LIBMUSICXML_EXPORTS
16 #  define EXP __declspec(dllexport)
17
18 # elif defined(LIBMUSICXML_STATIC)
19 #  define EXP
20
21 # else
22 #  define EXP __declspec(dllimport)
23 # endif
24
25 #else
26
27 # ifdef LIBMUSICXML_EXPORTS
28 #  define EXP __attribute__((visibility("default")))
29 # else
30 #  define EXP
31 # endif
32
33 #endif
34
35 #endif
```

## 13.15 Dynamic type checking

Enumeration types are not ideal to distinguish variants when inheritance is used, mainly because adding new derived types imposes the addition of new constants, thus impacting other areas in the code base.

`dynamic_cast` is used in those cases, such as:

```

1  // handle the option
2  if (
3      // options group?
4      S_oahGroup
5      group =
6          dynamic_cast<oahGroup*>(&(*element))
7  ) {
8      registerOahElementUse (
9          group, optionNameUsed, ""); // "===group==="; // JMI to debug
10 }
11
12 else if (
13     // options subgroup?
14     S_oahSubGroup
15     subGroup =
16         dynamic_cast<oahSubGroup*>(&(*element))
17 ) {
18     registerOahElementUse (
19         subGroup, optionNameUsed, ""); // "===subGroup==="; // JMI to debug
20 }

```

## 13.16 Input line numbers

The passes and converters in MusicFormats convert formats of scores from one format to another. In order to produce helpful warning and error messages, several descriptions contain a field:

```

1  int          fInputLineNumber;

```

An input line number in the `xmlelement` class is the only thing that has had to be added to `libmusicxml2` for the needs of MusicFormats.

Also, many methods contain an `int inputLineNumber` parameter, which is always the first one:

```

1  msrElement::msrElement (
2      int inputLineNumber)
3  {
4      fInputLineNumber = inputLineNumber;
5  }

```

Such input line numbers can be present in the output of the converters, such as:

```

1  jacquesmenu@macmini: ~/musicformats-git-dev/files > xml2ly -query input-line-numbers
2  --- Help for atom "input-line-numbers" in subgroup "Output"
3      -iln, -input-line-numbers
4          Generate after each note and barLine a comment containing
5          its MusicXML input line number.
6          This is useful when debugging xml2ly.

```

Generators such as `Mikrokosmos3Wandering` don't read any input, and the input line numbers they use are the ones in the source code, which is helpful for debugging. For example:

```

1 S_msrStaff Mikrokosmos3WanderingGenerator::createStaffInPart (
2     int          staffNumber,
3     const S_msrPart& part)
4 {
5     // create the staff
6     S_msrStaff
7         staff =
8         msrStaff::create (
9             __LINE__, msrStaffKind::kStaffKindRegular, staffNumber, part);
10
11     // append it to the part
12     part ->
13         addStaffToPartCloneByItsNumber ( // JMI NOT clone???
14             staff);
15
16     return staff;
17 }

```

In all output produced by MusicFormats tools, including trace informations and comments in the generated output referring to the input data, line numbers appear as:

```

1 line <number>

```

with a single space before the number, as in:

```

1 e16 %{ line 153 %} ] %{ line 163 kBeamEnd %}

```

This helps locating such occurrences in the debug process.

## 13.17 Static declarations

They are used for:

- classes methods such as method `msrTie::create ()`, method `msrTempo::createTempoPerMinute ()` and method `msrTemp::msrTempoBeatUnitsKindAsString ()`;
- classes constant fields such as constant `msrStanza::K_STANZA_NUMBER_UNKNOWN`, to be preferred to C-style `#define` preprocessor specifications for type safety;
- functions of methods remanent variables such as function private variable `pPrivateThisMethodHasBeenRun` in function `initializeMsrGenerationAPI ()`;
- library-wide variables such as global variable `gGlobalOutputStreamIndenter` and global variable `gGlobalTimingItem` that would be too cumbersome to pass to each and every method or function that uses them.

## 13.18 Avoiding MusicFormats multiple initializations

Such behaviour would create data structures several times, the result being unnecessary activities being performed. Avoiding it is done with function private variable `pPrivateThisMethodHasBeenRun`, here in `src/formats/bsr/bsr.cpp`:

```

1 void initializeBSR ()
2 {
3     // protect library against multiple initializations
4     static Bool pPrivateThisMethodHasBeenRun (false);
5
6     if (! pPrivateThisMethodHasBeenRun) {
7 #ifdef TRACING_IS_ENABLED
8         if (gGlobalOahEarlyOptions.getEarlyTracingOah () && ! gGlobalOahEarlyOptions.
9             getEarlyQuietOption ()) {
10             gLogStream <<
11                 "Initializing BSR basic types handling" <<
12                 std::endl;
13         }
14 #endif
15
16         // BSR braille output kinds handling
17         // -----
18         initializeBsrBrailleOutputKindsMap ();
19
20         // BSR texts languages handling
21         // -----
22         initializeBsrTextsLanguageKindsMap ();
23
24         pPrivateThisMethodHasBeenRun = true;
25     }
26 }
27

```

## 13.19 Enumeration types

All enumeration types use the C++17 'enum class' feature, such as:

```

1 enum class msrSlurTypeKind {
2     kSlurType_UNKNOWN,
3
4     kSlurTypeRegularStart, kSlurTypePhrasingStart,
5     kSlurTypeContinue,
6     kSlurTypeRegularStop, kSlurTypePhrasingStop
7 };

```

This prevents enumeration constants name conflicts across enumeration types, and qualified names such as constant `msrSlurTypeKind::kSlurTypeRegularStart` are quite explicit.

Many enumerations names end in 'Kind', which is a way distinguish them from rather similar classes names in some cases.

Enumeration constants in the form `k*_UNKNOWN` are used to indicate a value that *has not been set yet*. There are always the first one in the corresponding enumeration, to benefit from the the C++17 implicit initialization to the the equivalent of 0.

An enumeration constant may end in 'None', meaning that "[NONE]" is actually a possible value for the corresponding type:

```

1 // repeat winged
2 enum class msrBarLineRepeatWingedKind {
3     kBarLineRepeatWingedNone,
4
5     kBarLineRepeatWingedStraight, kBarLineRepeatWingedCurved,
6     kBarLineRepeatWingedDoubleStraight, kBarLineRepeatWingedDoubleCurved
7 };

```

Here how the "winged" MusicXML attribute of <repeat/> is analysed in :

```

1 void mxsr2msrTranslator::visitStart ( S_repeat& elt )
2 {
3     // ... ..
4
5     std::string winged = elt->getAttributeValue ("winged");
6
7     fCurrentBarLineRepeatWingedKind =
8         msrBarLineRepeatWingedKind::kBarLineRepeatWingedNone; // default value
9
10    if (winged.size ()) {
11        if (winged == "[NONE]") {
12            fCurrentBarLineRepeatWingedKind =
13                msrBarLineRepeatWingedKind::kBarLineRepeatWingedNone;
14        }
15        else if (winged == "straight") {
16            fCurrentBarLineRepeatWingedKind =
17                msrBarLineRepeatWingedKind::kBarLineRepeatWingedStraight;
18        }
19        else if (winged == "curved") {
20            fCurrentBarLineRepeatWingedKind =
21                msrBarLineRepeatWingedKind::kBarLineRepeatWingedCurved;
22        }
23        else if (winged == "double-straight") {
24            fCurrentBarLineRepeatWingedKind =
25                msrBarLineRepeatWingedKind::kBarLineRepeatWingedDoubleStraight;
26        }
27        else if (winged == "double-curved") {
28            fCurrentBarLineRepeatWingedKind =
29                msrBarLineRepeatWingedKind::kBarLineRepeatWingedDoubleCurved;
30        }
31        else {
32            std::stringstream s;
33
34            s <<
35                "repeat winged \"" << winged <<
36                "\" is unknown";
37
38            musicxmlError (
39                gGlobalServiceRunData->getInputSourceName (),
40                inputLineNumber,
41                __FILE__, __LINE__,
42                s.str ());
43        }
44    }
45
46    // ... ..

```

The MusicFormats enumeration type `s` come with a `<<` operator to display their values in a user friendly way, such as type `msrChordInKind` in <src/formats/msr/msrBasicTypes.h/.cpp>:

```

1 enum class msrChordInKind {
2     kChordIn_UNKNOWN,
3
4     kChordIn_Measure,
5     kChordIn_Tuplet,
6     kChordIn_GraceNotesGroup
7 };
8
9 EXP std::string msrChordInKindAsString (
10     msrChordInKind chordInKind);
11
12 std::ostream& operator << (std::ostream& os, const msrChordInKind& elt);

```

The implementation is:

```

1 std::string msrChordInKindAsString (
2     msrChordInKind chordInKind)
3 {
4     std::string result;
5
6     switch (chordInKind) {
7         case msrChordInKind::kChordIn_UNKNOWN:
8             result = "kChordIn_UNKNOWN";
9             break;
10        case msrChordInKind::kChordInMeasure:
11            result = "kChordInMeasure";
12            break;
13        case msrChordInKind::kChordInTuplet:
14            result = "kChordInTuplet";
15            break;
16        case msrChordInKind::kChordInGraceNotesGroup:
17            result = "kChordInGraceNotesGroup";
18            break;
19    } // switch
20
21    return result;
22 }
23
24 std::ostream& operator << (std::ostream& os, const msrChordInKind& elt)
25 {
26     os << msrChordInKindAsString (elt);
27     return os;
28 }

```

## 13.20 yes/no enumerations types

Boolean argument to methods calls are fine in simple cases such as:

```

1 void setCombinedBooleanVariables (Bool value);

```

But when there are multiple arguments, the semantics of the `true` or `false` constants is far from obvious.

This is why we use enum classes such as:

```

1 enum class msrVoiceCreateInitialLastSegmentKind {
2     kCreateInitialLastSegmentYes,
3     kCreateInitialLastSegmentNo
4 };

```

in such cases, so that the arguments bare a clear semantics:

```

1 fPartHarmoniesVoice =
2     msrVoice::create (
3         inputLineNumber,
4         msrVoiceKind::kVoiceKindHarmonies,
5         partHarmoniesVoiceNumber,
6         msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes,
7         fPartHarmoniesStaff);

```



## 13.21 Boolean values anyway

Defining a yes/no enumeration type for 'true' boolean values such as the variables containing the OAH options would be cumbersome. The C++ `bool` type suffers from the C heritage, in which integers and even pointers can be mixed in and considered as boolean values.

Moreover, a `bool` variable not explicitly initialized in the developer's code can lead to hard to fix bugs, in particular when the MusicFormats library is used on various hardware and operating systems.

For these reasons, MusicFormats features a class `Bool` defined in `src/utilities/mfBool.h/.cpp`. It encapsulates the actual `bool` value, enforcing that its initial value is not random, but known to the developer, through constructors. This also avoids in particular long sequences of initializations in the passes constructors.

## 13.22 On/off values

Some elements in MusicFormats are represented by a *three-state* value.

On/off values are handled by enumeration type `mfOnOffKind`, declared in `src/mflibrarymfOnOff.h`:

```
1 // -----
2 enum class mfOnOffKind {
3     kMfOnOffUnknown,
4     kMfOnOffOn, kMfOnOffOff
5 };
6
7 Bool mfOnOffKindAsBool (
8     mfOnOffKind onOffKind);
9
10 std::string mfOnOffKindAsString (
11     mfOnOffKind onOffKind);
12
13 std::ostream& operator << (std::ostream& os, const mfOnOffKind elt);
```

This is the case for example of page ragging information in `src/formats/lpsr//lpsr/lpsrPapers.h`:

```
1 // on/off values
2 mfOnOffKind          fRaggedBottom;
3 S_oahOnOffAtom       fRaggedBottomAtom;
4
5 mfOnOffKind          fRaggedLast;
6 S_oahOnOffAtom       fRaggedLastAtom;
7
8 mfOnOffKind          fRaggedLastBottom;
9 S_oahOnOffAtom       fRaggedLastBottomAtom;
10
11 mfOnOffKind          fRaggedRight;
12 S_oahOnOffAtom       fRaggedRightAtom;
```

Care must be taken in `src/mflibrarymfOnOff.cpp` when converting an enumeration type `mfOnOffKind` value to type `Bool`:

```
1 Bool mfOnOffKindAsBool (
2     mfOnOffKind onOffKind)
3 {
4     Bool result;
5
6     switch (onOffKind) {
7         case mfOnOffKind::kMfOnOffUnknown:
8             {
9                 mfError (
```

```

10     __FILE__, __LINE__,
11     "mfOnOffKind::kMfOnOffUnknown cannot be converted to Bool");
12 }
13 break;
14 case mfOnOffKind::kMfOnOffOn:
15     result = true;
16     break;
17 case mfOnOffKind::kMfOnOffOff:
18     result = false;
19     break;
20 } // switch
21
22 return result;
23 }

```

## 13.23 Iterating over numeration types

Such iterations rely on template classes. For this to work, the enumeration type should provide specific `AllFirst` and `AllLast` 'aliases' for the first and last constants in the type.

This is done for enumeration type `msrHarmonyKind` in `src/formats/msr/msrBasicTypes.h`:

```

1 // harmonies
2 // -----
3 enum class msrHarmonyKind {
4     kHarmony_UNKNOWN,
5
6     // MusicXML harmonies
7
8     kHarmonyMajor, kHarmonyMinor,
9     kHarmonyAugmented, kHarmonyDiminished,
10
11     // ... ..
12
13     // other
14
15     kHarmonyOther,
16
17     kHarmonyNone,
18
19     // aliases
20     // -----
21
22     AllFirst = kHarmony_UNKNOWN,
23     AllLast  = kHarmonyNone,
24
25     // ... ..
26 };

```

class `mfEnumAll` is defined in `src/mfutilities/mfEnumAll.h`:

```

1 // -----
2 /*
3  https://stackoverflow.com/questions/8498300/allow-for-range-based-for-with-enum-classes
4  */
5 template< typename T >
6 class mfEnumAll
7 {
8     public:
9
10     class Iterator
11     {
12     public:

```

```

13
14     Iterator (int value)
15         : fIterationIndex (value)
16         {}
17
18     T operator* (void) const
19         { return (T) fIterationIndex; }
20
21     void operator++ (void)
22         { ++fIterationIndex; }
23
24     Bool operator != (Iterator rhs)
25         { return fIterationIndex != rhs.fIterationIndex; }
26
27     private:
28
29         int fIterationIndex;
30     };
31 };
32
33 template< typename T >
34 typename mfEnumAll<T>::Iterator begin (mfEnumAll<T>)
35 {
36     return typename mfEnumAll<T>::Iterator ((int)T::AllFirst);
37 }
38
39 template< typename T >
40 typename mfEnumAll<T>::Iterator end (mfEnumAll<T>)
41 {
42     return typename mfEnumAll<T>::Iterator (((int)T::AllLast) + 1);
43 }

```

The mfEnumAll template class, defined in [src/mfutilities/mfEnumAll.h](#) can then be used to iterate from constant msrHarmonyKind::AllFirst to constant msrHarmonyKind::AllLast, here in [src/formats/msr/msrBasicT](#)

```

1 void initializeHarmonyStructuresMap ()
2 {
3     // protect library against multiple initializations
4     static Bool pPrivateThisMethodHasBeenRun (false);
5
6     if (! pPrivateThisMethodHasBeenRun) {
7         for (auto e : mfEnumAll<msrHarmonyKind> ()) {
8             // create the harmony structure
9             S_msrHarmonyStructure
10             harmonyStructure =
11                 msrHarmonyStructure::create (
12                     e);
13
14             // register it in the map
15             gGlobalHarmonyStructuresMap [e] =
16                 harmonyStructure;
17         } // for
18
19         pPrivateThisMethodHasBeenRun = true;
20     }
21 }

```

The mfEnumAll template class, defined in [src/mfutilities/mfEnumAll.h](#) can then be used to iterate from constant msdlTokenKind::AllFirst to constant msdlTokenKind::AllLast, here in [src/formats/msdl/msdlTokens](#)

```

1     for (auto e : EnumNonSeparators<msdlTokenKind> ()) {
2         std::string
3         nonSeparatorTokenAsMsdString =
4             msdlTokenKindAsMsdString (
5                 e,

```

```

6         keywordsLanguageKind);
7
8         // ... ..
9     } // for

```

All such class `Enum*` classes in `MusicFormats` are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'class Enum' *
2 formats/msdl/msdlTokens.h:class EnumNonSeparators
3 formats/msdl/msdlTokens.h:class EnumLanguageIndependent
4 formats/msdl/msdlTokens.h:class EnumLanguageDependent
5 formats/msr/msrBasicTypes.h:class EnumTrueHarmonies
6 utilities/mfutilities.h:class mfEnumAll

```

For example class `EnumTrueHarmonies`, that relies on constant `msrHarmonyKind::TrueHarmoniesFirst` and constant `msrHarmonyKind::TrueHarmoniesLast`:

```

1 void msrHarmonyStructure::printAllHarmoniesStructures (std::ostream& os)
2 {
3     os <<
4     "All the known harmonies structures are:" <<
5     std::endl << std::endl;
6
7     ++gIndenter;
8
9     for (auto e : EnumTrueHarmonies<msrHarmonyKind> ()) {
10        // create the harmony intervals
11        S_msrHarmonyStructure
12        harmonyStructure =
13        msrHarmonyStructure::create (
14        e);
15
16        // print it
17        os <<
18        harmonyStructure <<
19        std::endl;
20    } // for
21
22    --gIndenter;
23 }

```

## 13.24 Rational numbers

`MusicFormats` uses rationals for notes sounding and display whole notes and measure positionss, among others. class `Rational` is defined by `libmusicxml2` in `libmusicxml/src/lib!Rational.h/.cpp`:

```

1 class EXP Rational {
2
3     private:
4
5         long int fNumerator;
6         long int fDenominator;
7
8         // Used by rationalise()
9         long int gcd(long int a, long int b);
10
11     public:
12
13         Rational(long int num = 0, long int denom = 1);
14         Rational(const Rational& d);
15         Rational(const std::string &str);
16

```

```

17 // ... ..
18 };

```

Rationals are not used, however, for tuples factors, see .

## 13.25 Default values

The guide lines for MusicFormats in this matter are:

- smart pointers are initialized to `nullptr` in the class SMARTP constructor (they're smart after all), defined by libmusicxml2 in `libmusicxml/src/lib!smartpointer.h`:

```

1 template<class T> class SMARTP {
2     private:
3         ///! the actual pointer to the class
4         T* fSmartPtr;
5
6     public:
7         ///! an empty constructor - points to null
8         SMARTP() : fSmartPtr(0) {}
9
10    // .....

```

- all variables and classes fields of non-class types, such as `int`, `float` and enumeration type `s`, are to be initialized explicitly;
- MusicFormats functions and methods parameters never have default values: overloading is used instead.

## 13.26 create\* methods

All concrete classes, i.e. those that are not pure virtual, have `create* ()` methods paired with a constructor with the exact same parameters.

In most cases, there are just named `create* ()`, but a couple of them have more explicit names.

One case is that of class `msrTempo` in `src/formats/msr/msrTempos.h/.cpp`, because calls to them would be hard to distinguish at first glance otherwise:

```

1 class EXP msrTempo : public msrMeasureElement
2 {
3     // ... ..
4
5     static SMARTP<msrTempo> createTempoWordsOnly (
6         int inputLineNumber,
7         S_msrWords tempoWords);
8
9     static SMARTP<msrTempo> createTempoPerMinute (
10        int inputLineNumber,
11        msrDottedDuration tempoBeatUnit,
12        std::string tempoPerMinute,
13        msrTempoParenthesizedKind
14            tempoParenthesizedKind,
15        msrPlacementKind tempoPlacementKind);
16
17    static SMARTP<msrTempo> createTempoBeatUnitEquivalent (
18        int inputLineNumber,
19        msrDottedDuration tempoBeatUnit,

```

```

20     msrDottedDuration  tempoEquivalentBeatUnit ,
21     msrTempoParenthesizedKind
22         tempoParenthesizedKind ,
23     msrPlacementKind  tempoPlacementKind);
24
25     static SMARTP<msrTempo> createTempoNotesRelationship (
26         int                inputLineNumber ,
27         S_msrTempoNotesRelationshipElements
28             tempoNotesRelationshipLeftElements ,
29         msrTempoNotesRelationshipKind
30             tempoNotesRelationshipKind ,
31         S_msrTempoNotesRelationshipElements
32             tempoNotesRelationshipRightElements ,
33         msrTempoParenthesizedKind
34             tempoParenthesizedKind ,
35         msrPlacementKind  tempoPlacementKind);
36
37     // ... ..
38 };

```

Another case is that of class `msrKey` in `src/formats/msr/msrKeys.h/.cpp`, in which the variant chosen is made explicit:

```

1  class EXP msrKey : public msrMeasureElement
2  {
3      // ... ..
4
5      static SMARTP<msrKey> createTraditional (
6          int                inputLineNumber ,
7          msrQuarterTonesPitchKind keyTonicPitchKind ,
8          msrModeKind         modeKind ,
9          int                keyCancel);
10
11     static SMARTP<msrKey> createHumdrumScot (
12         int                inputLineNumber);
13
14     // ... ..
15 };

```

## 13.27 `get*()`, `set*()` and `fetch*()` methods

As is usual, classes private member variables are accessed through `set*()` () and `get*()` () methods. The name of these methods is obtained by replacing the 'f' in the field name by 'set' and 'get', respectively. In `src/formats/msr/msrTies.h`, one finds:

```

1      // set and get
2      // -----
3
4      void                setTieKind (msrTieKind tieKind)
5          { fTieKind = tieKind; }
6
7      msrTieKind          getTieKind () const
8          { return fTieKind; }
9
10     void                setTiePlacementKind (msrPlacementKind placementKind)
11         { fTiePlacementKind = placementKind; }
12
13     msrPlacementKind    getTiePlacementKind () const
14         { return fTiePlacementKind; }

```

`fetch` is used when the result is not store in a variable, but has to computed in some way. `src/formats/msr/msrSegme` contains:

```

1 S_msrStaff msrSegment::fetchSegmentUpLinkToStaff () const
2 {
3     S_msrStaff result;
4
5     if (fSegmentUpLinkToVoice) {
6         result =
7             fSegmentUpLinkToVoice->
8                 getVoiceUpLinkToStaff ();
9     }
10
11     return result;
12 }

```

Some methods exist in two versions, the second one with a `NonConst` suffix in the name. This can be the case if a private class field is to be modified outside the class, or if there are specific needs.

This happens for example in `src/formats/msr/mfslNotes.h/.cpp`:

```

1 // articulations
2 const std::list<S_msrArticulation>&
3     getNoteArticulations () const
4     { return fNoteArticulations; }
5
6 std::list<S_msrArticulation>&
7     getNoteArticulationsNonConst ()
8     { return fNoteArticulations; }

```

Another case is in `src/interpreters/mfsl/mfslDriver.h/.cpp`:

```

1 const yy::location& getScannerLocation () const
2     { return fScannerLocation; }
3
4 yy::location& getScannerLocationNonConst ()
5     // no const here
6     // due to constraints in the Flex-generated code
7     { return fScannerLocation; }

```

## 13.28 initialize\*() and finalize\*() methods

When a description contains many fields, the ones initialized by the values of the constructor's parameters are initialized in the latter, and the others are in an `initialize*() ()` method, such as:

```

1 msrPart::msrPart (
2     int inputLineNumber,
3     std::string partID,
4     S_msrPartGroup partUpLinkToPartGroup)
5     : msrPartGroupElement (inputLineNumber)
6 {
7     // replace spaces in partID to set fPartID
8     for_each (
9         partID.begin (),
10        partID.end (),
11        mfStringSpaceReplacer (fPartID, '_'));
12
13 /* JMI
14 // sanity check
15 mfAssert (
16     __FILE__, __LINE__,
17     partUpLinkToPartGroup != nullptr,
18     "partUpLinkToPartGroup is null");
19 */
20 }

```

```

21 // set part number
22 fPartAbsoluteNumber = ++gPartsCounter;
23
24 // set part's part group upLink
25 fPartUpLinkToPartGroup = partUpLinkToPartGroup;
26
27 // do other initializations
28 initializePart ();
29 }

```

Some finalize\*() () methods exist.

## 13.29 \*asString() and \*fromString() functions

Each enumeration type comes with an \*AsString() () function, to display the constant values as strings. Some also have a fromString () function to convert strings to the corresponding constant. For example, one finds in `src/formats/msr/msrBasicTypes.h/.cpp`:

```

1 // placement
2 // -----
3 enum class msrPlacementKind {
4     kPlacement_UNKNOWN,
5
6     kPlacementAbove, kPlacementBelow
7 };

```

```

1 // placement
2 // -----
3 msrPlacementKind msrPlacementKindFromString (
4     int     inputLineNumber,
5     std::string placementString)
6 {
7     msrPlacementKind result = msrPlacementKind::kPlacement_UNKNOWN; // default value
8
9     if (placementString == "above")
10         result = msrPlacementKind::kPlacementAbove;
11     else if (placementString == "below")
12         result = msrPlacementKind::kPlacementBelow;
13     else {
14         if (placementString.size ()) {
15             std::stringstream s;
16
17             s <<
18                 "placement \"" << placementString <<
19                 "\" should be 'above' or 'below'";
20
21             musicxmlError (
22                 gGlobalServiceRunData->getInputSourceName (),
23                 inputLineNumber,
24                 __FILE__, __LINE__,
25                 s.str ());
26         }
27     }
28
29     return result;
30 }

```

```

1 std::string msrPlacementKindAsString (
2     msrPlacementKind placementKind)
3 {
4     std::string result;
5

```



```

6   switch (placementKind) {
7       case msrPlacementKind::kPlacement_UNKNOWN:
8           result = "noPlacement";
9           break;
10      case msrPlacementKind::kPlacementAbove:
11          result = "placementAbove";
12          break;
13      case msrPlacementKind::kPlacementBelow:
14          result = "placementBelow";
15          break;
16  } // switch
17
18  return result;
19 }

```

Many classes have `asStringShort()` () methods to provide more compact a description as the one provided by the corresponding `asString()` () method.

### 13.30 `translate*()` methods and `convert*()` functions

To translate and to convert are alias in the context of MusicFormats.

For semantic clearness, `translate*()` () methods are supplied by the individual translators, as in `src/passes/msr2m`

```

1  // -----
2  class EXP msr2msrTranslator :
3
4      // MSR score
5
6      public visitor<S_msrScore>,
7
8      // ... ..
9
10 {
11     public:
12
13         msr2msrTranslator ();
14
15         virtual ~msr2msrTranslator ();
16
17         S_msrScore translateMsrToMsr (
18             const S_msrScore& theMsrScore);
19
20     // ... ..
21 };

```

```

1  S_msrScore msr2msrTranslator::translateMsrToMsr (
2      const S_msrScore& theMsrScore)
3  {
4      // sanity check
5      mfAssert (
6          __FILE__, __LINE__,
7          theMsrScore != nullptr,
8          "theMsrScore is null");
9
10     // the MSR score we're visiting
11     fVisitedMsrScore = theMsrScore;
12
13     // create the resulting MSR score
14     fResultingNewMsrScore =
15         msrScore::create (
16             K_MF_INPUT_LINE_UNKNOWN,
17             "msrScore::create()");

```

```

18
19 // create a msrScore browser
20 msrBrowser<msrScore> browser (this);
21
22 // browse the visited score with the browser
23 browser.browse (*fVisitedMsrScore);
24
25 // forget about the visited MSR score
26 fVisitedMsrScore = nullptr;
27
28 return fResultingNewMsrScore;
29 }

```

The `convert*() ()` functions are the interfaces to the translators, for example in `src/passes/msr2msr/msr2msrInte`

```

1 S_msrScore translateMsrToMsr (
2     S_msrScore      originalMsrScore,
3     const S_msrOahGroup&    msrOpts,
4     S_msr2msrOahGroup msr2msrOpts,
5     const std::string&      passNumber,
6     const std::string&      passDescription)
7 {
8     // ... ..
9
10    // the msr2msrTranslator
11    msr2msrTranslator
12        translator;
13
14    // build the resulting MSR score
15    S_msrScore
16        resultingNewMsrScore =
17        translator.translateMsrToMsr (
18            originalMsrScore);
19
20    // ... ..
21 }

```

## 13.31 context arguments

Some methods have such an argument, a `std::string`, to provide helpful information to the maintainer of MusicFormats. An example is method `msrMeasureRepeat::displayMeasureRepeat ()`, defined in `src/formats/msr/ms`

```

1 void msrMeasureRepeat::displayMeasureRepeat (
2     int      inputLineNumber,
3     const std::string& context)
4 {
5     gLogStream <<
6     std::endl <<
7     "*****>> MeasureRepeat " <<
8     ", measureRepeatMeasuresNumber: '" <<
9     fMeasureRepeatMeasuresNumber <<
10    ", measureRepeatSlashesNumber: '" <<
11    fMeasureRepeatSlashesNumber <<
12    "', voice:" <<
13    std::endl <<
14    fUpLinkToMeasureRepeatToVoice->getVoiceName () <<
15    " (" << context << ")" <<
16    ", line " << inputLineNumber <<
17    " contains:" <<
18    std::endl;
19
20    ++gIndenter;
21    print (gLogStream);

```

```

22  --gIndenter;
23
24  gLogStream <<
25      " <<*****" <<
26      std::endl << std::endl;

```

An call example in `src/formats/msr/msrVoices.h` is:

```

1  displayVoiceMeasureRepeatAndVoice (
2      inputLineNumber,
3      "createMeasureRepeatFromItsFirstMeasures() 1");

```

## 13.32 Sorting and compare\*() methods

MusicFormats sometimes needs to sort some data structures:

```

1  jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r '\.sort (' *
2  oah/oahBasicTypes.cpp:  optionsMapElementsNamesList.sort ();
3  passes/lpsr2lilypond/lpsr2lilypondTranslator.cpp:  frameFrameNotesList.sort (
4  formats/msr/msrMeasuresSlices.cpp:  fSliceNotesFlatList.sort (
5  formats/msr/msrMeasuresSlices.cpp:  fSliceNoteEventsList.sort (
6  formats/msr/msrStaves.cpp:  fStaffAllVoicesList.sort (
7  formats/msr/msrStaves.cpp:  fStaffAllVoicesList.sort (
8  formats/msr/msrStaves.cpp:  fStaffRegularVoicesList.sort (
9  formats/msr/msrMeasures.cpp:  fMeasureElementsList.sort (
10 formats/msr/msrMeasures.cpp:  fMeasureElementsList.sort (
11 formats/msr/msrParts.cpp:  fPartAllStavesList.sort (
12 formats/msr/msrParts.cpp:  fPartAllStavesList.sort (
13 formats/lpsr/lpsrParts.cpp:  fPartBlockElementsList.sort (

```

There are thus a number of compare\* () methods according to the needs:

```

1  jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r compare * | grep '\.h'
2  oah/oahBasicTypes.h:  const std::multiset<S_oahElement, compareOahElements>&
3  oah/oahBasicTypes.h:  std::multiset<S_oahElement, compareOahElements>
4  oah/oahElements.h:struct compareOahElements {
5  formats/msr/msrParts.h:  static bool
6      compareStavesToHaveFiguredBassesBelowCorrespondingPart (
7  formats/msr/msrNotes.h:  static bool  compareNotesByIncreasingMeasurePosition (
8  formats/msr/msrMeasureElementsWithoutUpLinkToMeasure.h:  static bool
9      compareMeasureElementsByIncreasingMeasurePosition (
10 formats/msr/msrStaves.h:  static bool  compareVoicesByIncreasingNumber (
11 formats/msr/msrStaves.h:  static bool
12      compareVoicesToHaveHarmoniesAboveCorrespondingVoice (
13 formats/msr/msrStaves.h:  static bool
14      compareVoicesToHaveFiguredBassesBelowCorrespondingVoice (
15 formats/msr/msrMeasuresSlices.h:  static bool
16      compareNotesEventsByIncreasingMeasurePosition (
17 formats/msr/msrMeasuresSlices.h:  static bool
18      compareSimultaneousNotesChunksByIncreasingMeasurePosition (
19 formats/lpsr/lpsrParts.h:  static bool
20      compareElementsToHaveHarmoniesAboveCorrespondingStaff (
21 formats/lpsr/lpsrParts.h:  static bool  compareStaffBlockWithOtherElement (
22 formats/lpsr/lpsrParts.h:  static bool
23      compareChordNamesContextWithOtherElement (
24 utilities/mfutilities.h:  // compare indentation value

```

An example is:

```

1 bool msrPart::compareStavesToHaveFiguredBassesBelowCorrespondingPart (
2     const S_msrStaff& first,
3     const S_msrStaff& second)
4 {
5     int
6     firstStaffNumber =
7         first->getStaffNumber (),
8     secondStaffNumber =
9         second->getStaffNumber ();
10
11     if (firstStaffNumber > msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER) {
12         firstStaffNumber -= msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER + 1;
13     }
14     if (secondStaffNumber > msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER) {
15         secondStaffNumber -= msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER + 1;
16     }
17
18     bool result =
19         firstStaffNumber > secondStaffNumber;
20
21     return result;
22 }

```

## 13.33 Mutually dependent classes

C++17 has some constraints on how such classes can be defined, see for example <https://cplusplus.com/forum/articles/10627/>. MusicFormats sticks to having one header file per important class, with one exception.

In some cases, two classes have to know about each other, such as `msrPartGroup` and `msrPartGroupElement`. This is because part groups can be nested: a part group contains part group element, which can be staves as well as other part groups. This loop can be seen in red in figure 19.3 [The MSR classes hierarchy], page 182.

### 13.33.1 Pre-declarations

Handling such a mutual dependency in C++ is rather easy if both classes are declared in one and the same `.h` file, with a pre-declaration of one of them before the other one is declared, such as:

```

1 class msrPartGroup;
2 typedef SMARTP<msrPartGroup> S_msrPartGroup;

```

All one can do with a forward declared type is to declare a *pointer* or *reference* to said type. In particular, this precludes a forward declared type to be that of a class member or the return type of a class method.

As a matter of taste, MusicFormats follows libmusicxml2 by declaring the `create ()` static class methods this way:

```

1 // -----
2 class EXP msrBarLine : public msrMeasureElement
3 {
4     public:
5
6     // creation from MusicXML
7     // -----
8
9     static SMARTP<msrBarLine> create (

```

As an alternative, the same could be written as follows, at the cost of an extra pre-declaration for each class:

```

1 class EXP msrBarLine;
2 typedef SMARTP<msrBarLine> S_msrBarLine;
3
4 class EXP msrBarLine : public msrMeasureElement
5 {
6     public:
7
8     // creation from MusicXML
9     // -----
10
11     static S_msrBarLine create (...)
12
13     // ... ..
14 };

```

There are many classes pre-declarations in MusicFormats. We thus use a couple of *\*EnumTypes\** files to contain most of them, in order to avoid numerous header files includes:

```

1 jacquesmenu@macstudio-1:~/JMI_Developpement/musicformats-git-dev/src > look EnumTypes
2 ./representations/msdl/msdlEnumTypes.h
3 ./representations/msdl/msdlEnumTypes.cpp
4 ./representations/msr/msrTupletsEnumTypes.h
5 ./representations/msr/msrTemposEnumTypes.h
6 ./representations/msr/msrTupletsEnumTypes.cpp
7 ./representations/msr/msrMeasuresEnumTypes.cpp
8 ./representations/msr/msrTemposEnumTypes.cpp
9 ./representations/msr/msrStavesEnumTypes.h
10 ./representations/msr/msrNotesEnumTypes.h
11 ./representations/msr/msrNotesEnumTypes.cpp
12 ./representations/msr/msrStavesEnumTypes.cpp
13 ./representations/msr/msrMeasuresEnumTypes.h
14 ./representations/msr/msrRepeatsEnumTypes.cpp
15 ./representations/msr/msrRepeatsEnumTypes.h
16 ./representations/lpsr/lpsrEnumTypes.cpp
17 ./representations/lpsr/lpsrEnumTypes.h
18 ./representations/bsr/bsrEnumTypes.cpp
19 ./representations/bsr/bsrEnumTypes.h

```

### 13.33.2 Simple mutual dependency using separate header files

A `msrPartGroupElement` is either a `msrPart` or another `msrPartGroupElement`, since the latter can be nested.

A `msrPartGroupElement` cannot contain an uplink to an instance of `S_msrPartGroup`: pre-declaring type `S_msrPartGroup`:

```

1 class msrPartGroup;
2 typedef SMARTP<msrPartGroup> S_msrPartGroup;

```

and using it as the type of a class member common to all `msrPartGroupElement` sub-classes leads to the following error:

```

1 error: member access into incomplete type 'MusicFormats::msrPartGroup'

```

So `src/formats/msr/msrPartGroupsElements.h` contains only:

```

1  /*
2   Parts and part groups can be found in part groups,
3   hence class   msrPartGroupElement
4  */
5
6  class EXP msrPartGroupElement : public msrElement
7  /*
8   a purely virtual common ancestor to the msrPartGroup and msrPart classes,
9   which can be inside an msrPartGroup
10 */
11 {
12     // ... ..
13
14     private:
15
16         // private fields
17         // -----
18
19         /*
20          The part group uplink is declared in the sub-classes,
21          i.e. msrPart and msrPartGroup,
22          to allow for separate *.h files, C++ constraint
23         */
24 };

```

Then a `msrPartGroup`, a sub-class of `msrPartGroupElement` declared in `src/formats/msr/msrPartGroups.h`, can contain such an uplink to a `msrPartGroup` instance:

```

1  #include "msrPartGroupElements.h"
2
3  // ... ..
4
5  class   msrPartGroup;
6  typedef SMARTP<msrPartGroup> S_msrPartGroup;
7
8  // ... ..
9
10 class EXP msrPartGroup : public msrPartGroupElement
11 {
12     // ... ..
13
14     private:
15
16         // private fields
17         // -----
18
19         // upLinks
20
21         S_msrPartGroup      fPartGroupUpLinkToPartGroup;
22                             // part groups can be nested
23
24         // ... ..
25
26         // allowing for both parts and (sub-)part groups as elements
27
28         std::list<S_msrPartGroupElement>
29             fPartGroupElementsList;
30 };

```

Type `S_msrPartGroup` is used in the declaration of field `fPartGroupElementsList`, hence its pre-declaration.

Class `msrPart`, another sub-class of `msrPartGroupElement`, is defined in `src/formats/msr/msrParts.h` this way:

```

1 #include "msrPartGroupElements.h"
2
3 // ... ..
4
5 class EXP msrPart : public msrPartGroupElement
6 {
7     // ... ..
8
9     public:
10
11     // set and get
12     // -----
13
14     // upLinks
15
16     void                setPartUpLinkToPartGroup (
17                         const S_msrPartGroup& partGroup)
18                         { fPartUpLinkToPartGroup = partGroup; }
19
20     S_msrPartGroup      getPartUpLinkToPartGroup () const
21                         { return fPartUpLinkToPartGroup; }
22     // ... ..
23
24     private:
25
26     // private fields
27     // -----
28
29     // upLinks
30
31     S_msrPartGroup      fPartUpLinkToPartGroup;
32
33     // ... ..
34 };

```

This rather complex situation is depicted at the top of figure 19.3 [The MSR classes hierarchy], page 182.

### 13.33.3 More complex mutual dependencies

The case of notes, chords, triplets and grace notes groups is more intricate:

- a note can be standalone in a measure;
- a note can be part of:
  - a chord;
  - a triplet;
  - a grace notes group;
  - a double tremolo;
- a chord can be standalone in a measure;
- a chord can be part of:
  - a triplet;
  - a grace notes group;
- a triplet can be standalone in a measure;
- a triplet can be part of:

- another tuplet;
- a grace notes group is attached to:
  - a note;
- a double tremolo is standalone in a measure.

Class `msrDoubleTremolo` is a sub-class of `msrMeasureElement`.

Regarding classes `msrNote`, `msrChord` and `msrTuplet`:

- they have to be sub-classes of class `msrMeasureElement` in some way, since they can be standalone in a `msrMeasure` instance;
- they should be sub-classes of `msrTupletElement`, since they can be members of a `msrTuplet` instance.

This leads to the following hierarchy:

- `msrNote`, `msrChord` and `msrTuplet` are direct sub-classes of class `msrTupletElement`
- class `msrTupletElement` is a direct sub-class of `msrMeasureElement`.

This rather complex situation is depicted at the bottom of figure 19.3 [The MSR classes hierarchy], page 182.

But then, the mutual dependency of need a more complex representation.

## 13.34 Templates and functional programming usage

There are currently few templates in the MusicFormats code base, namely:

- some are used by the two-phase visitors pattern, see chapter 14 [The two-phase visitors pattern], page 87;
- some exist for enumeration types, such as:

```

1 template< typename T >
2 class EnumNonSeparators
3 {
4     public:
5
6     class Iterator
7     {
8     public:
9
10         Iterator (int value)
11             : fIterationIndex (value)
12             {}
13
14         T operator* (void) const
15             { return (T) fIterationIndex; }
16
17         void operator++ (void)
18             { ++fIterationIndex; }
19
20         Bool operator != (Iterator rhs)
21             { return fIterationIndex != rhs.fIterationIndex; }
22
23     private:
24 
```



```

25     int fIterationIndex;
26 };
27 };
28
29 template< typename T >
30 typename EnumNonSeparators<T>::Iterator begin (EnumNonSeparators<T>)
31 {
32     return typename EnumNonSeparators<T>::Iterator (((int)T)::NonSeparatorsFirst);
33 }
34
35 template< typename T >
36 typename EnumNonSeparators<T>::Iterator end (EnumNonSeparators<T>)
37 {
38     return typename EnumNonSeparators<T>::Iterator (((int)T)::NonSeparatorsLast) + 1);
39 }

```

- some are used by the code created by `bison`, like:

```

1  /// Construct and fill.
2  template <typename T>
3  value_type (YY_RVREF (T) t)
4      : yytypeid_ (&typeid (T))
5  {
6      ISCM_ASSERT (sizeof (T) <= size);
7      new (yyas_<T> ()) T (YY_MOVE (t));
8  }

```

There could be more templates use once MusicFormats reaches a rather stable code base and it is clear what parts of it can be restructured with generic code.

In the same vein, there is little use as of this writing of higher-level facilities such as `lambda` and functors.

---

## Part III

# The two-phase visitors pattern

## Chapter 14

# The two-phase visitors pattern

MusicFormats uses a two-phase visitors pattern designed by Dominique Fober to traverse data structures such an `xmlElement tree` or an MSR description, handling each node in the structure in a systematic way. This is in contrast to a programmed top-down traversal.

Such data structures traversals is actually *data driven*: a visitor can decide to 'see' only selected node types.

There are case where visiting is not the way to go, see the sections below.

### 14.1 Basic mechanism

Visiting a node in a data structure is done in this order:

- first phase: visit the node for the fist time, top-down;
- visit the node contents, using the same two-phase visitors pattern;
- second phase: visit the node for the second time, bottom-up.

The first can be used to prepare data needed for the node contents visit, for example. Then the second phase can used such data, if relevant, as well as data created by the node contents visit, do consolidate the whole.

A visitor class should:

- inherit from `basevisitor`;
- inherit from the smart pointer classes it visits;
- define methods `visitStart ()` and/or `visitEnd ()` depending on which phases it wants to handle. The parameter of all such `visit* ()` methods is always a reference to a smart pointer.

`basevisitor` is defined in `libmusicxml/src/visitors!basevisitor.h`, and contains nothing:

```
1 class basevisitor
2 {
3     public:
4         virtual ~basevisitor() {}
5 };
```

It is used as the base class of all visitors in `browsedata ()` methods:

```

1 void msrWords::acceptIn (basevisitor* v)
2 {
3     if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
4         gLogStream <<
5             "% ==> msrWords::acceptIn ()" <<
6             std::endl;
7     }
8
9     if (visitor<S_msrWords>*
10         p =
11         dynamic_cast<visitor<S_msrWords>*> (v)) {
12         S_msrWords elem = this;
13
14         if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
15             gLogStream <<
16                 "% ==> Launching msrWords::visitStart ()" <<
17                 std::endl;
18         }
19         p->visitStart (elem);
20     }
21 }

```

## 14.2 Browser template classes

There are several such classes, all with the same specification as the one in [libmusicxml/src/lib!tree\\_browser.h](#), named to allow easy search for them in the code base. For example, in [src/formats/msr/msrElements.h](#), there is:

```

1 // -----
2 template <typename T> class msrBrowser : public browser <T>
3 {
4     public:
5
6         msrBrowser (basevisitor* v) : fVisitor (v) {}
7
8         virtual ~msrBrowser () {}
9
10    public:
11
12        virtual void set (basevisitor* v) { fVisitor = v; }
13
14        virtual void browse (T& t) {
15            enter (t);
16
17            t.browseData (fVisitor);
18
19            leave (t);
20        }
21
22    protected:
23
24        basevisitor* fVisitor;
25
26        virtual void enter (T& t) { t.acceptIn (fVisitor); }
27        virtual void leave (T& t) { t.acceptOut (fVisitor); }
28 };

```

## 14.3 A first example: counting notes in MusicXML data

In `libmusicxml/samples/countnotes.cpp`, counting the notes in MusicXML data needs only see `S_note` nodes. class `countnotes` thus inherits only from a visitor for this type of node, and all the other node types are simply ignored.

vVisitor method `countnotes::visitStart` only has to increment the notes count:

Listing 14.1: `countnotes.cpp`

```

1 class countnotes :
2   public visitor<S_note>
3 {
4   public:
5     int fCount;
6
7     countnotes() : fCount (0) {}
8
9     virtual ~countnotes () {}
10
11    void visitStart ( S_note& elt )    { fCount++; }
12 };

```

## 14.4 A more complex example

Let's look at the `<scaling/>` MusicXML element:

```

1 <scaling>
2   <millimeters>7</millimeters>
3   <tenths>40</tenths>
4 </scaling>

```

It contains a `<millimeter/>` and a `<tenth/>` element. The latter two don't contain any other elements, so `visitStart ()` is enough for them.

There is nothing to do on the visit start upon `<scaling/>`, so there is no such method. On the visit end upon `<scaling/>`, though, the values grabbed from the `<millimeter/>` and `<tenth/>` elements are used to create the class `msrScaling` description.

Should a visit start method have been written, the execution order would have been:

```

1 mxsr2msrTranslator::visitStart ( S_scaling& elt)
2   mxsr2msrTranslator::visitStart ( S_millimeters& elt )
3   mxsr2msrTranslator::visitStart ( S_tenths& elt )
4   mxsr2msrTranslator::visitEnd ( S_scaling& elt)

```

or, depending on the order in which the subelements of `<scaling/>` are visited:

```

1 mxsr2msrTranslator::visitStart ( S_scaling& elt)
2   mxsr2msrTranslator::visitStart ( S_tenths& elt )
3   mxsr2msrTranslator::visitStart ( S_millimeters& elt )
4   mxsr2msrTranslator::visitEnd ( S_scaling& elt)

```

In `src/passes/mxsr2msr/mxsr2msrTranslator.cpp`, visiting a `<scaling/>` element is handled this way:

Listing 14.2: Visiting &lt;scaling /&gt;

```

1 void mxsr2msrTranslator::visitStart ( S_millimeters& elt )
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
5             gLogStream <<
6                 "--> Start visiting S_millimeters" <<
7                 ", line " << elt->getInputLineNumber () <<
8                 std::endl;
9         }
10    #endif
11
12    fCurrentMillimeters = (float)(*elt);
13 }
14
15 void mxsr2msrTranslator::visitStart ( S_tenths& elt )
16 {
17     #ifdef TRACING_IS_ENABLED
18         if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
19             gLogStream <<
20                 "--> Start visiting S_tenths" <<
21                 ", line " << elt->getInputLineNumber () <<
22                 std::endl;
23         }
24     #endif
25
26    fCurrentTenths = (float)(*elt);
27 }
28
29 void mxsr2msrTranslator::visitEnd ( S_scaling& elt)
30 {
31     int inputLineNumber =
32         elt->getInputLineNumber ();
33
34     #ifdef TRACING_IS_ENABLED
35         if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
36             gLogStream <<
37                 "--> End visiting S_scaling" <<
38                 ", line " << inputLineNumber <<
39                 std::endl;
40         }
41     #endif
42
43     // create a scaling
44     S_msrScaling
45         scaling =
46             msrScaling::create (
47                 inputLineNumber,
48                 fCurrentMillimeters,
49                 fCurrentTenths);
50
51     #ifdef TRACING_IS_ENABLED
52         if (gGlobalTracingOahGroup->getTraceGeometry ()) {
53             gLogStream <<
54                 "There are " << fCurrentTenths <<
55                 " tenths for " << fCurrentMillimeters <<
56                 std::endl;
57         }
58     #endif
59
60     // set the MSR score's scaling
61     fMsrScore->
62         setScaling (scaling);
63 }

```

## 14.5 Data browsing order

The order of the visit of a node's subnodes is programmed in `browseData ()` methods, such as:

Listing 14.3: `msrDoubleTremolo::browseData (basevisitor* v)`

```

1 void msrDoubleTremolo::browseData (basevisitor* v)
2 {
3     if (fDoubleTremoloFirstElement) {
4         // browse the first element
5         msrBrowser<msrElement> browser (v);
6         browser.browse (*fDoubleTremoloFirstElement);
7     }
8
9     if (fDoubleTremoloSecondElement) {
10        // browse the second element
11        msrBrowser<msrElement> browser (v);
12        browser.browse (*fDoubleTremoloSecondElement);
13    }
14 }
```

Since this order is set in the `browsearch ()` methods, it cannot be influenced by the visitors of the corresponding class instances.

There are cases where the data should be sorted prior to being browsed, such as the staves in parts: this ensures that they are browsed in this order: harmonies staff, other staves, figured bass staff.

## 14.6 Selectively inhibiting data browsing

### 14.6.1 Inhibiting data browsing in the code

In some cases, it is desirable not to browse part of the data. This is the case when a given class contains non-normalized data, i.e. data that occurs elsewhere and will be browsed in another class instance.

For example, class `msrMultipleFullBarRests` contains class `msrMeasure` instances. class `msrScore` contains:

```

1 // in <multiple-rest/>, the multiple full-bar rests are explicit,
2 // whereas LilyPond only needs the number of multiple full-bar rests
3 Bool                                fInhibitMultipleFullBarRestsBrowsing;
4
5 void                                setInhibitMultipleFullBarRestsBrowsing ()
6 {
7     fInhibitMultipleFullBarRestsBrowsing = true;
8 }
9
10 Bool                                getInhibitMultipleFullBarRestsBrowsing () const
11 {
12     return
13         fInhibitMultipleFullBarRestsBrowsing;
14 }
```

Class `lpsr2lilypondTranslator` checks this setting:

```

1 void lpsr2lilypondTranslator::visitEnd (S_msrNote& elt)
2 {
3     // ... ..
4
5     if (fOnGoingMultipleFullBarRests) {
6         switch (elt->getNoteKind ()) {
7             case msrNoteKind::kNoteRestInMeasure:
8                 // don't handle multiple full-bar rests, that's done in visitEnd (
9                 S_msrMultipleFullBarRests&)
10                if (elt->getNoteOccupiesAFullMeasure ()) {
11                    Bool inhibitMultipleFullBarRestsBrowsing =
12                        fVisitedLpsrScore->
13                        getMsrScore ()->
14                        getInhibitMultipleFullBarRestsBrowsing ();
15
16                    if (inhibitMultipleFullBarRestsBrowsing) {
17                        #ifdef TRACING_IS_ENABLED
18                            if (
19                                gGlobalTracingOahGroup->getTraceNotes ()
20                                ||
21                                gGlobalTracingOahGroup->getTraceMultipleFullBarRests ()
22                            ) {
23                                gLogStream <<
24                                    "% ==> end visiting multiple full-bar rests is ignored" <<
25                                    std::endl;
26                            }
27                        #endif
28
29                        #ifdef TRACING_IS_ENABLED
30                            if (gGlobalTracingOahGroup->getTraceNotesDetails ()) {
31                                gLogStream <<
32                                    "% ==> returning from visitEnd (S_msrNote&)" <<
33                                    std::endl;
34                            }
35                        #endif
36
37                        noteIsToBeIgnored = true;
38                    }
39                }
40                break;
41            // ... ..
42        }
43    }

```

Another example is in the class `lpsr2lilypondTranslator` constructor:

```

1 lpsr2lilypondTranslator::lpsr2lilypondTranslator (
2     S_lpsrScore      lpsrScore,
3     const S_msrOahGroup& msrOpts,
4     const S_lpsrOahGroup& lpsrOpts,
5     std::ostream&      lilypondCodeStream)
6 : fLilypondCodeStream (
7     lilypondCodeStream)
8 {
9     fMsrOahGroup = msrOpts;
10    fLpsrOahGroup = lpsrOpts;
11
12    // the LPSR score we're visiting
13    fVisitedLpsrScore = lpsrScore;
14
15    // inhibit the browsing of grace notes groups before,
16    // since they are handled at the note level
17    fVisitedLpsrScore->
18        getMsrScore ()->
19        setInhibitGraceNotesGroupsBeforeBrowsing ();
20
21    // inhibit the browsing of grace notes groups after,

```



```

22 // since they are handled at the note level
23 fVisitedLpsrScore->
24   getMsrScore ()->
25   setInhibitGraceNotesGroupsAfterBrowsing ();

```

The test for browsing inhibition is done in `src/formats/msr/msrNotes.cpp`:

```

1 void msrNote::browseData (basevisitor* v)
2 {
3   // browse the grace notes group before if any
4   if (fNoteGraceNotesGroupBefore) {
5     // fetch the score
6     S_msrScore
7     score =
8       fetchUpLinkToNoteToScore ();
9
10    if (score) {
11      Bool
12      inhibitGraceNotesGroupsBeforeBrowsing =
13        score->
14        getInhibitGraceNotesGroupsBeforeBrowsing ();
15
16      if (inhibitGraceNotesGroupsBeforeBrowsing) {
17        #ifdef TRACING_IS_ENABLED
18          if (
19            gGlobalMsrOahGroup->getTraceMsrVisitors ()
20            ||
21            gGlobalTracingOahGroup->getTraceNotes ()
22            ||
23            gGlobalTracingOahGroup->getTraceGraceNotes ()
24          ) {
25            gLogStream <<
26              "% ==> visiting grace notes groups before is inhibited" <<
27              std::endl;
28          }
29        #endif
30      }
31      else {
32        // browse the grace notes group before
33        msrBrowser<msrGraceNotesGroup> browser (v);
34        browser.browse (*fNoteGraceNotesGroupBefore);
35      }
36    }
37  }
38
39  // ... ..
40 }

```

### 14.6.2 Inhibiting data browsing by options

Choosing which elements to browse can be more selective:

```

1 void msrStaff::browseData (basevisitor* v)
2 {
3   // ... ..
4
5   if (fStaffAllVoicesList.size ()) {
6     for (const S_msrVoice& voice : fStaffAllVoicesList) {
7       // is this voice name in the ignore voices set?
8       Bool ignoreVoice (false);
9
10      std::string voiceName =
11        voice->
12        getVoiceName ();

```

```

13
14     const std::set<std::string>&
15         ignoreMsrVoicesSet =
16         gGlobalMsr2msr0ahGroup->
17             getIgnoreMsrVoicesSet ();
18
19     // ... ..
20
21     if (ignoreMsrVoicesSet.size ()) {
22         ignoreVoice =
23             mfStringIsInStringSet (
24                 voiceName,
25                 ignoreMsrVoicesSet);
26     }
27
28     if (ignoreVoice) {
29 #ifdef TRACING_IS_ENABLED // JMI
30         if (gGlobalTracing0ahGroup->getTraceVoices ()) {
31             gLogStream <<
32                 "Ignoring voice \"" <<
33                 voiceName <<
34                 "\" " <<
35                 std::endl;
36         }
37 #endif
38     }
39
40     else {
41         msrBrowser<msrVoice> browser (v);
42         browser.browse (*voice);
43     }
44 } // for
45 }
46
47 // ... ..
48 }

```

## 14.7 Adapting visitors to data browsing order with booleans

A frequent situation is when the visitor's actions should be tuned depending upon which elements are being visited. In simple case, this can be handled with boolean variables.

For example, <system-layout/> may occur both in the <defaults/> and <print/> MusicXML markups:

```

1 <defaults>
2   <scaling>
3     <millimeters>7.3</millimeters>
4     <tenths>40</tenths>
5   </scaling>
6   <page-layout>
7     <page-height>1534</page-height>
8     <page-width>1151</page-width>
9     <page-margins type="both">
10       <left-margin>54.7945</left-margin>
11       <right-margin>54.7945</right-margin>
12       <top-margin>27.3973</top-margin>
13       <bottom-margin>27.3973</bottom-margin>
14     </page-margins>
15   </page-layout>
16   <system-layout>
17     <system-margins>
18       <left-margin>15</left-margin>
19       <right-margin>0</right-margin>

```

```

20     </system-margins>
21     <system-distance>92.5</system-distance>
22     <top-system-distance>27.5</top-system-distance>
23 </system-layout>
24
25 // ... ..
26
27 <part id="P1">
28     <measure number="1">
29         <print>
30             <system-layout>
31                 <system-margins>
32                     <left-margin>75.625</left-margin>
33                     <right-margin>0</right-margin>
34                 </system-margins>
35                 <top-system-distance>410.9375</top-system-distance>
36             </system-layout>
37             <staff-layout>
38                 <?DoletSibelius JustifyAllStaves=false?>
39                 <?DoletSibelius ExtraSpacesAbove=3?>
40             </staff-layout>
41             <measure-layout>
42                 <measure-distance>20</measure-distance>
43             </measure-layout>
44         </print>

```

To know which element is being visited, we use boolean `fOnGoing*` variables, such as `fOnGoingPrintLayout` in class `msr2mxsrTranslator`.

It is assigned in:

```

1 void msr2mxsrTranslator::visitStart (S_msrPrintLayout& elt)
2 {
3     // ... ..
4
5     fOnGoingPrintLayout = true;
6 }
7
8 void msr2mxsrTranslator::visitEnd (S_msrPrintLayout& elt)
9 {
10    // ... ..
11
12    fOnGoingPrintLayout = false;
13 }

```

and checked for example in:

```

1 void msr2mxsrTranslator::visitStart (S_msrSystemLayout& elt)
2 {
3     // ... ..
4
5     // create a system layout element
6     Sxmlelement
7     systemLayoutElement =
8         createMxmlElement (k_system_layout, "");
9
10    if (fOnGoingPrintLayout) {
11        // append it to the current print element
12        fCurrentPrintElement->push (
13            systemLayoutElement);
14    }
15    else {
16        // don't append it at once to the score defaults element
17        fScoreDefaultsSystemLayoutElement = systemLayoutElement;
18    }

```

When the data browsing order does not fit the needs of a visitor, the latter has to store the values gathered until they can be processed. This occurs for example in `mxsr2msrTranslator`, which uses `fCurrentPrintLayout` for this purpose:

```

1 void mxsr2msrTranslator::visitStart ( S_system_layout& elt )
2 {
3     // ... ..
4
5     // create the system layout
6     fCurrentSystemLayout =
7         msrSystemLayout::create (
8             inputLineNumber);
9
10    fOnGoingSystemLayout = true;
11 }
12
13 void mxsr2msrTranslator::visitEnd ( S_system_layout& elt )
14 {
15     // ... ..
16
17     if (fOnGoingPrint) {
18         // set the current print layout's system layout
19         fCurrentPrintLayout->
20             setSystemLayout (
21                 fCurrentSystemLayout);
22     }
23     else {
24         // set the MSR score system layout
25         fMsrScore->
26             setSystemLayout (
27                 fCurrentSystemLayout);
28     }
29
30     // forget about the current system layout
31     fCurrentSystemLayout = nullptr;
32
33     fOnGoingSystemLayout = false;
34 }

```

## 14.8 Adapting visitors to data browsing order with stacks

In more complex cases, the visiting order leads to have several on-going elements simultaneously. This is the case with class `msrTuplet`, which can be nested.

They are handled in `src/passes/mxsr2msr/mxsr2msrTranslator` and `src/passes/lpsr2lilypond/lpsr2lilypond` for example, using a stack to keep track of them.

`MusicFormats` never uses C++ STL stacks, because they cannot be iterated over:

```

1 std::list<S_msrTuplet>    fOnGoingTupletsStack;

```

```

1 void lpsr2lilypondTranslator::visitStart (S_msrTuplet& elt)
2 {
3     // ... ..
4
5     if (fOnGoingTupletsStack.size ()) {
6         // elt is a nested tuplet
7
8         S_msrTuplet
9             containingTuplet =
10             fOnGoingTupletsStack.top ();
11
12         // unapply containing tuplet factor,

```

```

13 // i.e 3/2 inside 5/4 becomes 15/8 in MusicXML...
14 elt->
15     unapplySoundingFactorToTupletMembers (
16         containingTuplet->
17             getTupletFactor ();
18     )
19
20 // ... ..
21
22 // push the tuplet on the tuplets stack
23 fOnGoingTupletsStack (elt);
24
25 // ... ..
26 }
27
28 void lpsr2lilypondTranslator::visitEnd (S_msrTuplet& elt)
29 {
30     // ... ..
31
32     // pop the tuplet from the tuplets stack
33     fOnGoingTupletsStack ();
34
35     // ... ..
36 }

```

## 14.9 Avoiding the visiting pattern by cascading

There are cases where we need a deterministic traversal of some data handled by MusicFormats. For example, appending a `msrStaffDetails` instance to a part should be cascaded to its staves. It would be an overkill to create a specific browser for this purpose.

This is what method `msrPart::appendStaffDetailsToPart ()` does:

```

1 void msrPart::appendStaffDetailsToPart (
2     const S_msrStaffDetails& staffDetails)
3 {
4     // ... ..
5
6     // register staff details in part
7     fCurrentPartStaffDetails = staffDetails;
8
9     // append staff details to registered staves
10    for (
11        std::map<int, S_msrStaff>::const_iterator i =
12            getPartStaveNumbersToStavesMap.begin ();
13        i != getPartStaveNumbersToStavesMap.end ();
14        ++i
15    ) {
16        S_msrStaff
17            staff = (*i).second;
18
19        staff->
20            appendStaffDetailsToStaff (
21                staffDetails);
22    } // for
23 }

```

Another case is the handling the various elements attached to an class `msrNote` instance, among them chords, grace notes groups and tuplet, all of which contain notes too.

Doing things in the right order can be tricky, see [src/passes/lpsr2lilypond/lpsr2lilypondTranslator.h/.cpp](#).

The time-oriented representation of scores in MSR is also printed by cascading through `printSlices ()` methods, see chapter 20 [MSR time-oriented representation], page 200.

---

## Part IV

# MusicFormats components

## Chapter 15

# MusicFormats components (MFC)

MusicFormats supports keeping the history of its components versions using a dedicated representation, as an alternative to separate release notes. The source files are in [src/components/](#).

### 15.1 Versions numbers

The basic data structure is class `mfcVersionNumber`:

```
1 class    mfcVersionNumber: public smartable
2 {
3     // ... ..
4
5     public:
6
7         // public services
8         // -----
9
10        Bool                operator== (const mfcVersionNumber& other) const;
11
12        Bool                operator!= (const mfcVersionNumber& other) const;
13
14        Bool                operator<  (const mfcVersionNumber& other) const;
15
16        Bool                operator>= (const mfcVersionNumber& other) const;
17
18        Bool                operator>  (const mfcVersionNumber& other) const;
19
20        Bool                operator<= (const mfcVersionNumber& other) const;
21
22    public:
23
24        // print
25        // -----
26
27        std::string          asString () const;
28
29        void                 print (std::ostream& os) const;
30
31    private:
32
33        // fields
34        // -----
35
36        int                  fMajorNumber;
37        int                  fMinorNumber;
38        int                  fPatchNumber;
```

```

39     std::string          fPreRelease;
40 };

```

## 15.2 Versions descriptions

Each component version is described by a class `mfcVersionDescr` instance:

```

1 class   mfcVersionDescr : public smartable
2 {
3     // ... ..
4
5     private:
6
7         // fields
8         // -----
9
10        S_mfcVersionNumber    fVersionNumber;
11        std::string          fVersionDate;
12        std::list<std::string>
13                               fVersionDescriptionItems;
14 };

```

## 15.3 Versions histories

An instance of `mfcVersionsHistory` is essentially a list of `mfcVersionDescr` instances:

```

1 class   mfcVersionsHistory : public smartable
2 {
3     // ... ..
4
5     public:
6
7         // public services
8         // -----
9
10        void                appendVersionDescrToHistory (
11                               const S_mfcVersionDescr& versionDescr);
12
13        S_mfcVersionDescr    fetchMostRecentVersion () const;
14
15        S_mfcVersionNumber    fetchMostRecentVersionNumber () const;
16
17        // ... ..
18
19        protected:
20
21            // protected fields
22            // -----
23
24            std::list<S_mfcVersionDescr>
25                fVersionsList;
26 };

```

The current version of a component is the last one appended to `fVersionsList`:



```

1 S_mfcVersionDescr mfcVersionsHistory::fetchMostRecentVersion () const
2 {
3     // sanity check
4     mfAssert (
5         __FILE__, __LINE__,
6         fVersionsList.size () > 0,
7         "fVersionsList is empty");
8
9     return fVersionsList.back ();
10 }

```

## 15.4 Components descriptions

The components of MusicFormats are described by enumeration type `mfcComponenKind`:

```

1 enum class mfcComponenKind {
2     kComponentRepresentation,
3     kComponentPass,
4     kComponentGenerator,
5     kComponentConverter,
6     kComponentLibrary
7 };

```

The purely virtual class `mfcComponent` is a superclass to the ones describing formats, passes, generators, converters and the MusicFormats library itself:

```

1 class mfcComponent : public smartable
2 {
3     // ... ..
4
5     public:
6
7         // public services
8         // -----
9
10        S_mfcVersionDescr    fetchComponentMostRecentVersion () const
11                                {
12                                    return
13                                        fVersionsHistory->
14                                            fetchMostRecentVersion ();
15                                }
16
17        // ... ..
18
19        public:
20
21            // print
22            // -----
23
24            std::string        asString () const;
25
26            std::string        mostRecentVersionNumberAndDateAsString () const;
27
28            virtual void        print (std::ostream& os) const;
29
30            virtual void        printVersion (std::ostream& os) const;
31            virtual void        printHistory (std::ostream& os) const;
32
33        protected:
34
35            // protected services
36            // -----

```

```

37
38     virtual void                printOwnHistory (std::ostream& os) const;
39
40 protected:
41
42     // protected fields
43     // -----
44
45     std::string                fComponentName;
46
47     mfcComponenKind            fComponenKind;
48
49     S_mfcVersionsHistory        fVersionsHistory;
50 };

```

The virtual `printVersion ()` and `printHistory ()` methods are called by the `--v`, `--version` and `--hist`, `--history` options to the various generators and converters.

Representations and passes have a single, linear history, whereas the generators, the converters and MusicFormats itself use several of them, each with its own history. This leads to a hierarchy of classes:

- class `mfcRepresentationComponent` for formats;
- class `mfcPassComponent` for passes;
- purely virtual class `mfcMultiComponent` for the generators, converters and MusicFormats library, itself the superclass of:
  - class `mfcGeneratorComponent`;
  - class `mfcConverterComponent`;
  - class `mfcLibraryComponent`.

Multi-components have their own history, hence field method `mfcComponent::printOwnHistory ()`. Class `mfcMultiComponent` is described below.

## 15.5 Multi-components

Class `mfcMultiComponent` contains lists of the formats and passes used:

```

1 class    mfcMultiComponent : public mfcComponent
2 {
3     // ... ..
4
5     protected:
6
7     // protected fields
8     // -----
9
10    std::list<S_mfcRepresentationComponent>
11                                fRepresentationComponentsList;
12    std::list<S_mfcPassComponent>
13                                fPassComponentsList;
14
15    // should the version number be at least equal to
16    // the ones of the components?
17    mfcMultiComponentEntropyKind
18                                fComponentEntropyKind;
19
20    mfcMultiComponentUsedFromTheCLIKind
21                                fComponentUsedFromTheCLIKind;
22 };

```

Enumeration type `mfcMultiComponentEntropicityKind` is used to check that the version number of a `mfcMultiComponent` instance is at least equal to the version numbers of the formats and passes it uses:

```
1 enum class mfcMultiComponentEntropicityKind {
2     kComponentEntropicityYes,
3     kComponentEntropicityNo
4 };
```

Enumeration type `mfcMultiComponentUsedFromTheCLIKind` is used to display context sensitive output with the `-version`, `-v` and `-history`, `-hist` options when the library is used from command line tools or through the functional API:

```
1 enum class mfcMultiComponentUsedFromTheCLIKind {
2     kComponentUsedFromTheCLIYes,
3     kComponentUsedFromTheCLINo
4 };
```

This allows for the maintainers of little used tools not to worry about using components with version numbers greater than their own.

Only constant `mfcMultiComponentUsedFromTheCLIKind::kComponentUsedFromTheCLIYes` is used at the time of this writing.

Method `mfcMultiComponent::print ()` displays the regular version numbers:

```
1 jacquesmenu@macmini: ~ > xml2ly -version
2 Command line version of musicxml2lilypond converter v0.9.51 (October 12 2021)
3
4 Representations versions:
5   MXSR
6     v0.9.5 (October 6 2021)
7   MSR
8     v0.9.51 (October 14 2021)
9   LPSR
10    v0.9.5 (October 6 2021)
11
12 Passes versions:
13   mxsr2msr
14     v0.9.5 (October 6 2021)
15   msr2msr
16     v0.9.5 (October 6 2021)
17   msr2lpsr
18     v0.9.5 (October 6 2021)
19   lpsr2lilypond
20     v0.9.5 (October 6 2021)
```

Method `mfcMultiComponent::printHistory ()` displays information analogous to release notes:

```
1 jacquesmenu@macmini: ~ > xml2brl -history
2 Command line version of musicxml2braille converter v0.9.51 (October 12 2021)
3
4 Own history:
5   v0.9.5 (October 6 2021):
6     Start of sequential versions numbering
7
8   v0.9.51 (October 12 2021):
9     Fixed trace OAH issue
10
11 Representations history:
12   MXSR
13     v0.9.5 (October 6 2021):
14       Start of sequential versions numbering
15
16   MSR
17     v0.9.5 (October 6 2021):
```

```

18      Start of sequential versions numbering
19
20      v0.9.51 (October 14 2021):
21          Refined MSR names and summary display options
22
23      BSR
24          v0.9.5 (October 6 2021):
25              Start of sequential versions numbering
26
27      Passes history:
28          mxsr2msr
29              v0.9.5 (October 6 2021):
30                  Start of sequential versions numbering
31
32          msr2msr
33              v0.9.5 (October 6 2021):
34                  Start of sequential versions numbering
35
36          msr2bsr
37              v0.9.5 (October 6 2021):
38                  Start of sequential versions numbering
39
40          bsr2bsr
41              v0.9.5 (October 6 2021):
42                  Start of sequential versions numbering
43
44          bsr2braille
45              v0.9.5 (October 6 2021):
46                  Start of sequential versions numbering

```

## 15.6 Versions history creation

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 "v0.9.61" 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.

The versions history must exist before the `-version`, `-v` and `-history`, `-hist` options are handled. They are thus created early by specific functions, placed in `*Component.h/.cpp` files.

The functions that create them ensure than that is done at most once.

### 15.6.1 Representations and passes components creation

This is done in `create*RepresentationComponent ()` and `create*PassComponent ()` functions, respectively.

For example, MSR versions are handled by function `createMsrRepresentationComponent ()` in `src/formats/msr/msrHistory.h/.cpp`:

```

1 S_mfcRepresentationComponent EXP createMsrRepresentationComponent ()
2 {
3     static S_mfcRepresentationComponent pRepresentationComponent;
4
5     // protect library against multiple initializations
6     if (! pRepresentationComponent) {
7
8     #ifdef TRACING_IS_ENABLED
9         if (gGlobalOahEarlyOptions.getEarlyTraceComponents ()) {
10             gLogStream <<
11                 "Initializing MSR format component" <<
12                 std::endl;
13         }
14     #endif
15
16     // create the format component
17     pRepresentationComponent =
18         mfcRepresentationComponent::create (
19             "MSR");
20
21     // populate it
22     pRepresentationComponent->
23         appendVersionDescrToComponent (
24             mfcVersionDescr::create (
25                 mfcVersionNumber::createFromString ("0.9.50"),
26                 "October 6, 2021",
27                 std::list<std::string> {
28                     "Start of sequential versions numbering"
29                 }
30             ));
31
32     pRepresentationComponent->
33         appendVersionDescrToComponent (
34             mfcVersionDescr::create (
35                 mfcVersionNumber::createFromString ("0.9.51"), // JMI
36                 "October 14, 2021",
37                 std::list<std::string> {
38                     "Refined MSR names and summary display options"
39                 }
40             ));
41     }
42
43     return pRepresentationComponent;
44 }

```

The conversion of MusicXML to MXSR does not belong to MusicFormats since it is provided by libmusicxml2.

### 15.6.2 Generators and converters components creation

In that case, the formats and passes components used by the multi-component should be created as well.

For example, the formats and passes used by the `musicxml2braille` converter are appended to the atoms versions list in its history in function `createMusicxml2brailleConverterComponent ()` in `src/converters/musicxml2braille/musicxml2brailleConverterComponent.cpp`:

```

1 S_mfcConverterComponent EXP createMusicxml2brailleConverterComponent ()
2 {
3     static S_mfcConverterComponent pConverterComponent;
4
5     // protect library against multiple initializations
6     if (! pConverterComponent) {
7
8     #ifdef TRACING_IS_ENABLED

```

```

9      if (gGlobalOahEarlyOptions.getEarlyTraceComponents ()) {
10          gLogStream <<
11              "Creating the musicxml2braille component" <<
12              std::endl;
13      }
14  #endif
15
16      // create the converter component
17      pConverterComponent =
18          mfcConverterComponent::create (
19              "musicxml2braille",
20              mfcMultiComponentEntropyKind::kComponentEntropyNo,
21              mfcMultiComponentUsedFromTheCLIKind::kComponentUsedFromTheCLISYes); // JMI ???
22
23      // populate the converter's own history
24      pConverterComponent->
25          appendVersionDescrToComponent (
26              mfcVersionDescr::create (
27                  mfcVersionNumber::createFromString ("0.9.50"),
28                  "October 6, 2021",
29                  std::list<std::string> {
30                      "Start of sequential versions numbering"
31                  }
32              ));
33
34      pConverterComponent->
35          appendVersionDescrToComponent (
36              mfcVersionDescr::create (
37                  mfcVersionNumber::createFromString ("0.9.51"),
38                  "October 12, 2021",
39                  std::list<std::string> {
40                      "Fixed trace OAH issue"
41                  }
42              ));
43
44      // populate the converter's formats list
45      pConverterComponent->
46          appendRepresentationToMultiComponent (
47              createMxsrRepresentationComponent ());
48      pConverterComponent->
49          appendRepresentationToMultiComponent (
50              createMsrRepresentationComponent ());
51      pConverterComponent->
52          appendRepresentationToMultiComponent (
53              createBsrRepresentationComponent ());
54
55      pConverterComponent->
56          appendPassToMultiComponent (
57              createMxsr2msrComponent ());
58
59      pConverterComponent->
60          appendPassToMultiComponent (
61              createMsr2msrComponent ());
62
63      pConverterComponent->
64          appendPassToMultiComponent (
65              createMsr2bsrComponent ());
66
67      pConverterComponent->
68          appendPassToMultiComponent (
69              createBsr2bsrComponent ());
70
71      pConverterComponent->
72          appendPassToMultiComponent (
73              createBsr2brailleComponent ());
74  }
75

```

```

76     return pConverterComponent;
77 }

```

### 15.6.3 MusicFormats library component creation

This is done in function `createLibraryComponent ()` in `src/utilities/mfcLibraryComponent.h/.cpp`:

```

1  S_mfcLibraryComponent EXP createLibraryComponent ()
2  {
3      static S_mfcLibraryComponent pLibraryComponent;
4
5      // protect library against multiple initializations
6      if (! pLibraryComponent) {
7
8      #ifdef TRACING_IS_ENABLED
9          if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
10             gLogStream <<
11                 "Creating the  MFC library component" <<
12                 std::endl;
13         }
14     #endif
15
16     // create the library's history
17     pLibraryComponent =
18         mfcLibraryComponent::create (
19             "musicformats",
20             mfcMultiComponentEntropicityKind::kComponentEntropicityNo,
21             mfcMultiComponentUsedFromTheCLIKind::kComponentUsedFromTheCLISYes); // JMI ???
22
23     // populate the library's own history
24     pLibraryComponent->
25         appendVersionDescrToComponent (
26             mfcVersionDescr::create (
27                 mfcVersionNumber::createFromString ("0.9.50"),
28                 "October 6, 2021",
29                 std::list<std::string> {
30                     "Start of sequential versions numbering"
31                 }
32             ));
33
34     pLibraryComponent->
35         appendVersionDescrToComponent (
36             mfcVersionDescr::create (
37                 mfcVersionNumber::createFromString ("0.9.51"),
38                 "October 12, 2021",
39                 std::list<std::string> {
40                     "Adding a version number to the MusicFormats library",
41                     "Fixed trace OAH issue in the musicxml2* converters)"
42                 }
43             ));
44
45     pLibraryComponent->
46         appendVersionDescrToComponent (
47             mfcVersionDescr::create (
48                 mfcVersionNumber::createFromString ("0.9.52"),
49                 "October 12, 2021",
50                 std::list<std::string> {
51                     "Added MusicFormats library versions history to '-hist, -history'"
52                 }
53             ));
54
55     pLibraryComponent->
56         appendVersionDescrToComponent (
57             mfcVersionDescr::create (

```

```

58         mfcVersionNumber::createFromString ("0.9.53"),
59         "October 22, 2021",
60         std::list<std::string> {
61             "Replaced bool by class Bool in variables and fields",
62             "Created MFC (MusicFormats components)"
63         }
64     ));
65
66     pLibraryComponent->
67         appendVersionDescrToComponent (
68             mfcVersionDescr::create (
69                 mfcVersionNumber::createFromString ("0.9.54"),
70                 "November 6, 2021",
71                 std::list<std::string> {
72                     "Replaced std::cout and std::cerr by gOutputStream and gLogStream respectively
73                     in the CLI samples",
74                     "Finalized components numbering (MFC)"
75                 }
76             ));
77
78     // populate the library's components history
79     pLibraryComponent->
80         appendRepresentationToMultiComponent (
81             createMsrRepresentationComponent ());
82     pLibraryComponent->
83         appendRepresentationToMultiComponent (
84             createLpsrRepresentationComponent ());
85     pLibraryComponent->
86         appendRepresentationToMultiComponent (
87             createBsrRepresentationComponent ());
88     pLibraryComponent->
89         appendRepresentationToMultiComponent (
90             createMxsrRepresentationComponent ());
91
92     pLibraryComponent->
93         appendPassToMultiComponent (
94             createMsr2msrComponent ());
95
96     pLibraryComponent->
97         appendPassToMultiComponent (
98             createMsr2lpsrComponent ());
99     pLibraryComponent->
100         appendPassToMultiComponent (
101             createLpsr2lilypondComponent ());
102
103     pLibraryComponent->
104         appendPassToMultiComponent (
105             createMsr2bsrComponent ());
106     pLibraryComponent->
107         appendPassToMultiComponent (
108             createBsr2bsrComponent ());
109     pLibraryComponent->
110         appendPassToMultiComponent (
111             createBsr2brailleComponent ());
112
113     pLibraryComponent->
114         appendPassToMultiComponent (
115             createMsr2mxsrComponent ());
116
117     pLibraryComponent->
118         appendPassToMultiComponent (
119             createMxsr2musicxmlComponent ());
120
121     pLibraryComponent->
122         appendPassToMultiComponent (
123             createMxsr2guidoComponent ());

```



```

124
125     return pLibraryComponent;
126 }

```

Functions `createLibraryComponent ()` is called in `src/clisamples/displayMusicformatsVersion.cpp` and `src/clisamples/displayMusicformatsHistory.cpp`.

#### 15.6.4 Version and history options handling

In order to be able to execute the `-version`, `-v` and `-history`, `-hist` options of a generator or converter, a `oahHandler` instance must be supplied with a `mfcMultiComponent` instance.

Field `oahHandler::fHandlerMultiComponent` is used for this purpose:

```

1 // -----
2 class EXP oahHandler : public smartable
3 {
4     // ... ..
5
6     protected:
7
8         // protected initialization
9         // -----
10
11         virtual void          initializeHandlerMultiComponent () = 0;
12
13     public:
14
15         // set and get
16         // -----
17
18         // ... ..
19
20         S_mfcMultiComponent    getHandlerMultiComponent () const
21                                 { return fHandlerMultiComponent; }
22
23         // ... ..
24
25     protected:
26
27         // protected fields
28         // -----
29
30         // ... ..
31
32         // compound versions
33         S_mfcMultiComponent    fHandlerMultiComponent;
34 };

```

Field `oahHandler::fHandlerMultiComponent` is set in the `oahHandler` sub-classes constructors by a call to the overridden `initializeHandlerMultiComponent ()`.

For example in constructor `xml2xmlInsiderHandler::xml2xmlInsiderHandler ()`:

```

1 xml2xmlInsiderHandler::xml2xmlInsiderHandler (
2     const std::string& serviceName,
3     std::string handlerHeader)
4 : oahInsiderHandler (
5     serviceName,
6     handlerHeader,
7     R"(
8         Welcome to the MusicXML to MusicXML converter
9         delivered as part of the MusicFormats library.

```

```

10
11      --- https://github.com/jacques-menu/musicformats ---
12  )",
13  R"(
14  Usage: xml2xml [[option]* [MusicXMLFile|-] [[option]*
15  )")
16  {
17      // ... ..
18
19      // initialize the multi-component
20      initializeHandlerMultiComponent ();
21
22      // ... ..
23  }

```

The overridden `initializeHandlerMultiComponent ()` methods merely get the atom or compound versions to assign it to field `oahHandler::fHandlerMultiComponent`.

For example, for `Mikrokosmos3Wandering`, the compound versions is simply set in the corresponding insider class `Mikrokosmos3WanderingInsiderHandler`:

```

1 void Mikrokosmos3WanderingInsiderHandler::initializeHandlerMultiComponent ()
2 {
3     fHandlerMultiComponent =
4         createMikrokosmos3WanderingGeneratorComponent ();
5 }

```

## 15.7 Accessing versions in regular handlers

A regular handler merely gets the compound versions of the insider handler it relies upon in its overridden `initializeHandlerMultiComponent ()` method:

```

1 class EXP oahRegularHandler : public oahHandler
2 /*
3  A regular OAH handler relies on the existence of so-called 'insider' handler,
4  that contains all the options values gathered from the user,
5  grouped according to the internal representations and passes used.
6
7  The variables containing the values of the options chosen by the user
8  are actually held by the insider handler.
9  */
10 {
11     // ... ..
12
13     protected:
14
15         // protected initialization
16         // -----
17
18         // ... ..
19
20         void initializeHandlerMultiComponent () override
21         {
22             fHandlerMultiComponent =
23                 fInsiderHandler->
24                     getHandlerMultiComponent ();
25         }
26
27         // ... ..
28 };

```

## 15.8 Getting current version numbers

Apart from the version and history options, such current version numbers may be used in the output from generators and converters, depending on the options. A component description is the way to achieve that in the latter two cases.

### 15.8.1 Current version numbers in options

Option `-version`, `-v` displays the versions of generators and converters:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/musicxml > xml2xml -version
2 Command line version of musicxml2musicxml converter v0.9.51 (October 12 2021)
3
4 Representations versions:
5   MXSR
6     v0.9.5 (October 6 2021)
7   MSR
8     v0.9.51 (October 14 2021)
9
10 Passes versions:
11   mxsr2msr
12     v0.9.5 (October 6 2021)
13   msr2msr
14     v0.9.5 (October 6 2021)
15   msr2mxsr
16     v0.9.5 (October 6 2021)
17   mxsr2musicxml
18     v0.9.5 (October 6 2021)

```

Option `-history`, `-hist` display the versions history of generators and converters:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/musicxml > xml2gmn -history
2 Command line version of musicxml2guido converter v0.9.51 (October 12 2021)
3
4 Own history:
5   v0.9.5 (October 6 2021):
6     Start of sequential versions numbering
7
8   v0.9.51 (October 12 2021):
9     Fixed trace OAH issue
10
11 Representations history:
12   MXSR
13     v0.9.5 (October 6 2021):
14       Start of sequential versions numbering
15
16   MSR
17     v0.9.5 (October 6 2021):
18       Start of sequential versions numbering
19
20     v0.9.51 (October 14 2021):
21       Refined MSR names and summary display options
22
23 Passes history:
24   mxsr2msr
25     v0.9.5 (October 6 2021):
26       Start of sequential versions numbering
27
28   msr2msr
29     v0.9.5 (October 6 2021):
30       Start of sequential versions numbering
31
32   msr2mxsr

```

```

33     v0.9.5 (October 6 2021):
34         Start of sequential versions numbering
35
36     mxsr2guido
37     v0.9.5 (October 6 2021):
38         Start of sequential versions numbering

```

In `src/oah/oahAtomsCollection.h/.cpp`, class `oahVersionAtom` contains method `printVersion ()`:

```

1  class EXP oahVersionAtom : public oahPureHelpAtomWithoutAValue
2  {
3      // ... ..
4
5      public:
6
7          // public services
8          // -----
9
10         void                applyElement (std::ostream& os) override;
11
12         // ... ..
13
14     public:
15
16         // print
17         // -----
18
19         // ... ..
20
21         void                printVersion (std::ostream& os) const;
22 };

```

The option is applied by method `oahVersionAtom::applyElement ()`:

```

1  void oahVersionAtom::applyElement (std::ostream& os)
2  {
3      #ifdef TRACING_IS_ENABLED
4          if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
5              gLogStream <<
6                  "==> option ' ' << fetchNames () << " ' is a oahVersionAtom" <<
7                  std::endl;
8          }
9      #endif
10
11      int saveIndent = gIndenter.getIndent ();
12
13      gIndenter.resetToZero ();
14
15      printVersion (os);
16
17      gIndenter.setIndent (saveIndent);
18 }

```

The work is done by method `oahVersionAtom::printVersion ()`:

```

1  void oahVersionAtom::printVersion (std::ostream& os) const
2  {
3      // get the handler version
4      S_mfcMultiComponent
5          handlerMultiComponent =
6              fetchAtomUpLinkToHandler ()->
7                  getHandlerMultiComponent ();
8
9      // sanity check
10     mfAssert (

```

```

11  __FILE__, __LINE__,
12  handlerMultiComponent != nullptr,
13  "handlerMultiComponent is null");
14
15  handlerMultiComponent->
16  printVersion (os);
17 }

```

The situation is analog for histories with `printVersion ()` replaced by `printHistory ()`.

### 15.8.2 Current version numbers in formats

When creating LilyPond output, the current version number of the converter used is indicated as a comment when the option `-lilypond-generation-infos`, `-lpgi` option is used:

```

1  jacquesmenu@macmini: ~/musicformats-git-dev/musicxml > xml2ly --lilypond-generation-infos
   basic/HelloWorld.xml
2  \version "2.22.0"
3
4  % Pick your choice from the next two lines as needed
5  %myBreak = { \break }
6  myBreak = {}
7
8  % Pick your choice from the next two lines as needed
9  %myPageBreak = { \pageBreak }
10 myPageBreak = {}
11
12 % Generated by xml2ly v0.9.51 (October 12 2021)
13 % on Thursday 2021-11-11 @ 11:15:56 CET
14 % from "basic/HelloWorld.xml"
15
16 % ... ..

```

Class `lpsrScore` contains an MFC component field:

```

1  class EXP lpsrScore : public lpsrElement
2  {
3      // ... ..
4
5      private:
6
7          // private fields
8          // -----
9
10         // ... ..
11
12         // the multi-component
13         // -----
14         S_mfcMultiComponent    fMultiComponent;
15
16         // ... ..
17 };

```

In `src/formats/lpsr//lpsrScores.cpp`, constructor `lpsrScore::lpsrScore ()` stores the multi-component value and uses it to create an `lpsrComment` instance:

```

1  lpsrScore::lpsrScore (
2      int                inputLineNumber,
3      const S_msrScore&  theMsrScore,
4      const S_mfcMultiComponent& multiComponent)
5      : lpsrElement (inputLineNumber)
6  {
7      // ... ..

```

```

8
9   fMsrScore = theMsrScore;
10
11  fMultiComponent = multiComponent;
12
13  // should the initial comments about the service and the options used
14  // be generated?
15  if (gGlobalLpsr2lilypondOahGroup->getXml2lyInfos ()) {
16      // create the 'input source name and translation date' comment
17      {
18          std::stringstream s;
19
20          s <<
21              "Generated by " <<
22              gGlobalOahOahGroup->getOahOahGroupServiceName () <<
23              ' ' <<
24              fMultiComponent->
25                  mostRecentVersionNumberAndDateAsString () <<
26              std::endl <<
27
28              "% on " <<
29              gGlobalServiceRunData->getTranslationDateFull () <<
30              std::endl <<
31
32              "% from ";
33
34          if (gGlobalServiceRunData->getInputSourceName () == "-") {
35              s << "standard input";
36          }
37          else {
38              s << "\"" << gGlobalServiceRunData->getInputSourceName () << "\"";
39          }
40
41          fInputSourceNameComment =
42              lpsrComment::create (
43                  inputLineNumber,
44                  s.str (),
45                  lpsrCommentGapAfterwardsKind::kCommentGapAfterwardsNo);
46      }
47
48      // ... ..
49  }
50
51  // ... ..
52  }

```

### 15.8.3 Current version numbers in passes

Another case is that of the generation of MusicXML output:

```

1  jacquesmenu@macmini: ~/musicformats-git-dev/musicxml > xml2xml -musicxml-generation-infos
   basic/HelloWorld.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      Created by xml2xml v0.9.5 (October 6 2021)
9      on Thursday 2021-11-11 @ 11:04:06 CET
10     from basic/HelloWorld.xml
11     =====
12     -->
13     <work>

```

```

14     <work-number/>
15     <work-title>Hello World!</work-title>
16 </work>
17 <movement-number/>
18 <movement-title/>
19 <identification>
20     <encoding>
21         <software>xml2xml v0.9.5 (October 6 2021), https://github.com/jacques-menu/
musicformats</software>
22         <encoding-date>2021-11-10</encoding-date>
23     </encoding>
24     <miscellaneous>
25         <miscellaneous-field name="description"/>
26     </miscellaneous>
27 </identification>
28
29 <!-- ... .. -->

```

In `src/passes/msr2mxsr/msr2mxsrTranslator.cpp`, the start visitor of `msrScore` instances does that this way:

```

1 void msr2mxsrTranslator::visitStart (S_msrScore& elt)
2 {
3     // ... ..
4
5     // get the pass component
6     S_mfcPassComponent
7     passComponent =
8         createMsr2mxsrComponent ();
9
10    // get the pass component current version number and date
11    std::string
12    passComponentMostRecentVersionNumberAndDateAsString =
13        passComponent->
14            mostRecentVersionNumberAndDateAsString ();
15
16    // create the initial creation comment
17    std::stringstream s;
18    s <<
19    std::endl <<
20    "===== " <<
21    std::endl <<
22    "Created by " <<
23    gGlobalOahOahGroup->getOahOahGroupServiceName () <<
24    ' ' <<
25    passComponentMostRecentVersionNumberAndDateAsString <<
26    std::endl <<
27
28    "on " <<
29    gGlobalServiceRunData->getTranslationDateFull () <<
30    std::endl <<
31
32    "from " <<
33    gGlobalServiceRunData->getInputSourceName () <<
34    std::endl <<
35
36    "===== " <<
37    std::endl;
38
39    // append the initial creation comment to the score part wise element
40    fResultingMusicxmlelement->push (createMxmlelement (kComment, s.str ()));
41
42    // create a software element
43    Sxmlelement
44    softwareElement =
45        createMxmlelement (

```

```
46     k_software ,
47     gGlobalOahOahGroup->getOahOahGroupServiceName ()
48     + ' '
49     + passComponentMostRecentVersionNumberAndDateAsString +
50     ", https://github.com/jacques-menu/musicformats";
51
52 // append it to the identification encoding
53 appendToScoreIdentificationEncoding (softwareElement);
54
55 // ... ..
56 }
```



---

# Part V

## Options and help

## Chapter 16

# Options and help (OAH)

OAH is a powerful way of representing the options together with the corresponding help. The classical function `getopt*()` family of functions are not up to the task because:

- there is a great number of options in MusicFormats;
- attaching the help to the options in a clean, neat way was highly desirable;
- more important still, the re-use of options whenever translators are combined into converters could only be achieved with an object oriented handling of the options and help.

The output of the help goes to standard output, so that the user can pipe it into a `more` or `less` command.

### 16.1 OAH basics

- 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!"
- options handling is organized as a hierarchical, introspective set of classes. An options and its corresponding help are grouped in a single object.
- the options can be supplied thru:
  - the command line, in `argv`. This allows for mixed options and arguments in any order, à la GNU;
  - the API functions such as function `musicxmlfile2lilypond()`, in an options and arguments.
- class `oahElement` is the super-class of all options types, including groups and subgroups. It contains a short name and a long name, as well as a description. Short and long names can be used and mixed at will in the command line and in option vectors (API), as well as `'-'` and `'--'`. The short name is mandatory, but the long name may be empty if the short name is explicit enough.
- prefixes such `'-t='` and `-help=` allow for a contracted form of options. For example, `-t=meas,notes` is short for `'-t-meas, -tnotes'`. A `oahPrefix` contains the prefix name, the ersatz by which to replace it, and a description.
- a class `oahHandler` contains a list of `oahGroup`'s, each handled in a pair of `.h/.cpp` files such as `src/formats/msr/msrOah.h` and `src/formats/msr/msrOah.cpp`, and a list of options prefixes.
- a class `oahGroup` contains a list of `oahSubGroup`'s and an `upLink` to the containing `oahHandler`.
- a class `oahSubGroup` contains a list of `oahAtom`'s and an `upLink` to the containing `oahGroup`.
- each class `oahAtom` contains an atomic option and the corresponding help, and an `upLink` to the containing `oahSubGroup`.

## 16.2 Features

- the values of the various options can be displayed with the option `-display-options-values`, `-dov` option;
- partial help can be obtained, i.e. help about any group, subgroup or atom, showing the path in the hierarchy down to the corresponding option;
- there are various sub-classes of class `oahAtom` such as class `oahIntegerAtom`, class `oahBooleanAtom` and class `oahRationalAtom`, to control options values of common types;
- class `oahThreeBooleansAtom`, for example, allows for three boolean settings to be controlled at once with a single option;
- class `oahAtomStoringAValue` describes options for which a value is supplied in the command line or in option vectors (API);
- a class such as class `lpsrPitchesLanguageAtom` is used to supply a `std::string` value to be converted into an internal enumerated type;
- a class `oahCombinedBooleansAtom` contains a list of boolean atoms to manipulate several such atoms as a single one, see the 'class `cubase`' combined booleans atom in `src/passes/mxsr2msr/mxsr2msrOah.cpp`;
- class `oahMultiplexBooleansAtom` contains a list of boolean atoms sharing a common prefix to display such atoms in a compact manner, see the 'ignore-redundant-clefs' multiplex booleans atom in `src/passes/mxsr2msr/mxsr2msrOah.cpp`;
- storing options and the corresponding help in class `oahGroup`'s makes it easy to re-use them. For example, file `xml2ly` and file `xml2lbr` have their three first passes in common, (up to obtaining the MSR description of the score), as well as the corresponding options and help;
- `src/oah/oahAtomsCollection` contains a bunch of general purpose options such as class `oahContactAtom`, class `oahFloatAtom` and class `oahLengthAtom`;
- a regular handler (used by default unless the option `-insider` is used), presents the options and help grouped by subject, such as voices and tuplets. It uses an insider handler, which groups them by internal representation and conversion pass. This is how options groups are re-used for various converters such as file `xml2ly`, file `xml2brl` and file `xml2xml`.

## 16.3 OAH classes inheritance

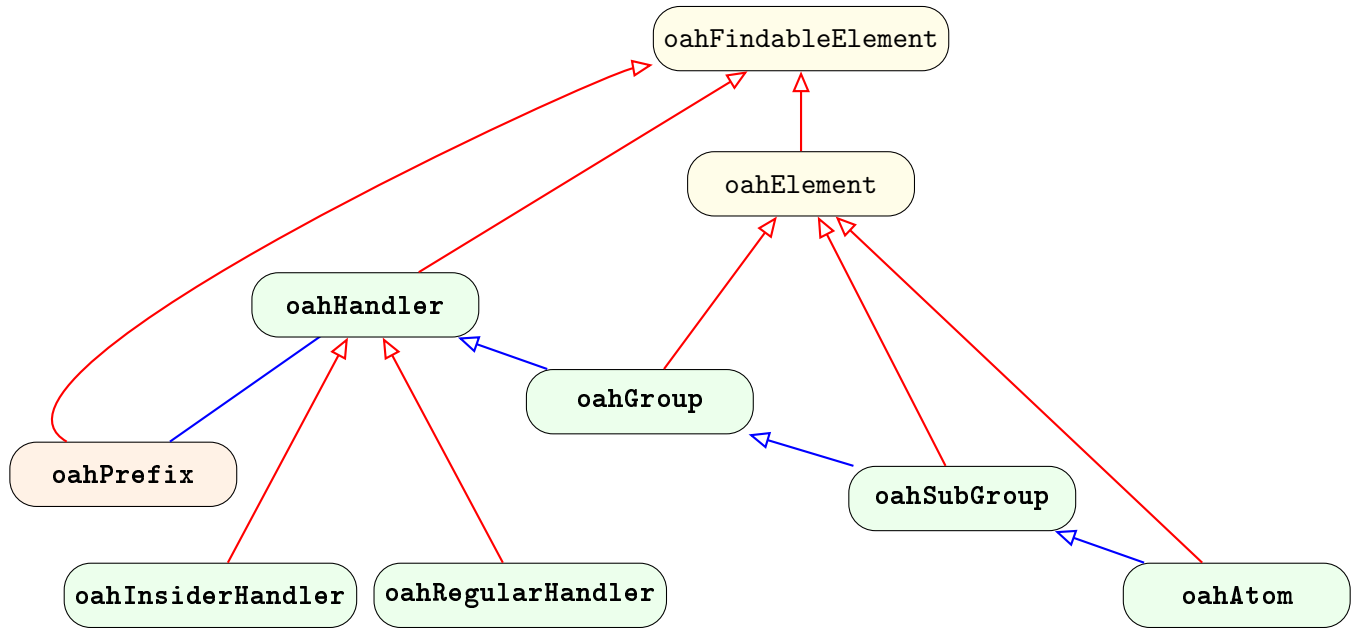
The picture at figure 16.1 [The OAH classes hierarchy], page 120, shows the hierarchy of the main OAH classes. The colors are used as follows:

The background colors are used as follows:

- **green**: a OAH element that is expected to be found in an options and help user view, such as class `oahHandler` and class `oahSubGroup`;
- **pink**: a complementary element provided by OAH, such as class `oahPrefix`;
- **yellow**: a base class with name class `oah*Element` for elements that can be used in another class, such as class `oahFindableElement`;

The arrows colors have the following meaning:

Figure 16.1: The OAH classes hierarchy



- **red:** a link from a class to its base class. For example, class `oahElement` is derived from class `oahFindableElement`;
- **blue:** a link from a class to another that uses smart pointers to one or more instances of the former. For example, an `msrTuplet` instance may be an element of an `msrGraceNotesGroup` instance.

There is a whole hierarchy of `oahAtom` sub-classes, some of which are provided in `src/oah/oahAtomsCollection.h.h/`. Here is the essentials of the OAH classes:

```

1 jacquesmenu@macmini:~/musicformats-git-dev/src/oah > gr Element | grep class
2 oahAtomsCollection.h:2085:class EXP oahNaturalNumbersSetElementAtom : public
   oahAtomStoringAValue
3 oahAtomsCollection.h:2271:class EXP oahIntSetElementAtom : public oahAtomStoringAValue
4 oahAtomsCollection.h:2357:class EXP oahStringSetElementAtom : public oahAtomStoringAValue
5 oahAtomsCollection.h:2450:class EXP oahStringToIntMapElementAtom : public
   oahAtomStoringAValue
6 oahAtomsCollection.h:2537:class EXP oahStringToStringMapElementAtom : public
   oahAtomStoringAValue
7 oahAtomsCollection.h:2624:class EXP oahStringToStringMultiMapElementAtom : public
   oahAtomStoringAValue
8 oahBasicTypes.h:373:class EXP oahPrefix : public oahFindableElement
9 oahBasicTypes.h:472:class EXP oahAtom : public oahElement
10 oahBasicTypes.h:994:class EXP oahSubGroup : public oahElement
11 oahBasicTypes.h:1163:class EXP oahGroup : public oahElement
12 oahBasicTypes.h:1396:class EXP oahHandler : public oahElement
13 oahElements.h:36:enum class oahElementValueKind {
14 oahElements.h:48:enum class oahElementVisibilityKind {
15 oahElements.h:60:enum class oahElementHelpOnlyKind {
16 oahElements.h:72:class oahElement;
17 oahElements.h:82:class EXP oahFindableElement : public smartable
18 oahElements.h:206:class EXP oahElement : public oahFindableElement
19 oahElements.h:391:class EXP oahElementUse : public smartable

```

Class `oahFindableElement` is the base class for all those that can be introspected with the `-find` option.

## 16.4 Atoms expecting a value

Some options expect a value, such a length or a color, to be supplied in the command line or in a type `oahOptionsVector`.

Purely virtual class `oahAtomExpectingAValue`, defined in `src/oah/oahBasicTypes.h/.cpp`, is a common ancestor to all the classes describing such options:

```

1 class EXP oahAtomExpectingAValue : public oahAtom
2 /*
3  a purely virtual common ancestor for all atom classes
4  that take a value from argv or an oahOptionsVector
5 */
6 {
7     // ... ..
8
9     public:
10
11         // public services
12         // -----
13
14         void                applyElement (std::ostream& os) override;
15                             // reports an error
16
17         virtual void        applyAtomWithValue (
18                             const std::string& theString,
19                             std::ostream&      os) = 0;
20
21         virtual void        applyAtomWithDefaultValue (std::ostream& os);
22                             // used only if fElementValueKind
23                             // is oahElementValueKind::kElementValueImplicit
24                             // or oahElementValueKind::kElementValueOptional
25 };

```

The classes derived from `oahAtomExpectingAValue` are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grh 'public oahAtomExpectingAValue'
2 oah/oahBasicTypes.h:class EXP oahAtomStoringAValue : public oahAtomExpectingAValue
3 oah/oahBasicTypes.h:class EXP oahPureHelpAtomExpectingAValue : public
4   oahAtomExpectingAValue
5 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondRelativeOctaveEntryAtom : public
6   oahAtomExpectingAValue
7 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondFixedOctaveEntryAtom : public
8   oahAtomExpectingAValue

```

### 16.4.1 The `oahAtomStoringAValue` class

Purely virtual class `oahAtomStoringAValue`, defined in `src/oah/oahBasicTypes.h/.cpp`, is the base class for them:

```

1 class EXP oahAtomStoringAValue : public oahAtomExpectingAValue
2 {
3     // ... ..
4
5     public:
6
7         // print
8         // -----
9
10        virtual void        printAtomWithVariableEssentials (
11                             std::ostream& os,
12                             int fieldWidth) const;
13
14        virtual void        printAtomWithVariableEssentialsShort (

```

```

14         std::ostream& os,
15         int fieldWidth) const;
16
17     void        print (std::ostream& os) const override;
18     void        printFull (std::ostream& os) const override;
19
20     void        printHelp (std::ostream& os) const override;
21
22     virtual void printAtomWithVariableOptionsValues (
23         std::ostream& os,
24         int        valueFieldWidth) const override;
25
26 protected:
27
28     // protected fields
29     // -----
30
31     std::string      fValueSpecification;
32
33     std::string      fVariableName;
34     Bool            fSetByAnOption;
35 };

```

The field `oahAtomStoringAValue::fSetByAnOption` is necessary because some value types do not have an obvious 'neutral' element. This is the case for a note's octave, `oahLengthUnitKindAtom` and `oahColorRGBAtom`, for example. It is not used for data structures such as sets, lists and vector, since this is indicated by their size.

`fSetByAnOption` is set in `set*Variable ()` methods, as in method `oahIntegerAtom::setIntegerVariable ()` in `src/oah/oahAtomsCollection.cpp`:

```

1 void oahIntegerAtom::setIntegerVariable (int value)
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
5             gLogStream <<
6                 "Setting option '" <<
7                 fetchNames () <<
8                 "' integer variable to '" <<
9                 value <<
10                "' <<
11                std::endl;
12         }
13     #endif
14
15     fIntegerVariable = value;
16     fSetByAnOption = true;
17 }

```

### 16.4.2 The `oahBooleanAtom` special case

Class `oahBooleanAtom` has its own `fSetByAnOption` field, because is it not derived from class `oahAtomStoringAValue`: there isn't any value to be supplied in the command line, since `fBooleanVariable` is false by default:

```

1 // -----
2 class EXP oahBooleanAtom : public oahAtom
3 {
4     /*
5     an atom controlling a Bool variable variableName,
6     but expecting no value to be supplied:
7     the variable is false initially,

```

```

8  and is set to true by the mere occurrence of the atom
9  */
10
11  // ... ..
12
13  protected:
14
15  // protected fields
16  // -----
17
18  std::string          fVariableName;
19  Bool&                fBooleanVariable;
20  Bool                 fSetByAnOption;
21 };

```

### 16.4.3 Checking whether an option has been selected

This is done by calls to the `getSetByAnOption ()` methods.

For example, method `lpsr2lilypondTranslator::generateLilypondVersion ()` in `src/formats/lpsr//lpsr2lilypondTranslator.cpp` contains:

```

1  void lpsr2lilypondTranslator::generateLilypondVersion ()
2  {
3      // LilyPond version
4      Bool
5          lilypondVersionHasBeenSet =
6          gGlobalLpsr2lilypondOahGroup->
7          getLilypondVersionAtom ()->
8          getSetByAnOption ();
9
10     std::string
11         lilypondVersion =
12         lilypondVersionHasBeenSet
13         ? gGlobalLpsr2lilypondOahGroup->
14         getLilypondVersion ()
15         : gGlobalLpsr2lilypondOahGroup->
16         getLilypondVersionDefaultValue ();
17
18     fLilypondCodeStream <<
19     "\\version \"" <<
20     lilypondVersion <<
21     "\"" <<
22     std::endl << std::endl;
23 }

```

The default LilyPond version number is 2.22.0. Another can be chosen with the `-lilypond-version`, `-lpv` option:

```

1  jacquesmenu@macmini > xml2ly -find lilypond-version
2  1 occurrence of std::string "lilypond-version" has been found:
3      1:
4      -lilypond-version, -lpv
5      Set the Lilypond '\version' to STRING in the Lilypond code.
6      The default is '2.22.0'.

```

### 16.4.4 The oahAtomStoringAValue sub-classes

The classes derived from oahAtomStoringAValue are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grh 'public oahAtomStoringAValue'
2 oah/harmoniesExtraOah.h:class EXP extraShowAllHarmoniesStructuresAtom : public
   oahAtomStoringAValue
3 oah/harmoniesExtraOah.h:class EXP extraShowAllHarmoniesContentsAtom : public
   oahAtomStoringAValue
4 oah/harmoniesExtraOah.h:class EXP extraShowHarmonyDetailsAtom : public
   oahAtomStoringAValue
5 oah/harmoniesExtraOah.h:class EXP extraShowHarmonyAnalysisAtom : public
   oahAtomStoringAValue
6 oah/oahAtomsCollection.h:class EXP oahIntegerAtom : public oahAtomStoringAValue
7 oah/oahAtomsCollection.h:class EXP oahFloatAtom : public oahAtomStoringAValue
8 oah/oahAtomsCollection.h:class EXP oahStringAtom : public oahAtomStoringAValue
9 oah/oahAtomsCollection.h:class EXP oahRationalAtom : public oahAtomStoringAValue
10 oah/oahAtomsCollection.h:class EXP oahNaturalNumbersSetElementAtom : public
   oahAtomStoringAValue
11 oah/oahAtomsCollection.h:class EXP oahColorRGBAtom : public oahAtomStoringAValue
12 oah/oahAtomsCollection.h:class EXP oahIntSetElementAtom : public oahAtomStoringAValue
13 oah/oahAtomsCollection.h:class EXP oahStringSetElementAtom : public oahAtomStoringAValue
14 oah/oahAtomsCollection.h:class EXP oahStringToIntMapElementAtom : public
   oahAtomStoringAValue
15 oah/oahAtomsCollection.h:class EXP oahStringAndIntegerAtom : public oahAtomStoringAValue
16 oah/oahAtomsCollection.h:class EXP oahStringAndTwoIntegersAtom : public
   oahAtomStoringAValue
17 oah/oahAtomsCollection.h:class EXP oahLengthUnitKindAtom : public oahAtomStoringAValue
18 oah/oahAtomsCollection.h:class EXP oahLengthAtom : public oahAtomStoringAValue
19 oah/oahAtomsCollection.h:class EXP oahMidiTempoAtom : public oahAtomStoringAValue
20 formatsgeneration/brailleGeneration/brailleGenerationOah.h:class EXP brailleOutputKindAtom
   : public oahAtomStoringAValue
21 formatsgeneration/brailleGeneration/brailleGenerationOah.h:class EXP brailleUTFKindAtom :
   public oahAtomStoringAValue
22 formatsgeneration/brailleGeneration/brailleGenerationOah.h:class EXP
   brailleByteOrderingKindAtom : public oahAtomStoringAValue
23 formatsgeneration/msrGeneration/msrGenerationBasicTypes.h:class EXP
   msrGenerationAPIKindAtom : public oahAtomStoringAValue
24 formatsgeneration/multiGeneration/mfMultiGenerationOah.h:class EXP
   mfMultiGenerationOutputKindAtom : public oahAtomStoringAValue
25 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondScoreOutputKindAtom : public
   oahAtomStoringAValue
26 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondTransposePartNameAtom : public
   oahAtomStoringAValue
27 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondTransposePartIDAtom : public
   oahAtomStoringAValue
28 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondAccidentalStyleKindAtom : public
   oahAtomStoringAValue
29 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondChordsDisplayAtom : public
   oahAtomStoringAValue
30 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondLyricsDurationsKindAtom : public
   oahAtomStoringAValue
31 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP
   lilypondDynamicsTextSpannersStyleKindAtom : public oahAtomStoringAValue
32 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondBreakPageAfterMeasureNumberAtom
   : public oahAtomStoringAValue
33 passes/msr2lpsr/msr2lpsrOah.h:class EXP msr2lpsrScoreOutputKindAtom : public
   oahAtomStoringAValue
34 passes/msr2msr/msr2msrOah.h:class EXP msrIgnorePartAtom : public oahAtomStoringAValue
35 passes/msr2msr/msr2msrOah.h:class EXP msrKeepPartAtom : public oahAtomStoringAValue
36 passes/mxsr2msr/mxsr2msrOah.h:class EXP msrReplaceClefAtom : public oahAtomStoringAValue
37 formats/bsr/bsrOah.h:class EXP bsrFacSimileKindAtom : public oahAtomStoringAValue
38 formats/bsr/bsrOah.h:class EXP bsrTextsLanguageAtom : public oahAtomStoringAValue
39 formats/lpsr/lpsrOah.h:class EXP lpsrPitchesLanguageAtom : public oahAtomStoringAValue
40 formats/lpsr/lpsrOah.h:class EXP lpsrChordsLanguageAtom : public oahAtomStoringAValue
41 formats/lpsr/lpsrOah.h:class EXP lpsrTransposeAtom : public oahAtomStoringAValue

```



```

42 formats/msdl/msdlInputOah.h:class EXP msdlKeywordsLanguageAtom : public
    oahAtomStoringAValue
43 formats/msdl/msdlInputOah.h:class EXP msdlCommentsTypeAtom : public oahAtomStoringAValue
44 formats/msdl/msdlInputOah.h:class EXP msdlUserLanguageAtom : public oahAtomStoringAValue
45 formats/msdl/msdlInputOah.h:class EXP msdlPitchesLanguageAtom : public
    oahAtomStoringAValue
46 formats/msr/msrOah.h:class EXP msrPitchesLanguageAtom : public oahAtomStoringAValue
47 formats/msr/msrOah.h:class EXP msrRenamePartAtom : public oahAtomStoringAValue

```

## 16.5 Pure help atoms

Some options, such as `-a`, `-about`, only provide help to the user. Such pure help atoms can be with or without a value.

### 16.5.1 Pure help atoms without a value

The base `oahPureHelpAtomWithoutAValue` class is defined in `src/oah/oahBasicTypes.h/.cpp`:

```

1 // -----
2 class EXP oahPureHelpAtomWithoutAValue : public oahAtom
3 {
4     // ... ..
5
6     protected:
7
8     // protected fields
9     // -----
10
11     std::string          fHelpAtomWithoutAValueServiceName;
12 };

```

The actual pure help atoms without a value are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grh 'public oahPureHelpAtomWithoutAValue
,
2 oah/oahAtomsCollection.h:class EXP oahOptionsUsageAtom : public
    oahPureHelpAtomWithoutAValue
3 oah/oahAtomsCollection.h:class EXP oahHelpAtom : public oahPureHelpAtomWithoutAValue
4 oah/oahAtomsCollection.h:class EXP oahHelpSummaryAtom : public
    oahPureHelpAtomWithoutAValue
5 oah/oahAtomsCollection.h:class EXP oahAboutAtom : public oahPureHelpAtomWithoutAValue
6 oah/oahAtomsCollection.h:class EXP oahVersionAtom : public oahPureHelpAtomWithoutAValue
7 oah/oahAtomsCollection.h:class EXP oahLibraryVersionAtom : public
    oahPureHelpAtomWithoutAValue
8 oah/oahAtomsCollection.h:class EXP oahHistoryAtom : public oahPureHelpAtomWithoutAValue
9 oah/oahAtomsCollection.h:class EXP oahLibraryHistoryAtom : public
    oahPureHelpAtomWithoutAValue
10 oah/oahAtomsCollection.h:class EXP oahContactAtom : public oahPureHelpAtomWithoutAValue

```

### 16.5.2 Pure help atoms expecting a value

The base `oahPureHelpAtomExpectingAValue` class is defined in `src/oah/oahBasicTypes.h/.cpp`:

```

1 class EXP oahPureHelpAtomExpectingAValue : public oahAtomExpectingAValue
2 {
3     // ... ..
4
5     protected:
6
7         // protected fields
8         // -----
9
10        std::string                fHelpAtomExpectingAValueServiceName; // JMI ???
11 };

```

The actual pure help atoms expecting a value are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grh 'public
   oahPureHelpAtomExpectingAValue'
2 oah/oahAtomsCollection.h:class EXP oahQueryOptionNameAtom : public
   oahPureHelpAtomExpectingAValue
3 oah/oahAtomsCollection.h:class EXP oahFindStringAtom : public
   oahPureHelpAtomExpectingAValue
4 formats/msdl/msdlInputOah.h:class EXP oahDisplayMsdlKeywordsInLanguageAtom : public
   oahPureHelpAtomExpectingAValue
5 formats/msdl/msdlInputOah.h:class EXP oahDisplayMsdlTokensInLanguageAtom : public
   oahPureHelpAtomExpectingAValue

```

## 16.6 Options implicitly storing a value

There are options in multiGeneration to select the generated output :

```

1 jacquesmenu@macmini > msdlconverter --help-generate-output
2 --- Help for subgroup "Generated output" in group "Generated output group" ---
3 Generated output group (-help-generate-output-group, -hgc-group):
4 -----
5 Generated output (-help-generate-output, -hgo):
6     -guido
7         Generate Guido code as output.
8     -lilypond
9         Generate LilyPond code as output.
10    -braille
11        Generate braille code as output.
12    -musicxml
13        Generate MusicXML code as output.

```

No value is supplied in the command line or in a type `oahOptionsVector`, but a variable is used to store a value alright.

Purely virtual class `oahAtomImplicitlyStoringAValue` is the base class for this:

```

1 class EXP oahAtomImplicitlyStoringAValue : public oahAtom
2 /*
3     a purely virtual common ancestor for all atom classes
4     that store a value in a variable
5     without taking it from argv or an oahOptionsVector
6 */
7 {
8     // ... ..
9
10    protected:
11
12        // protected fields
13        // -----
14

```

```

15     std::string          fVariableName;
16     Bool                fSetByAnOption;
17 };

```

This used by class `mfMultiGenerationOutputKindAtom` defined in `src/formatsgeneration/multiGeneration/multiGenerationOah.h/.cpp`:

```

1 class EXP mfMultiGenerationOutputKindAtom : public oahAtomImplicitlyStoringAValue
2 {
3     // ... ..
4
5     private:
6
7     // private fields
8     // -----
9
10    mfMultiGenerationOutputKind&
11        fMultiGenerationOutputKindVariable;
12 };

```

The value is stored in the variable in constructor `mfMultiGenerationOutputKindAtom::mfMultiGenerationOutputKindAtom ()`:

```

1 mfMultiGenerationOutputKindAtom::mfMultiGenerationOutputKindAtom (
2     const std::string&      longName,
3     const std::string&      shortName,
4     const std::string&      description,
5     const std::string&      variableName,
6     mfMultiGenerationOutputKind& mfMultiGenerationOutputKindVariable)
7 : oahAtomImplicitlyStoringAValue (
8     longName,
9     shortName,
10    description,
11    variableName,
12    oahElementValueKind::kElementValueWithout),
13    fMultiGenerationOutputKindVariable ( // this is where the value is supplied
14        mfMultiGenerationOutputKindVariable)
15 {}

```

## 16.7 Options and help handling

- each option short name and non-empty long name must be unique in a given handler, to avoid ambiguities;
- an service `main ()` calls method `oahHandler::handleOptionsAndArgumentsFromArgcArgv ()`, in which:
  - method `oahHandler::handleOptionNameCommon ()` handles the option names;
  - `handleOptionValueOrArgument()` and the arguments to the service.
- contracted forms are expanded in method `oahHandler::handleOptionNameCommon ()` before the resulting, uncontracted options are handled;
- options handling works in two passes:
  - the first one creates a list of class `oahElementUse` instances from `argc/argv` or an options and arguments;
  - the second one traverses this list to apply the options that are used.

- the options are applied by virtual method `applyElement ()`, virtual method `applyAtomWithValue ()` and virtual method `applyAtomWithDefaultValue ()`;
- method `oahHandler::handleKnownArgvAtom ()` associates the value to the (preceding) field `oahHandler::fHandlerArgumentsVector` if not null, or appends it to field `oahHandler::fHandlerArgumentsVector` to otherwise;
- `fPendingArgvAtomExpectingAValue` is used in `argv` contents handling to associate an option name with its value, which is the next element in `argv`.

## 16.8 Basic OAH types

They are defined in `src/oah/oahBasicTypes.h/.cpp`. The classes are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class    oahBasicTypes.h
2 // PRE-declarations for class mutual dependencies
3 class    oahAtom;
4 class    oahSubGroup;
5 class    oahGroup;
6 class    oahHandler;
7 enum class oahOptionsDefaultValuesStyle {
8 enum class oahHandlerUsedThruKind {
9 class    oahPrefix;
10 class EXP oahPrefix : public smartable
11     a common ancestor for all atom classes,
12     this class    contains only an uplink to the containing subgroup
13 class EXP oahAtom : public oahElement
14 /* this class is purely virtual
15 class EXP oahAtomExpectingAValue : public oahAtom
16     a common ancestor for all atom classes
17 /* this class is purely virtual
18     a common ancestor for all atom classes
19 class EXP oahAtomStoringAValue : public oahAtomExpectingAValue
20 /* this class is purely virtual
21 class EXP oahPureHelpAtomWithoutAValue : public oahAtom
22 /* this class is purely virtual
23 class EXP oahPureHelpAtomExpectingAValue : public oahAtomExpectingAValue
24 /* this class is purely virtual
25 class EXP oahSubGroup : public oahElement
26 class EXP oahGroup : public oahElement
27 class EXP oahHandler : public smartable
28 /* this class is purely virtual
29 enum class oahOptionalValuesStyleKind {
30 class EXP oahAtomWithoutAValue : public oahAtom
31 / * this class is purely virtual

```

## 16.9 Prefixes handling

### 16.10 argc/argv versus oahOptionsVector

Passing the options and arguments over to the library when using MusicFormats can be done in two ways:

- command line tools get them from `argc/argv` as usual;
- application using the library through the API should place them in an `oahOptionsVector`, defined in `src/mflibrarymfMusicformatsErrors.h`:

```

1 typedef std::vector<std::pair<std::string, std::string> > oahOptionsVector;

```

Using an `oahOptionsVector` can be done for example:

- in Web sites;
- in the generators CLI tools found in the `src/clisamples/` folder `src/clisamples/xml2Any.cpp`, `src/clisamples/libMultipleInitsTest.cpp`, `src/clisamples/Mikrokosmos3Wandering.cpp` and `src/clisamples/LilyPondIssue34.cpp`, as well as in `src/clisamples/msdl.cpp`, the MSDL converter command line interface.

In these tools, and `oahOptionsVector` is instantiated and populated from `argc/argv` with `convertArgcArgvToOptions` defined in `src/oah/oahBasicTypes.h`:

```
1 EXP Bool convertArgcArgvToOptionsAndArguments (
2     int          argc,
3     char         *argv[],
4     oahOptionsVector& theOptionsVector)
```

class `oahHandler` in `src/oah/oahBasicTypes.h/.cpp` contains:

```
1 // options and arguments handling
2 oahElementHelpOnlyKind
3         handleOptionsFromOptionsAndArguments (
4             std::string      serviceName,
5             const oahOptionsVector& theOptionsVector);
6
7 oahElementHelpOnlyKind
8         handleOptionsAndArgumentsFromArgcArgv (
9             int      argc,
10            char*    argv[]);
11
12 virtual void      checkOptionsAndArgumentsConsistency ();
13
14 virtual void      checkOptionsAndArguments () const = 0;
```

## 16.11 Applying options

Each `oahElement`, defined in `src/oah/oahElements.h/.cpp`, has an `applyElement` method:

```
1 virtual void      applyElement (std::ostream& os) = 0;
```

Atoms that can have an associated value are described in `src/oah/oahBasicTypes.h/.cpp` by class `oahAtomExpectingValue` which has methods `applyAtomWithValue` and `applyAtomWithDefaultValue`:

```
1 virtual void      applyAtomWithValue (
2     const std::string& theString,
3     std::ostream& os) = 0;
4
5 virtual void      applyAtomWithDefaultValue (std::ostream& os);
6 // used only if fElementValueKind
7 // is oahElementValueKind::kElementValueImplicit
8 // or oahElementValueKind::kElementValueOptional
```

There are two methods for that:

```
1 void      applyElement (std::ostream& os) override; %%%JMI
```

The last option is checked by method `oahHandler::checkMissingPendingArgvAtomExpectingAValueValue ()` in `src/oah/oahBasicTypes.cpp`.

## 16.12 A OAH atoms collection

Frequent OAH atoms have been grouped in `src/oah/oahAtomsCollection.h/.cpp`. They are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class oahAtomsCollection.h
2 class EXP oahAtomAlias : public oahAtom
3 class EXP oahMacroAtom : public oahAtom
4 class EXP oahOptionsUsageAtom : public oahPureHelpAtomWithoutAValue
5 class EXP oahHelpAtom : public oahPureHelpAtomWithoutAValue
6 class EXP oahHelpSummaryAtom : public oahPureHelpAtomWithoutAValue
7 class EXP oahAboutAtom : public oahPureHelpAtomWithoutAValue
8 class EXP oahVersionAtom : public oahPureHelpAtomWithoutAValue
9 class EXP oahContactAtom : public oahPureHelpAtomWithoutAValue
10 class EXP oahBooleanAtom : public oahAtom
11 class EXP oahTwoBooleansAtom : public oahBooleanAtom
12 class EXP oahThreeBooleansAtom : public oahBooleanAtom
13 class EXP oahCombinedBooleansAtom : public oahAtom
14 class EXP oahCommonPrefixBooleansAtom : public oahAtom
15 class EXP oahIntegerAtom : public oahAtomStoringAValue
16 class EXP oahTwoIntegersAtom : public oahIntegerAtom
17 class EXP oahFloatAtom : public oahAtomStoringAValue
18 class EXP oahStringAtom : public oahAtomStoringAValue
19 class EXP oahFactorizedStringAtom : public oahAtom
20 class EXP oahStringWithDefaultValueAtom : public oahStringAtom
21 class EXP oahRationalAtom : public oahAtomStoringAValue
22 class EXP oahNaturalNumbersSetElementAtom : public oahAtomStoringAValue
23 class EXP oahColorRGBAtom : public oahAtomStoringAValue
24 class EXP oahIntSetElementAtom : public oahAtomStoringAValue
25 class EXP oahStringSetElementAtom : public oahAtomStoringAValue
26 class EXP oahStringToIntMapElementAtom : public oahAtomStoringAValue
27 class EXP oahStringAndIntegerAtom : public oahAtomStoringAValue
28 class EXP oahStringAndTwoIntegersAtom : public oahAtomStoringAValue
29 class EXP oahLengthUnitKindAtom : public oahAtomStoringAValue
30 class EXP oahLengthAtom : public oahAtomStoringAValue
31 class EXP oahMidiTempoAtom : public oahAtomStoringAValue
32 class EXP oahOptionNameHelpAtom : public oahStringWithDefaultValueAtom
33 class EXP oahQueryOptionNameAtom : public oahPureHelpAtomExpectingAValue
34 class EXP oahFindStringAtom : public oahPureHelpAtomExpectingAValue

```

See chapter 39 [The OAH atoms collection], page 266, for more details.

## 16.13 An option and help example

Option `-beam-all-grace-notes` controls whether beams should be added to grace notes. Here is how it is implemented and used.

First, we must determine to which internal representation or conversion pass it is applied to. In this case, that is the conversion pass of an MXSR to MSR. Thus we have in `src/passes/mxsr2msr/mxsr2msrOah.h`:

```

1 class EXP mxsr2msrOahGroup : public oahGroup
2
3     Bool                fBeamAllGraceNotes;
4
5     Bool                getBeamAllGraceNotes () const
6                         { return fBeamAllGraceNotes; }

```

In `src/passes/mxsr2msr/mxsr2msrOah.cpp`, the option is created this way:

```

1 void mxsr2msrOahGroup::initializeNotesOptions ()
2
3 // beam all grace notes
4 // -----
5
6 fBeamAllGraceNotes = false;
7
8 S_oahBooleanAtom
9   beamAllGraceNotesAtom =
10     oahBooleanAtom::create (
11       "beamagn", "beam-all-grace-notes",
12 R"(Add a beam to all grace notes)",
13       "beamAllGraceNotes",
14       fBeamAllGraceNotes);
15   subGroup->
16     appendAtomToSubGroup (
17       beamAllGraceNotesAtom);

```

And that's it.

The option value is checked in `src/passes/mxsr2msr/mxsr2msrTranslator.cpp.h/.cpp`:

```

1 void mxsr2msrTranslator::visitStart ( S_grace& elt )
2
3 // should all grace notes be beamed?
4 if (gGlobalMxsr2msrOahGroup->getBeamAllGraceNotes ()) {
5   fCurrentGraceIsBeamed = true;
6 }
7
8 void mxsr2msrTranslator::handleStandaloneOrDoubleTremoloNoteOrGraceNoteOrRest (
9   const S_msrNote& newNote)
10
11 // create grace notes group
12 fPendingGraceNotesGroup =
13   msrGraceNotesGroup::create (
14     inputLineNumber,
15     msrGraceNotesGroupKind::kGraceNotesGroupBefore, // default value
16     fCurrentGraceIsSlashed,
17     fCurrentGraceIsBeamed,
18     fCurrentMeasureNumber);

```

## 16.14 Options and help introspection

OAH represents options and the associated help in a tree of groups containing subgroups containing atoms. Searching it is easy, and there are options `-query` and `-find` for that.

Option `'-query'` provides informations about an option name:

```

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

```

Option `-find` searches the OAH tree ignoring letter cases:

```

1 jacquesmenu@macmini > xml2ly -find grace
2 5 occurrences of std::string "grace" have been found:
3   1:
4     -hgraces-group, -help-grace-notes-group
5
6   2:
7     -hgraces, -help-grace-notes

```

```

8
9   3:
10    -slashagn, -slash-all-grace-notes
11    Add a slash to all grace notes
12   4:
13    -sluragn, -slur-all-grace-notes
14    Add a slur to all grace notes
15   5:
16    -beamagn, -beam-all-grace-notes
17    Add a beam to all grace notes

```

These two options are implemented as `oahQueryOptionNameAtom` and `oahFindStringAtom` respectively in `src/oah/oahAtomsCollection.h/.cpp`.

## 16.15 Insider versus regular handlers

MusicFormats features two 'views' of the options and help available:

- the full view, named 'insider', contains the options grouped by format or pass;
- the default user view, named 'regular', contains options grouped by topic, such as tuplets or MIDI.

The 'regular' views have been introduced because there are many options and it was cumbersome to browse them without a user-oriented view by topics.

class `oahRegularHandler` relies on the corresponding insider handler:

```

1  protected:
2
3      // protected fields
4      // -----
5
6      S_oahHandler          fInsiderHandler;

```

A regular handler merely selects options from the `fInsiderHandler`, and presents them in groups and subgroups its own way. The group names are hidden to the user, and only the subgroups are seen in the help provided to the user.

For example, in `src/clisamples/xml2xml/xml2xmlRegularHandler.cpp`, there is:

```

1  void xml2xmlRegularHandler::createInformationsRegularGroup ()
2  {
3      // group
4
5      S_oahGroup
6      group =
7          oahGroup::create (
8              "Informations group",
9              "help-informations-group", "hinfos-group",
10             "",
11             oahElementVisibilityKind::kElementVisibilityWhole);
12  appendGroupToRegularHandler (group);
13
14  // subgroup
15
16  S_oahSubGroup
17  subGroup =
18      oahSubGroup::create (
19          "Informations",
20          "help-informations", "hinfos",
21          "",

```



```

22     oahElementVisibilityKind::kElementVisibilityWhole,
23     group);
24 group->
25     appendSubGroupToGroup (subGroup);
26
27 // atoms from the insider handler
28
29 registerAtomInRegularSubgroup ("about", subGroup);
30 registerAtomInRegularSubgroup ("version", subGroup);
31 registerAtomInRegularSubgroup ("version-full", subGroup);
32 registerAtomInRegularSubgroup ("history", subGroup);
33 registerAtomInRegularSubgroup ("mf-version", subGroup);
34 registerAtomInRegularSubgroup ("mf-history", subGroup);
35 registerAtomInRegularSubgroup ("contact", subGroup);
36 registerAtomInRegularSubgroup ("display-prefixes", subGroup);
37 registerAtomInRegularSubgroup ("display-single-character-options", subGroup);
38
39 registerAtomInRegularSubgroup ("display-cpu-usage", subGroup);
40 }

```

An insider handler is always created, and a regular one relying on it is created if relevant. Here is how this is done this way, here in `src/clisamples/msdl.cpp`:

```

1 // create an msdlConverter insider OAH handler
2 // -----
3
4 const S_msdlConverterInsiderHandler&
5     insiderOahHandler =
6     msdlConverterInsiderHandler::create (
7         serviceName,
8         serviceName + " insider OAH handler with argc/argv",
9         multiGenerationOutputKind);
10
11 // the OAH handler to be used, a regular handler is the default
12 // -----
13
14 if (insiderOption) {
15     // use the insider msdlConverter OAH handler
16     handler = insiderOahHandler;
17 }
18 else {
19     // create a regular msdlConverter OAH handler
20     handler =
21     msdlConverterRegularHandler::create (
22         serviceName,
23         serviceName + " regular OAH handler with argc/argv",
24         insiderOahHandler,
25         multiGenerationOutputKind);
26 }

```

## 16.16 Deciphering the options and arguments

### 16.16.1 Options and arguments multi-pass analysis

The options and arguments are first placed in a `mfOptionsAndArguments` instance:

- the command line services do this with interface function `convertArgcArgvToOptionsAndArguments` () in their function, for example in `src/clisamples/Mikrokosmos3Wandering.cpp`:

```

1 int main (int argc, char* argv[])
2 //-----
3 {
4     // ... ..
5
6     // create the global run data
7     // -----
8
9     gGlobalServiceRunData =
10     mfServiceRunData::create (serviceName);
11
12     // ... ..
13 }

```

- the API functions receive an `mfOptionsAndArguments` as an argument, here in `src/converters/musicxml2musicxml`

```

1 EXP mfMusicformatsErrorKind musicxmlfile2musicxml (
2     const char*          fileName,
3     mfOptionsAndArguments& handlerOptionsAndArguments,
4     std::ostream&         out,
5     std::ostream&         err)
6 {
7     SXMLFile
8     sxmlfile =
9     createSXMLFileFromFile (
10         fileName,
11         "Pass 1",
12         "Create an MXSR reading a MusicXML file");
13
14     if (sxmlfile) {
15         return
16         xmlFile2musicxmlWithOptionsAndArguments (
17             sxmlfile,
18             handlerOptionsAndArguments,
19             out,
20             err);
21     }
22
23     return mfMusicformatsErrorKind::kMusicformatsErrorInvalidFile;
24 }

```

This is done using a two-pass scheme:

- first, a list of the options uses is built;
- then, the options and their arguments, if any, in this list are applied.

class `oahHandler` contains:

```

1 // elements uses
2 std::list<S_oahElementUse> fElementUsesList;
3
4 // atoms waiting for a value
5 S_oahAtomExpectingAValue
6     fPendingArgvAtomExpectingAValue;
7 std::string         fNameUsedForPendingArgvAtomExpectingAValue;

```

### 16.16.2 Pure help runs

A pure help run is one in which MusicFormats in which help, without any other option. In such a case the run quit silently, otherwise it proceeds to performs its task. The type describing that is enumeration type `oahElementHelpOnlyKind`, defined in `src/oah/oahElements.h`:

```
1 enum class oahElementHelpOnlyKind {
2     kElementHelpOnlyYes,
3     kElementHelpOnlyNo
4 };
```

### 16.16.3 Applying options

The options are applied in `src/oah/oahBasicTypes.cpp` by method `oahHandler::applyOptionsFromElementUsesList` defined in `src/oah/oahBasicTypes.h/.cpp`:

```
1 oahElementHelpOnlyKind oahHandler::applyOptionsFromElementUsesList ()
```

The heart of it is:

```
1 oahElementHelpOnlyKind oahHandler::applyOptionsFromElementUsesList ()
2 {
3     // ... ..
4
5     // the heart of it
6     if (
7         // group?
8         S_oahGroup
9         group =
10            dynamic_cast<oahGroup*>(&(*elementUsed))
11     ) {
12         group->
13             applyElement (
14                 gOutputStream);
15     }
16
17     else if (
18         // subgroup?
19         S_oahSubGroup
20         subGroup =
21            dynamic_cast<oahSubGroup*>(&(*elementUsed))
22     ) {
23         subGroup->
24             applyElement (
25                 gOutputStream);
26     }
27
28     else {
29         // this is an atom
30
31         S_oahAtom
32         atom =
33            dynamic_cast<oahAtom*>(&(*elementUsed));
34
35         oahElementValueKind
36         atomValueKind =
37             atom->
38                 getElementValueKind ();
39
40         if (
41             // atom expecting a value?
42             S_oahAtomExpectingAValue
43             atomExpectingAValue =
```

```

44         dynamic_cast<oahAtomExpectingAValue*>(&(*elementUsed))
45     ) {
46         switch (atomValueKind) {
47             case oahElementValueKind::kElementValueWithout:
48                 {
49                     std::stringstream s;
50
51                     s <<
52                         "Atom with value " <<
53                         atomExpectingAValue->fetchNamesBetweenQuotes () <<
54                         " has been registered as without value";
55
56                     oahInternalError (s.str ());
57                 }
58                 break;
59
60             case oahElementValueKind::kElementValueImplicit:
61                 atomExpectingAValue->
62                     applyAtomWithDefaultValue (
63                         gOutputStream);
64                 break;
65
66             case oahElementValueKind::kElementValueMandatory:
67                 if (valueUsed.size ()) {
68                     atomExpectingAValue->
69                         applyAtomWithValue (
70                             valueUsed,
71                             gOutputStream);
72                 }
73                 else {
74                     std::stringstream s;
75
76                     s <<
77                         "Atom expecting a value " <<
78                         atomExpectingAValue->fetchNamesBetweenQuotes () <<
79                         " needs a non-empty value";
80
81                     oahInternalError (s.str ());
82                 }
83                 break;
84
85             case oahElementValueKind::kElementValueOptional:
86                 if (valueUsed.size ()) {
87                     atomExpectingAValue->
88                         applyAtomWithValue (
89                             valueUsed,
90                             gOutputStream);
91                 }
92                 else {
93                     atomExpectingAValue->
94                         applyAtomWithDefaultValue (
95                             gOutputStream);
96                 }
97                 break;
98             } // switch
99     }
100
101     else {
102 #ifdef TRACING_IS_ENABLED
103         if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
104             gLogStream <<
105                 "**** Handling atom not expecting a value:" <<
106                 std::endl;
107
108             ++gIndenter;
109
110             gLogStream <<

```

```

111         atom <<
112         std::endl;
113
114         --gIndenter;
115     }
116 #endif
117
118     atom->
119     applyElement (
120         gOutputStream);
121 }
122 }
123
124 // has a help-only been applied?
125 switch (elementUsed->getElementHelpOnlyKind ()) {
126     case oahElementHelpOnlyKind::kElementHelpOnlyYes:
127         // a help option has been applied
128         this->
129         setOahHandlerFoundAHelpOption (
130             elementUsed->
131             fetchNamesBetweenQuotes ());
132         break;
133     case oahElementHelpOnlyKind::kElementHelpOnlyNo:
134         break;
135 } // switch
136 }
137
138 else {
139     std::stringstream s;
140
141     s <<
142     "Element from the from the atom uses list for \"" <<
143     nameUsed <<
144     "\" is null";
145
146     oahInternalError (s.str ());
147
148     // ... ..
149 }

```

#### 16.16.4 Early handling of some options

Debugging OAH needs the trace handling tracing options option `-trace-oah`, `-toah` and option `-trace-oah-detail`. `-toahd` to be activated first, even if they are not the first ones supplied.

The same holds to handle the option `-insider` option, since it involves using the insider handler and not a regular one.

Also, the `-quiet`, `-q` option should be handled early, to avoid errors in the options being reported.

Another case is the option `-trace-components`, `-tcomps` option. The versions data should exist when the option `-version`, `-v` is applied in method `oahVersionAtom::applyElement ()` in `src/oah/oahAtomsCollection.c` but building them should be able to produce a trace if this option is selected. This circularity circularity should thus be broken. Version history handling is presented in chapter ?? [musicformats components], page ??.

The early options names are declared in `src/oah/oahEarlyOptions.h`:

```

1 // ... ..
2
3 // insider
4 EXP extern const std::string K_INSIDER_OPTION_LONG_NAME;
5 EXP extern const std::string K_INSIDER_OPTION_SHORT_NAME;
6
7 // ... ..
8
9 class EXP oahEarlyOptions
10 {
11     // ... ..
12
13     public:
14
15         // set and get
16         // -----
17
18         void                setEarlyInsiderOption ();
19         Bool                getEarlyInsiderOption () const
20                             { return fEarlyInsiderOption; }
21
22         // ... ..
23
24     private:
25
26         // fields
27         // -----
28
29         Bool                fEarlyInsiderOption;
30
31         // ... ..
32 }

```

Then, in `src/oah/oahEarlyOptions.cpp`, there is:

```

1 // ... ..
2
3 const std::string K_INSIDER_OPTION_LONG_NAME = "insider";
4 const std::string K_INSIDER_OPTION_SHORT_NAME = "ins";
5
6 void oahEarlyOptions::setEarlyInsiderOption ()
7 {
8     if (fTraceEarlyOptions) {
9         gLogStream <<
10             "Setting fEarlyInsiderOption" <<
11             std::endl;
12     }
13
14     fEarlyInsiderOption = true;
15 }
16
17 // ... ..

```

Method `oahEarlyOptions::applyEarlyOptionIfRelevant ()` performs the analysis:

```

1 void oahEarlyOptions::applyEarlyOptionIfRelevant (
2     const std::string& argumentWithoutDashToBeUsed,
3     const std::string& optionValue)
4 {
5     // this is OAH handling pass 1
6     if (
7         isEarlyOptionRecognized (
8             argumentWithoutDashToBeUsed, K_INSIDER_OPTION_LONG_NAME)
9         ||
10        isEarlyOptionRecognized (

```

```

11     argumentWithoutDashToBeUsed, K_INSIDER_OPTION_SHORT_NAME)
12 ) {
13     setEarlyInsiderOption ();
14 }
15
16 // ... ..
17 }

```

## 16.17 Implementing the -find option

Class `oahFindableElement` in `src/oah/oahElements.h/.cpp` is the base class of all those that can be found with this introspection option:

```

1 class EXP oahFindableElement : public smartable
2 {
3     public:
4
5         // creation from MusicXML
6         // -----
7
8         /* this class is purely virtual
9          * static SMARTP<oahFindableElement> create ();
10        */
11
12        // ... ..
13
14        public:
15
16            // public services
17            // -----
18
19            virtual Bool      findStringInFindableElement (
20                const std::string&      lowerCaseString,
21                std::list<S_oahFindStringMatch>& foundMatchesList,
22                std::ostream&            os) const = 0;
23
24        public:
25
26            // print
27            // -----
28
29            virtual std::string  asString () const = 0;
30
31            virtual void          print (std::ostream& os) const = 0;
32
33            virtual const std::string  containingFindableElementAsString () const = 0;
34
35        private:
36
37            // private fields
38            // -----
39 };

```

When matches are found, there are stored in a list of instances of `oahFindStringMatch`:

```

1 class oahFindStringMatch : public smartable
2 {
3     // ... ..
4
5     protected:
6
7         // protected fields
8         // -----

```

```

9
10     std::string          fFoundString;
11     std::string          fContainingFindableElementInfo;
12 };

```

The `fContainingFindableElementInfo` describes the OAH element that the `std::string` was found in, either in its option name(s) or in its description.

For example, method `oahHandler::findStringInFindableElement ()` in `src/oah/oahElement.cpp` creates an instance and appends it to the list :

```

1 Bool oahHandler::findStringInFindableElement (
2     const std::string&      lowerCaseString,
3     std::list<S_oahFindStringMatch>& foundMatchesList,
4     std::ostream&           os) const
5 {
6     Bool result;
7
8     // .. .. .
9
10    // does this handler's header match?
11    Bool headerMatches =
12        mfStringToLowerCase (fHandlerHeader).find (lowerCaseString) != std::string::npos;
13
14    // does this handler's description match?
15    Bool descriptionMatches =
16        mfStringToLowerCase (fHandlerDescription).find (lowerCaseString) != std::string::npos;
17
18    // does this handler's usage match?
19    Bool usageMatches =
20        mfStringToLowerCase (fHandlerUsage).find (lowerCaseString) != std::string::npos;
21
22    if (headerMatches || descriptionMatches || usageMatches) {
23        std::stringstream s;
24
25        s <<
26            fHandlerHeader <<
27            ' ' <<
28            fHandlerDescription <<
29            ' ' <<
30            fHandlerUsage;
31
32        // append the match to foundStringsList
33        foundMatchesList.push_back (
34            oahFindStringMatch::create (
35                s.str (),
36                containingFindableElementAsString ());
37
38        result = true;
39    }
40
41    // do this handler's prefixes match?
42    if (fHandlerPrefixesMap.size ()) {
43        ++gIndenter;
44
45        for (
46            std::map<std::string, S_oahPrefix>::const_iterator i =
47                fHandlerPrefixesMap.begin ();
48            i != fHandlerPrefixesMap.end ();
49            ++i
50        ) {
51            S_oahPrefix
52                prefix = (*i).second;
53
54            // does the prefix match?
55            prefix->

```



```

56         findStringInFindableElement (
57             lowerCaseString,
58             foundMatchesList,
59             os);
60     } // for
61
62     --gIndenter;
63 }
64
65 // do this handler's groups match?
66 if (fHandlerGroupsList.size ()) {
67     ++gIndenter;
68
69     for (S_oahGroup group : fHandlerGroupsList) {
70         group->
71             findStringInGroup (
72                 lowerCaseString,
73                 foundMatchesList,
74                 os);
75     } // for
76
77     --gIndenter;
78 }
79
80 return result;
81 }

```

The same holds for method `oahPrefix::findStringInFindableElement ()` in `src/oah/oahElements.cpp`.

## 16.18 Checking options consistency

The options groups have a `checkGroupOptionsConsistency ()` to check that the use of the options if this group are consistent.

For example:

```

1 void mxsr2msrOahGroup::checkGroupOptionsConsistency ()
2 {
3 }

```

## 16.19 Adding new options

In order to make a new option available, one should:

- choose a short name and possibly a long name for the option;
- choose an atom class in `src/oah/oahAtomsCollection.h/.cpp` or write a new one if needed;
- decide in which subgroup and group the option should be made available in an insider OAH group, and create the latter two if needed;
- create a suitable atom and append it to the desired OAH subgroup;
- check the use of the atom wherever needed in the code base;
- add the new atom's long name to the corresponding regular OAH group;
- and last but not least... test the result.

This should be done first with the `-insider`, `-ins` option, and then without it to ensure that the regular OAH handler knows the new option too.

### 16.19.1 Representations' vs. passes' options

When adding a new option, it is sometimes not clear whether to assign it to a format or to the passes that create or browse it.

For example, the tracing of <backup/> and <forward/> is used by both `mxsr2msr0ah` and `msr2mxsr0ah`. The corresponding options are thus placed in `src/formats/mxsr/mxsr0ah.h/.cpp`:

### 16.19.2 Using an existing OAH atom class

When option `-reverse-names-display-order`, `-rndo` was added to OAH by this author:

- class `oahBooleanAtom` was ready to be used;
- it was decided to place it in the global variable `gGlobalOahOahGroup` OAH group, in its `Options help` sub group;
- class `oahOahGroup` in `src/oah/oahOah.h/.cpp` got a new `fReverseNamesDisplayOrder` field:

```

1 class EXP oahOahGroup : public oahGroup
2 {
3     void                setReverseNamesDisplayOrder ()
4                         { fReverseNamesDisplayOrder = true; }
5     Bool                getReverseNamesDisplayOrder () const
6                         { return fReverseNamesDisplayOrder; }
7
8     // ... ..
9
10    Bool                fReverseNamesDisplayOrder;
11
12    // ... ..
13 };

```

- method `oahOahGroup::initializeOahBasicHelpOptions ()` was augmented with:

```

1 void oahOahGroup::initializeOahBasicHelpOptions (
2     std::string serviceName)
3 {
4     // ... ..
5
6     // reverse names display order
7
8     fReverseNamesDisplayOrder = false;
9
10    subGroup->
11        appendAtomToSubGroup (
12            oahBooleanAtom::create (
13                "rndo", "reverse-names-display-order",
14                R"(Write the short names before the long ones.)",
15                "reverseNamesDisplayOrder",
16                fReverseNamesDisplayOrder));
17
18    // ... ..
19 }

```

- method `oahOahGroup::printOahOahValues ()` was augmented with:

```

1 void oahOahGroup::printOahOahValues (int valueFieldWidth)
2 {
3     gLogStream <<
4         "The basic options are:" <<
5         std::endl;
6
7     // ... ..
8
9     // options and help display
10    // -----
11
12    gLogStream << std::left <<
13        std::setw (valueFieldWidth) << "Options trace and display:" <<
14        std::endl;
15
16    ++gIndenter;
17
18    gLogStream << std::left <<
19        std::setw (valueFieldWidth) << "fReverseNamesDisplayOrder" << " : " <<
20        fReverseNamesDisplayOrder <<
21        std::endl <<
22
23    // ... ..

```

- then tests of the use of option `-reverse-names-display-order`, `-rndo` were added in `src/oah/oahElements.c` such as in method `oahElement::fetchNames ()`:

```

1 std::string oahElement::fetchNames () const
2 {
3     std::stringstream s;
4
5     if (
6         fShortName.size ()
7         &&
8         fLongName.size ()
9     ) {
10        if (gGlobalOahOahGroup->getReverseNamesDisplayOrder ()) {
11            s <<
12                '-' << fShortName <<
13                ", " <<
14                '-' << fLongName;
15        }
16        else {
17            s <<
18                '-' << fLongName <<
19                ", " <<
20                '-' << fShortName;
21        }
22    }
23
24    else {
25        if (fShortName.size ()) {
26            s <<
27                '-' << fShortName;
28        }
29        if (fLongName.size ()) {
30            s <<
31                '-' << fLongName;
32        }
33    }
34
35    return s.str ();
36 }

```

- and finally, all `*RegularHandler::createOahRegularGroup ()` methods were augmented with:

```

1 void msdl2brailleRegularHandler::createOahRegularGroup ()
2 {
3     // ... ..
4
5     registerAtomInRegularSubgroup ("reverse-names-display-order", subGroup);
6
7     // ... ..
8 }

```

### 16.19.3 Creating a new OAH atom class without a value

When class `oahHistoryAtom` was added to OAH, the first thing has been to add a `printHistory ()` in class `mfcMultiComponent` in `src/mfutilities/mfcBasicTypes.h`:

```

1 class mfcMultiComponent : public smartable
2 {
3     // ... ..
4
5     public:
6
7         // print
8         // -----
9
10        void                print (std::ostream& os) const;
11        void                printHistory (std::ostream& os) const;
12
13        // ... ..
14 };

```

Then the next thing has been to clone class `oahVersionAtom` in `src/oah/oahAtomCollection.h/.cpp`, renaming `printVersion ()` to `printHistory ()`:

```

1 //-----
2 class EXP oahHistoryAtom : public oahPureHelpAtomWithoutAValue
3 {
4     // ... ..
5
6     public:
7
8         // public services
9         // -----
10
11        void                applyElement (std::ostream& os) override;
12
13        public:
14
15        // visitors
16        // -----
17
18        void                acceptIn (basevisitor* v) override;
19        void                acceptOut (basevisitor* v) override;
20
21        void                browseData (basevisitor* v) override;
22
23        // print
24        // -----
25
26        void                print (std::ostream& os) const override;
27
28        void                printHistory (std::ostream& os) const;
29 };

```

Then in method `oahHistoryAtom::printHistory ()`, the call to `printVersion ()` has been replaced by a call to `printHistory ()`:

```

1 void oahHistoryAtom::applyElement (std::ostream& os)
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
5             gLogStream <<
6                 "==> option '" << fetchNames () << "' is a oahHistoryAtom" <<
7                 std::endl;
8         }
9     #endif
10
11     int saveIndent = gIndenter.getIndent ();
12
13     gIndenter.resetToZero ();
14
15     printHistory (os);
16
17     gIndenter.setIndent (saveIndent);
18 }

```

method `:: ()` has the be adapted as:

```

1 void oahHistoryAtom::printHistory (std::ostream& os) const
2 {
3     // get the handler history
4     S_mfcMultiComponent
5         handlerMultiComponent =
6             fetchAtomUpLinkToHandler ()->
7             getHandlerMultiComponent ();
8
9     // sanity check
10    mfAssert (
11        __FILE__, __LINE__,
12        handlerMultiComponent != nullptr,
13        "handlerMultiComponent is null");
14
15    handlerMultiComponent->
16        printHistory (os);
17 }

```

Then a new option has been added in method `oahOahGroup::initializeOahBasicHelpOptions ()`, in `src/oah/oahOah.cpp`:

```

1 // history
2
3 subGroup->
4     appendAtomToSubGroup (
5         oahHistoryAtom::create (
6             "hist", "history",
7             regex_replace (
8 R"(Display EXECUTABLE_NAME's history.)",
9                 std::regex ("EXECUTABLE_NAME"),
10                 serviceName),
11                 serviceName));

```

And the new option long name `version` has been added to all regular OAH handlers that already contained the option `-version`, `-v`, such as in method `xml2lyRegularHandler::createInformationsRegularGroup ()`, alongside the existing option `-version`, `-v`:

```

1 registerAtomInRegularSubgroup ("version", subGroup);
2 registerAtomInRegularSubgroup ("version-full", subGroup);
3 registerAtomInRegularSubgroup ("history", subGroup);
4 registerAtomInRegularSubgroup ("mf-version", subGroup);
5 registerAtomInRegularSubgroup ("mf-history", subGroup);

```

### 16.19.4 Creating a new OAH atom class EXPeCting a value

Let's look at how class `oahLengthAtom` is implemented.

class `msrLength` is defined in `src/formats/msr/msrBasicTypes.h/.cpp`:

```

1 class EXP msrLength : public smartable
2 {
3     // ... ..
4
5     // public services
6     // -----
7
8     Bool                operator== (const msrLength& other) const
9     {
10         // JMI convert to same length unit kind before comparing
11
12         return
13             fLengthUnitKind == other.fLengthUnitKind
14             &&
15             fLengthValue == other.fLengthValue;
16     }
17
18     Bool                operator!= (const msrLength& other) const
19     { return ! ((*this) == other); }
20
21     void                convertToLengthUnit (
22         msrLengthUnitKind lengthUnitKind);
23
24     // ... ..
25
26 private:
27     // private fields
28     // -----
29
30     msrLengthUnitKind    fLengthUnitKind;
31     float                fLengthValue;
32 };

```

Enumeration type `msrLengthUnitKind` is defined in `src/formats/msr/msrBasicTypes.h` as:

```

1 enum class msrLengthUnitKind {
2     kUnitInch, kUnitCentimeter, kUnitMillimeter
3 };

```

Here is the declaration of class `oahLengthAtom` in `src/oah/oahAtomsCollection.h`:

```

1 class EXP oahLengthAtom : public oahAtomStoringAValue
2 {
3     /*
4     an atom controlling a length variable
5     */
6
7     // ... ..
8
9     public:
10
11     // set and get
12     // -----
13
14     void                setLengthVariable (
15         msrLength value)
16     {
17         fLengthVariable = value;
18         fSetByAnOption = true;
19     }
20 };

```

```

19         }
20
21     msrLength      getLengthVariable () const
22                     { return fLengthVariable; }
23
24 public:
25
26     // public services
27     // -----
28
29     void            applyAtomWithValue (
30                     const std::string& theString,
31                     std::ostream&      os) override;
32
33     // ... ..
34
35 public:
36
37     // print
38     // -----
39
40     // ... ..
41
42     void            printAtomWithVariableOptionsValues (
43                     std::ostream& os,
44                     int          valueFieldWidth) const override;
45
46 private:
47
48     // private fields
49     // -----
50
51     msrLength&      fLengthVariable;
52 };

```

Method `oahLengthAtom::applyAtomWithValue ()` in `src/oah/oahAtomsCollection.cpp` deciphers the `std::string` supplied by the user and stores it the class `msrLength` variable:

```

1 void oahLengthAtom::applyAtomWithValue (
2     const std::string& theString,
3     std::ostream&      os)
4 {
5     // ... ..
6
7     std::regex e (regularExpression);
8     std::smatch sm;
9
10    regex_match (theString, sm, e);
11
12    unsigned int smSize = sm.size ();
13
14    // ... ..
15
16    if (smSize == 4) {
17        // leave the low level details to the STL...
18        float floatValue;
19        {
20            std::stringstream s;
21            // concatenate the integer and decimal parts
22            s << sm [ 1 ] << sm [ 2 ];
23            s >> floatValue;
24        }
25
26        std::string lengthUnitName = sm [ 3 ];
27
28        // is lengthUnitName known in the length unit names map?

```

```

29     std::map<std::string, msrLengthUnitKind>::const_iterator
30     it =
31         gGlobalMsrLengthUnitKindsMap.find (
32             lengthUnitName);
33
34     if (it == gGlobalMsrLengthUnitKindsMap.end ()) {
35         // no, length unit name is unknown in the map
36
37         std::stringstream s;
38
39         s <<
40             "length unit name \"" << lengthUnitName <<
41             "\" is unknown" <<
42             std::endl <<
43             "The " <<
44             gGlobalMsrLengthUnitKindsMap.size () <<
45             " known length unit names are:" <<
46             std::endl;
47
48         ++gIndenter;
49
50         s <<
51             existingMsrLengthUnitKinds (K_MF_NAMES_LIST_MAX_LENGTH);
52
53         --gIndenter;
54
55         oahError (s.str ());
56     }
57
58     setLengthVariable (
59         msrLength (
60             (*it).second,
61             floatValue));
62 }
63
64 else {
65     std::stringstream s;
66
67     s <<
68         "length value \"" << theString <<
69         "\" for option '" << fetchNames () <<
70         "' is ill-formed";
71
72     oahError (s.str ());
73 }
74 }

```

Method `oahLengthAtom::printAtomWithVariableOptionsValues ()` is in charge of displaying the length value when option `-display-options-valuesdov` is chosen:

```

1 void oahLengthAtom::printAtomWithVariableOptionsValues (
2     std::ostream& os,
3     int valueFieldWidth) const
4 {
5     os << std::left <<
6         std::setw (valueFieldWidth) <<
7         fVariableName <<
8         " : " <<
9         fLengthVariable.asString ();
10    if (fSetByAnOption) {
11        os <<
12            ", set by user";
13    }
14    os << std::endl;
15 }

```



Then an option to set the LilyPond paper height can be added to the relevant OAH options group in method `lpsrOahGroup::initializeLpsrPaperOptions ()` in `src/formats/lpsr//lpsrOah.cpp` by:

```

1 // paper height
2
3 fPaperHeight.setLengthUnitKind (msrLengthUnitKind::kUnitMillimeter);
4 fPaperHeight.setLengthValue (297);
5
6 fPaperHeightAtom =
7   oahLengthAtom::create (
8     "paper-height", "",
9     R"(Set the LilyPond 'paper-height' paper variable to HEIGHT in the LilyPond code.
10    HEIGHT should be a positive floating point or integer number,
11    immediately followed by a unit name, i.e. 'in', 'mm' or 'cm'.
12    By default, LilyPond uses 297 mm (A4 format).)",
13     "HEIGHT",
14     "paperHeight",
15     fPaperHeight);
16 subGroup->
17   appendAtomToSubGroup (
18     fPaperHeightAtom);

```

## 16.20 Extra options

The description of music scores in MusicFormats is quite rich, and it was easy (and tempting...) to offer options such as:

```

1 jacquesmenu@macmini > xml2ly -query show-harmony-analysis
2 --- Help for atom "show-harmony-analysis" in subgroup "Harmony analysis" of group "Extra"
3 ---
4 -sca, -show-harmony-analysis HARMONY_SPEC
5     Write an analysis of the harmony for the given diatonic (semitones) pitch
6     in the current language and the given harmony to standard output.
7     HARMONY_SPEC can be:
8     'ROOT_DIATONIC_PITCH HARMONY_NAME INVERSION'
9     or
10    "ROOT_DIATONIC_PITCH = HARMONY_NAME INVERSION"
11    Using double quotes allows for shell variables substitutions, as in:
12    HARMONY="maj7"
13    INVERSION=2
14    xml2ly -show-harmony-analysis "bes ${HARMONY} ${INVERSION}"

```

This is done in `src/oah/harmoniesExtraOah.h/.cpp`. It suffices to call function `createGlobalHarmoniesExtraOahG`.

```

1 #ifdef EXTRA_OAH_IS_ENABLED
2 // create the extra OAH group
3 appendGroupToHandler (
4   createGlobalHarmoniesExtraOahGroup ());
5 #endif

```

Macro `EXTRA_OAH_IS_ENABLED` is defined or not in `src/oah/enableHarmoniesExtraOahIfDesired.h`:

```

1 // comment the following definition if no extra options are wanted
2
3 #ifndef EXTRA_OAH_IS_ENABLED
4   #define EXTRA_OAH_IS_ENABLED
5 #endif

```

## 16.21 man pages generation

MusicFormats can create man pages for its command line tools by browsing their OAH hierarchy. This has not been finalized yet.

## 16.22 Specific global OAH groups

Some informations need to be available globally in the MusicFormats library, such as the conversion date and command line. They are grouped in `src/oah/generalOah.h/.cpp`:

```

1 class EXP generalOahGroup : public oahGroup
2 {
3     // ... ..
4
5     private:
6
7         // translation date
8         // -----
9
10        std::string          fTranslationDateFull;
11        std::string          fTranslationDateYYYYMMDD;
12
13        // warning and error handling
14        // -----
15
16        Bool                fQuiet;
17
18        Bool                fDontShowErrors;
19        Bool                fDontQuitOnErrors;
20
21        Bool                fDisplaySourceCodePositions;
22
23        // CPU usage
24        // -----
25
26        Bool                fDisplayCPUUsage;
27 };

```

There are also harmonies-specific options grouped in `src/oah/harmoniesExtraOah.h/.cpp`. They are available as icing on the cake icing on the cake independently of any conversion activity:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class  harmoniesExtraOah.h
2 class EXP extraShowAllHarmoniesStructuresAtom : public oahAtomStoringAValue
3 class EXP extraShowAllHarmoniesContentsAtom : public oahAtomStoringAValue
4 class EXP extraShowHarmonyDetailsAtom : public oahAtomStoringAValue
5 class EXP extraShowHarmonyAnalysisAtom : public oahAtomStoringAValue
6 class EXP harmoniesExtraOahGroup : public oahGroup

```

```

1 class EXP harmoniesExtraOahGroup : public oahGroup
2 {
3     // ... ..
4
5     public:
6
7         // fields
8         // -----
9
10        std::string          fHarmoniesRootAsString;
11 };

```

## 16.23 Visiting OAH groups

As an internal representation, OAH can be browsed with the two-phase visitors. This is useful:

- to produce man pages automatically from the options available;
- to create the code that proposes the options to the user in a Web site, also automatically.

---

# Part VI

## Representations

## Chapter 17

# Representations general principles

### 17.1 Trees vs graphs, denormalization

In databases, denormalization means that some data is present in several places. This is usually done for speed, at the cost of making updates more complex, since no such place should be ignored in an update.

A music score can be represented as a tree of elements, but performing conversions of such representations needs shortcuts to be more practical. MusicFormats used the term `link` for that.

### 17.2 Denormalization

#### 17.2.1 Descriptions sharing

MSR uses denormalization explicitly, with smart pointers to class instances being stored in other instances.

In particular, class `msrChord` contains elements that are actually detained by the notes it is composed of:

```
1 // articulations
2 std::list<S_msrArticulation>
3         fChordArticulations;
4
5 // spanners
6 std::list<S_msrSpanner>
7         fChordSpanners;
8
9 // single tremolo
10 S_msrSingleTremolo    fChordSingleTremolo;
```

This is to avoid having to browse the chord's components to obtain the corresponding information each time it is needed.

All such denormalization is done in MSR internally: the code using MSR does not have to denormalize itself. It can use whichever occurrence of any given denormalized data safely, though.

## 17.2.2 Multiple voices

Another aspect to account for is that of dynamics, lyrics, harmonies and figured bass.

LilyPond supplies specific support to have them outside of notes, chords and other sound-producing score elements. This provides flexibility when combining a score's staves and voices in various ways depending on the needs .

MusicFormats has explicit voice kinds for this, declared in `src/formats/msr/msrVoices.h`:

```
1 enum class msrVoiceKind {
2     kVoiceKindRegular,
3     kVoiceKindDynamics,
4     kVoiceKindHarmonies, // for MusicXML <harmony/>, LilyPond ChordNames
5     kVoiceKindFiguredBass // for MusicXML <figured-bass/>, LilyPond FiguredBass
6 };
```

In MSR, for example, a harmony is both attached to a note in a regular voice and an element of a harmony voice:

```
1 std::list<S_msrHarmony>
2     fNoteHarmoniesList;
```

An `msrNote` instance will thus be browsed twice, when those two voices are. The ones attached to a note are browsed this way:

```
1 void msrNote::browseData (basevisitor* v)
2 {
3     // ... ..
4
5     // browse the harmonies if any
6     if (fNoteHarmoniesList.size ()) {
7         ++gIndenter;
8         for (S_msrHarmony harmony : fNoteHarmoniesList) {
9             // browse the harmony
10            msrBrowser<msrHarmony> browser (v);
11            browser.browse (*harmony);
12        } // for
13        --gIndenter;
14    }
15
16    // ... ..
17};
```

## 17.3 Newborn clones

The multi-pass structure of the converters build with `musicformat` leads to a question: should an existing description, such as that of a `barLine` or a `note`, be used as is, or should it be built again?

Depending of the kind of description, both possibilities are used:

- the description is used as is if it is *shallow*, i.e. it doesn't contain smart-pointers to data – it is *self-contained*;
- otherwise, a new description is built, sharing some smart-pointers fieds with the existing one if needed. This newborn clone is then populated when it is inserted in the representation being built.

For example, in `src/passes/msr2lpsr/`, the `S_msrBarLine` values found in the MSR data are used also in the LPSR data:

```

1 void msr2lpsrTranslator::visitStart (S_msrBarLine& elt)
2 {
3     #ifdef TRACING_IS_ENABLED
4         int inputLineNumber =
5             elt->getInputLineNumber ();
6     #endif
7
8     #ifdef TRACING_IS_ENABLED
9         if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
10             gLogStream <<
11                 "--> Start visiting msrBarLine" <<
12                 ", line " << inputLineNumber <<
13                 std::endl;
14         }
15     #endif
16
17     // ... ..
18
19     // append the barLine to the current voice clone
20     fCurrentVoiceClone->
21         appendBarLineToVoice (elt);
22 }

```

On the opposite, a new `S_msrVoice` description is built for use by LPSR: this is how the LilyPond #34 issue is circumvented, adding skip notes where needed in the voices that don't have grace notes at their beginning.

Such new descriptions are created by `*NewbornClone ()` methods, such as:

```

1 S_msrTuplet msrTuplet::createTupletNewbornClone ()
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalTracingOahGroup->getTraceTuplets ()) {
5             gLogStream <<
6                 "Creating a newborn clone of tuplet " <<
7                 asString () <<
8                 std::endl;
9         }
10    #endif
11
12    S_msrTuplet
13        newbornClone =
14        msrTuplet::create (
15            fInputLineNumber,
16            fBarLineUpLinkToMeasure->getMeasureNumber (),
17            fTupletNumber,
18            fTupletBracketKind,
19            fTupletLineShapeKind,
20            fTupletShowNumberKind,
21            fTupletShowTypeKind,
22            fTupletFactor,
23            fMemberNotesSoundingWholeNotes,
24            fMemberNotesDisplayWholeNotes);
25
26    return newbornClone;
27 }

```

Such a newborn clone is created and used this way in method `:: ()`:

```

1 void msr2lpsrTranslator::visitStart (S_msrTuplet& elt)
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5             gLogStream <<
6                 "--> Start visiting msrTuplet" <<

```

```

7      ", line " << elt->getInputLineNumber () <<
8      std::endl;
9  }
10 #endif
11
12 // create the tuplet clone
13 S_msrTuplet
14     tupletClone =
15         elt->createTupletNewbornClone ();
16
17 // register it in this visitor
18 #ifdef TRACING_IS_ENABLED
19     if (gGlobalTracingOahGroup->getTraceTuplets ()) {
20         gLogStream <<
21             "++> pushing tuplet '" <<
22             tupletClone->asString () <<
23             "' to tuplets stack" <<
24             std::endl;
25     }
26 #endif
27
28 fTupletClonesStack.push (tupletClone);
29
30 // is Scheme support needed?
31 switch (elt->getTupletLineShapeKind ()) {
32     case msrTupletLineShapeKind::kTupletLineShapeStraight:
33         break;
34     case msrTupletLineShapeKind::kTupletLineShapeCurved:
35         fResultingLpsr->
36             // this score needs the 'tuplets curved brackets' Scheme function
37             setTupletsCurvedBracketsSchemeFunctionIsNeeded ();
38         break;
39 } // switch
40 }

```

## 17.4 Deep clones

Some classes in MusicFormats, such as class `msrVoice` in `src/formats/msr/msrVoices.h/.cpp`, have a `*DeepClone ()` method:

```

1 SMARTP<msrVoice> createVoiceDeepClone (
2     int          inputLineNumber,
3     msrVoiceKind voiceKind,
4     int          voiceNumber,
5     const S_msrStaff& containingStaff);

```

Deep copies of the MSR data is not used currently. This can be changed should the need arise in the future.

## 17.5 Inheritance

### 17.5.1 Single inheritance

Many classes in MusicFormats use single inheritance. For example, in `src/formats/msr/msrTimeSignature.h`:



```

1 class EXP msrTimeSignature : public msrMeasureElement
2 {
3     public:
4
5     // creation from MusicXML
6     // -----
7
8     static SMARTP<msrTimeSignature> create (
9         int inputLineNumber,
10        const S_msrMeasure& upLinkToMeasure,
11        msrTimeSignatureSymbolKind
12            timeSignatureSymbolKind);
13
14    // creation from the applications
15    // -----
16
17    static SMARTP<msrTimeSignature> createTwoEightsTime (
18        int inputLineNumber);
19
20    // ... ..
21
22    // creation from the applications
23    // -----
24
25    static SMARTP<msrTimeSignature> createTimeFromString (
26        int inputLineNumber,
27        std::string timeString);
28
29    // ... ..

```

The definitions in in `src/formats/msr/msrTimeSignature.cpp` are:

```

1 S_msrTimeSignature msrTimeSignature::create (
2     int inputLineNumber,
3     S_msrMeasure upLinkToMeasure,
4     msrTimeSignatureSymbolKind
5         timeSignatureSymbolKind)
6 {
7     msrTimeSignature* o =
8         new msrTimeSignature (
9             inputLineNumber,
10            upLinkToMeasure,
11            timeSignatureSymbolKind);
12     assert (o != nullptr);
13     return o;
14 }
15
16 msrTimeSignature::msrTimeSignature (
17     int inputLineNumber,
18     S_msrMeasure upLinkToMeasure,
19     msrTimeSignatureSymbolKind
20         timeSignatureSymbolKind)
21     : msrMeasureElement (
22         inputLineNumber)
23 {
24     fTimeSignatureSymbolKind = timeSignatureSymbolKind;
25
26     fTimeIsCompound = false;
27 }

```

### 17.5.2 Single inheritance for smart pointers

All classes for which smart pointers are needed should inherit from class `smartable`, such as in `src/formats/msdl/msd`

```

1 class msdlScanner : public smartable
2 {
3     public:
4
5         // creation
6         // -----
7
8         static SMARTP<msdlScanner> create (std::istream& inputStream);
9
10    public:
11
12        // constructors/destructor
13        // -----
14
15        msdlScanner (std::istream& inputStream);
16
17        // ... ..
18 };

```

This leads to the following in `src/formats/msdl/msdlScanner.cpp`:

```

1 S_msdlScanner msdlScanner::create (std::istream& inputStream)
2 {
3     msdlScanner* o =
4         new msdlScanner (inputStream);
5     assert (o != nullptr);
6     return o;
7 }
8
9 msdlScanner::msdlScanner (std::istream& inputStream)
10 : fInputStream (
11     inputStream),
12   fCurrentToken (
13       ),
14   fCurrentTokenKind (
15       fCurrentToken.getTokenKindNonConst ()),
16   fCurrentTokenDescription (
17       fCurrentToken.getTokenDescriptionNonConst ())
18 {
19     // trace
20 #ifdef TRACING_IS_ENABLED
21     fTraceTokens = gGlobalMsd12msr0ahGroup->getTraceTokens ();
22     fTraceTokensDetails = gGlobalMsd12msr0ahGroup->getTraceTokensDetails ();
23 #endif
24
25     // ... ..
26 }

```

### 17.5.3 Multiple inheritance for visitors

Multiple inheritance is used extensively in visitors, which is the way to specify what elements are it seen by the visitor. For example, in `src/formats/msr/msr2msrTranslator.h`, there is:

```

1 class EXP msr2msrTranslator :
2
3     public visitor<S_msrScore>,
4
5     // rights
6
7     public visitor<S_msrIdentification>,
8
9     public visitor<S_msrCredit>,
10    public visitor<S_msrCreditWords>,

```

```

11
12     // ... ..
13 };

```

Then there are `visitStart ()` and/or `visitEnd ()` methods to handle the corresponding elements:

```

1 void msr2msrTranslator::visitStart (S_msrIdentification& elt)
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5             gLogStream <<
6                 "--> Start visiting msrIdentification" <<
7                 ", line " << elt->getInputLineNumber () <<
8                 std::endl;
9         }
10    #endif
11
12    ++gIndenter;
13
14    // set the current identification
15    fCurrentIdentification = elt;
16
17    // store it in the resulting MSR score
18    fResultingNewMsrScore->
19        setIdentification (
20            fCurrentIdentification);
21
22    fOnGoingIdentification = true;
23 }

```

```

1 void msr2msrTranslator::visitEnd (S_msrIdentification& elt)
2 {
3     fOnGoingIdentification = false;
4
5     --gIndenter;
6
7     #ifdef TRACING_IS_ENABLED
8         if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
9             gLogStream <<
10                "--> End visiting msrIdentification" <<
11                ", line " << elt->getInputLineNumber () <<
12                std::endl;
13        }
14    #endif
15 }

```

Forgetting to define those `visit* ()` methods causes *no error message whatsoever*: the corresponding elements are just not handled by the visitor.

The visitors trace options are useful to detect such cases:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/musicxml > xml2ly -find visitors
2 3 occurrences of std::string "visitors" have been found:
3     1:
4         -tmxmltvis, -trace-mxsr-visitors
5         Write a trace of the MusicXML tree visiting activity to standard error.
6     2:
7         -tmsrvis, -trace-msr-visitors
8         Write a trace of the MSR graphs visiting activity to standard error.
9     3:
10        -tlpsrvis, -trace-lpsr-visitors
11        Write a trace of the LPSR graphs visiting activity to standard error.

```

### 17.5.4 Multiple inheritance in other classes

The only such case is class `mfIndentedOstream` in `src/utilities/mfIndentedTextOutput.cpp.h/.cpp`:

```

1 class EXP mfIndentedOstream: public std::ostream, public smartable
2 {
3     /*
4     Reference for this class:
5     https://stackoverflow.com/questions/2212776/overload-handling-of-stdendl
6
7     Usage:
8     mfIndentedOstream myStream (std::cout);
9
10    myStream <<
11    1 << 2 << 3 << std::endl <<
12    5 << 6 << std::endl <<
13    7 << 8 << std::endl;
14    */
15
16    public:
17
18        // creation
19        // -----
20
21        static SMARTP<mfIndentedOstream> create (
22            std::ostream& theOStream,
23            mfOutputIndenter& theIndenter)
24        {
25            mfIndentedOstream* o = new mfIndentedOstream (
26                theOStream,
27                theIndenter);
28            assert (o != nullptr);
29
30            return o;
31        }
32
33        // constructors/destructor
34        // -----
35
36        mfIndentedOstream (
37            std::ostream& theOStream,
38            mfOutputIndenter& theIndenter)
39            : std::ostream (
40                & fIndentedStreamBuf),
41              fIndentedStreamBuf (
42                theOStream,
43                theIndenter)
44        {}
45
46        virtual ~mfIndentedOstream () {};
47
48    public:
49
50        // public services
51        // -----
52
53        // flush
54        void flush ()
55            { fIndentedStreamBuf.flush (); }
56
57        // indentation
58        mfOutputIndenter& getIndenter () const
59            { return fIndentedStreamBuf.getOutputIndenter (); }
60
61        void incrIndentation ()
62            { ++ (fIndentedStreamBuf.getOutputIndenter ()); }
63

```

```

64         void                decrIndentation ()
65                             { -- (fIndentedStreamBuf.getOutputIndenter ()); }
66
67     private:
68
69         // private fields
70         // -----
71
72         // mfIndentedOstream just uses an mfIndentedStreamBuf
73         mfIndentedStreamBuf      fIndentedStreamBuf;
74
75 };
76 typedef SMARTP<mfIndentedOstream> S_indentedOstream;

```

### 17.5.5 Reversibility

All formats in MusicFormats that can be obtained by a conversion from another one should be convertible back in the latter, without information loss.

Thus:

- MXSR contains nearly everything that can be described in MusicXML data. The main `std::exception` at the time of this writing is the MIDI information, see subsection [25.1.1](#) [MusicXML coverage], page [219](#);
- MSR contains MusicXML-related informations, so as to convert it back to MXSR;
- LSPR and BSR contain an MSR component. This is why converting those formats back to MSR is merely getting the corresponding field.

## Chapter 18

# Displaying formats

MusicFormats is equipped with option `-display*` options as a help to the maintainer.

### 18.1 Display categories

### 18.2 Displaying in practise

```
1 %void lpsr2lilypondTranslator::generateCodeForNoteRegularInMeasure (
2 %   const S_msrNote& note)
3 %{
4 %   int inputLineNumber =
5 %       note->getInputLineNumber ();
6 %
7 %#ifdef TRACING_IS_ENABLED
8 %   if (gGlobalTracingOahGroup->getTraceNotes ()) {
9 %       std::stringstream s;
10 %
11 %       s <<
12 %           std::endl <<
13 %           "% --> generating code for noteRegularInMeasure " <<
14 %           note->asString () <<
15 %           ", line " << inputLineNumber <<
16 %           std::endl;
17 %
18 %       gLogStream          << s.str ();
19 %       fLilypondCodeStream << s.str ();
20 %   }
21 %#endif
22 %
23 %
```

## Chapter 19

# Music Scores Representation (MSR)

MSR is the central format of music scores in MusicFormats. It contains a very detailed representation of western notation music score elements. Most of it is handling music in a sequential way. See chapter 20 [MSR time-oriented representation], page 200, for a presentation of how it handles time-oriented concerns.

Some of the data in MSR are supplied by the code that uses MSR, as in class `msrSlur`:

```
1  static SMARTP<msrSlur> create (
2      int                inputLineNumber ,
3      int                slurNumber ,
4      msrSlurTypeKind    slurTypeKind ,
5      msrLineTypeKind    slurLineTypeKind ,
6      msrPlacementKind   slurPlacementKind);
7
8  // ... ..
9
10 // private fields
11 // -----
12
13
14 int                fSlurNumber;
15
16 msrSlurTypeKind    fSlurTypeKind;
17
18 msrLineTypeKind    fSlurLineTypeKind;
19
20 msrPlacementKind   fSlurPlacementKind;
```

Other data are computed by the MSR private methods. For example, in `src/formats/msr/msrVoices.h`:

```
1  // there can only be 4 regular voices in a staff
2  // (those that can contain beamed notes)
3  // and we need a number for the orientation of beams
4  int                fRegularVoiceStaffSequentialNumber;
5
6  // ... ..
7
8  // fVoiceShortestNoteDuration and fVoiceShortestNoteTupletFactor
9  // are used to compute a number of divisions per quarter note
10 // if needed, such as when generating MusicXML from MSR
11 Rational           fVoiceShortestNoteDuration;
12 msrTupletFactor     fVoiceShortestNoteTupletFactor;
```

There are also data that varies during the lifetime of the object, while it is being populated for example. One such case is class `msrMeasure`:

```

1 Rational fCurrentMeasureWholeNotesDuration;
2 // this increases when musical elements
3 // are appended to the measure

```

MSR has been designed to be as general as possible, leading it to contain informations fitted to the various textual formats that can be converted to it or output from it by MusicFormats tools.

It is a *very fine-grained* representation of scores:

- some informations it contains are present as such in the textual formats;
- others are computed when the representation is populated, such as, in `src/formats/msr/msrVoices.h`:

```

1 Rational fVoiceShortestNoteDuration;

```

This information is used when generating MusicXML output to set the `<divisions/>` value.

LPSR and BSR contain an MSR as a sub-component, in order to allow for easy two-way conversion. This avoids the loss of information. This is why converting LPSR and BSR to MSR is done at no cost: just get the MSR component.

Both LPSR and BSR complement their MSR sub-component with whatever is needed for their purpose:

- LPSR contains a description of the structure of the score for the needs of LilyPond output and export from LilyPond when this becomes available;
- BSR contains a description of how to layout the braille cell on the embossed page, in terms of cells per line and lines per page.

## 19.1 MSR basic types

Some types used throughout MSR are defined in `src/formats/msr/msrBasicTypes.h/.cpp`:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > egrep -rIn '^// '
  msrBasicTypes.h
2 msrBasicTypes.h:29:// input line numbers
3 msrBasicTypes.h:34:// names lists max length
4 msrBasicTypes.h:35:// -----
5 msrBasicTypes.h:39:// XMLLang
6 msrBasicTypes.h:52:// diatonic pitches
7 msrBasicTypes.h:69:// alterations
8 msrBasicTypes.h:90:// accidentals
9 msrBasicTypes.h:124:// editorial accidentals
10 ... ..
11 msrBasicTypes.h:1840:// moments
12 msrBasicTypes.h:1938:// tuplet factors
13 msrBasicTypes.h:2024:// harmonies intervals
14 msrBasicTypes.h:2134:// harmonies structure
15 msrBasicTypes.h:2231:// harmonies contents
16 msrBasicTypes.h:2320:// harmonies details and analysis
17 msrBasicTypes.h:2333:// RGB colors
18 msrBasicTypes.h:2391:// AlphaRGB colors
19 msrBasicTypes.h:2444:// score notation kinds
20 msrBasicTypes.h:2455:// global variables
21 msrBasicTypes.h:2500:// initialization

```



## 19.2 Data matching across formats

Choices have to be made regarding the way we represent music scores elements, since this varies across formats.

In particular, the way MusicXML structures the elements is not what MSR does. For example, class `msrIdentification` in `src/formats/msr/msrIdentification.h` contains:

```

1 class EXP msrIdentification : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // work
11        std::string          fWorkNumber;
12
13        // ... ..
14
15        // creators
16
17        // ... ..
18
19        std::list<std::string>
20            fSoftwaresList;
21
22        // ... ..
23 };

```

This information is stored in distinct elements in MusicXML:

```

1 <score-partwise>
2   <work>
3     <work-number>K. 331</work-number>
4     <work-title>Piano Sonata in A Major</work-title>
5   </work>
6   <identification>
7     <creator type="composer">Wolfgang Amadeus Mozart</creator>
8     <rights>Copyright © 2003 Recordare LLC</rights>
9     <encoding>
10      <software>Finale 2003 for Windows</software>
11      <software>Dolet for Finale 1.3</software>
12      <encoding-date>2003-03-14</encoding-date>
13    </encoding>

```

The same occurs for MusicXML's `<direction/>` elements, that contain distinct subelements `<words/>` and `<metronome/>`:

```

1   <direction>
2     <direction-type>
3       <words>Adagio</words>
4     </direction-type>
5     <direction-type>
6       <metronome>
7         <beat-unit>long</beat-unit>
8         <per-minute>100</per-minute>
9       </metronome>
10    </direction-type>
11  </direction>

```

Note that two `<direction-type/>` elements are needed, since only one of `<words/>` and `<metronome/>` can be present in a given instance, as stated in `direction.mod`:

```
1 <!ELEMENT direction-type (rehearsalMark+ | segno+ | coda+ |
2 (words | symbol)+ | wedge | dynamics+ | dashes |
3 bracket | pedal | metronome | octave-shift | harp-pedals |
4 damp | damp-all | eyeglasses | std::string-mute |
5 scordatura | image | principal-voice | percussion+ |
6 accordion-registration | staff-divide | other-direction)>
```

This is not a problem in GUI applications, since all those elements are simply *drawn*. MSR stores this in a single class `msrTempo` class in `src/formats/msr/msrTempos.h/.cpp`, since musicians use tempo indications as a whole. See chapter 42 [Tempos handling], page 296 and section 19.18 [Tempos], page 175 for more details.

## 19.3 Lengths

There are several cases where a length is used in MSR, hence:

```
1 enum class msrLengthUnitKind {
2     kUnitInch, kUnitCentimeter, kUnitMillimeter
3 };
```

```
1 class EXP msrLength : public smartable
2 {
3     // ... ..
4
5     msrLengthUnitKind    fLengthUnitKind;
6     float                fLengthValue;
```

## 19.4 Sounding and displayed durations

All durations are represented by Rational numbers whose denominators are powers of 2, such as `Rational` (3, 16, and relative to the duration of a whole note).

This information is a field of class `msrMeasureElement`:

```
1 Rational                fMeasureElementSoundingWholeNotes;
```

In a tuplet, the sounding durations are different than the written durations, so we store the written duration in class `msrNote`:

```
1 // whole notes
2 Rational                fNoteDisplayWholeNotes;
3
4 int                     fNoteDotsNumber;
5
6 msrDurationKind         fNoteGraphicDurationKind;
7
8 msrTupletFactor         fNoteTupletFactor;
9
10 msrQuarterTonesPitchKind
11 fNoteQuarterTonesDisplayPitchKind;
12 msrOctaveKind          fNoteDisplayOctaveKind;
13 // for unpitched notes
14 // and pitched rests
```

Enumeration type `msrDurationKind` is declared in `src/formats/msr/msrDurations.h`:

```

1 enum class msrDurationKind {
2     kDuration_UNKNOWN,
3
4     // from longest to shortest for the algorithms
5     kDurationMaxima, kDurationLonga, kDurationBreve,
6     kDurationWhole, kDurationHalf,
7     kDurationQuarter,
8     kDurationEighth, kDuration16th, kDuration32nd, kDuration64th,
9     kDuration128th, kDuration256th, kDuration512th, kDuration1024th
10 };

```

## 19.5 Measure positions and moments

Measure positions are represented by Rational numbers such as 3/8, 1/1 being a whole note.

Measure positions are stored in field `msrMeasureElement::fMeasureElementMeasurePosition` in class `msrMeasureElement`, defined in `src/formats/msr/msrMeasureElement.h/.cpp`:

```

1 class EXP msrMeasureElement : public msrElement
2 {
3     // ... ..
4
5     protected:
6
7         // protected fields
8         // -----
9
10        /*
11         The uplink to measure is declared in the sub-classes,
12         to allow for separate *.h files, C++ constraint
13        */
14
15        Rational                fMeasureElementSoundingWholeNotes;
16
17        std::string              fBarLineUpLinkToMeasure->getMeasureNumber ();
18
19        Rational                fMeasureElementMeasurePosition;
20        Rational                fMeasureElementVoicePosition;
21 };

```

LilyPond represents grace notes positions with a so-called *moment*, that complement the measure position with a relative offset. Grace notes durations are not accounted for in the whole notes duration of measures.

Class `msrMoment` stores a position in a measure, with a relative offset since harmonies can be placed on a note during its sounding time:

```

1     Rational                fWrittenPositionInMeseasure;
2     Rational                fSoundingRelativeOffset;

```

## 19.6 Rests and skips

A skip is an invisible rest, i.e. the meaning is the same as that in LilyPond. Skips are created to fill the holes between notes wherever needed, in order for all voices to be notes/rests/skips sequences.

Skips are also created in `src/passes/msr2lpsr/` to circumvent the LilyPond #34 issue.

## 19.7 Solo notes and rests

A solo note or rest is characterized as sounding alone in its multi-voice staff for its whole duration.

In the case of a solo rests, such detection allows for better output, in particular when LilyPond code is generated.

An example is at figure 19.1 [The solo rests problem], page 168 : the eighth rests in the second measure of voice 1 of the first staff should be placed on the middle line of the staff, as MuseScore does.

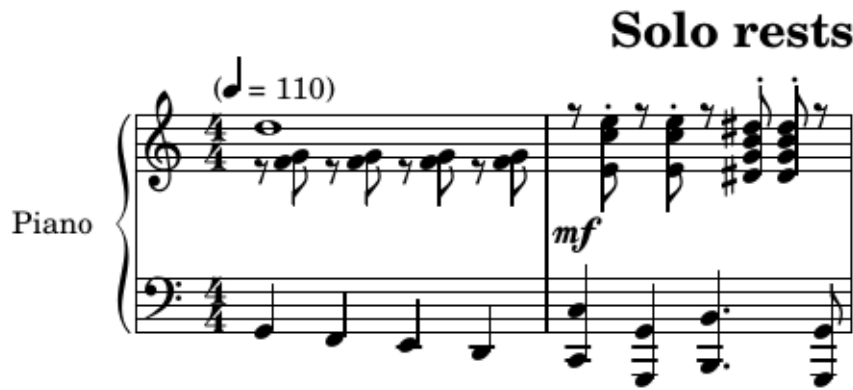


Figure 19.1: The solo rests problem

## 19.8 Linear versus time-oriented representation

Most music scoring GUI applications handle music as containing voices, which are made of sequences of notes, chords, tuplets and such. This is a horizontal, linear view of the music in the score.

Another view of the music is time-oriented, i.e., what are notes being played at a given moment in time? This is a vertical view of the music, which is highlighted in piano roll views.

MSR stores descriptions of so-called 'measures slice' through class `msrMeasuresSlice`, defined in `src/formats/msr/msrMeasuresSlice.h`. Then a time-oriented view of a voice, staff or part is a sequence of such measure slices, defined in class `msrMeasuresSlicesSequence`.

An class `msrMeasuresSlice` contains basically a slice measures vector:

```
1 // the measures in the slice
2 std::vector<S_msrMeasure> fSliceMeasuresVector;
```

From this, the following other descriptions are derived:

```
1 // notes flat list
2 std::list<S_msrNote> fSliceNotesFlatList;
3
4 // note events list
5 std::list<S_msrNoteEvent>
6     fSliceNoteEventsList;
7
8 // simultaneous notes chunks list
9 std::list<S_msrSimultaneousNotesChunk>
10     fSliceSimultaneousNotesChunksList;
```

Note events are distinguished with enumeration type `msrNoteEventKind`:

```
1 // -----
2 enum class msrNoteEventKind {
3     kNoteEventStart,
4     kNoteEventStop
5 };
```

Class `msrNoteEvent` contains:

```
1 Rational          fNoteEventMeasurePosition;
2 S_msrNote         fNoteEventNote;
3 msrNoteEventKind  fNoteEventKind;
```

## 19.9 Spanners

A spanner... spans from one note or rest to another one. A choice to be made about when to use spanners: should wedges < and > be handled as spanners, or simply as being attached to notes? It has been chosen to use spanners only for ligatures apart from true spanners.

MusicXML uses "start", "start" and "start" attributes, which need to be present in MSR for MusicXML generation. They are reflected in MSR as enumeration type enumeration type `msrSpannerTypeKind`, defined this way:

```
1 // spanner types
2 // -----
3 enum class msrSpannerTypeKind {
4     kSpannerType_UNKNOWN,
5
6     kSpannerTypeStart, kSpannerTypeContinue, kSpannerTypeStop
7 };
```

## 19.10 Uplinks, direct uplinks and sidelinks

An uplink is a direct pointer from one class instance to one that contains it. Some are a link to the containing class instance, whilst others are shortcut links higher in the graph for speed. For example, class `msrNote` contains:

```
1 // upLinks
2 // -----
3
4 S_msrMeasure          fNoteUpLinkToMeasure;
5
6 S_msrChord            fNoteShortcutUpLinkToChord;
7
8 S_msrGraceNotesGroup  fNoteShortcutUpLinkToGraceNotesGroup;
9
10 S_msrTuplet           fNoteShortcutUpLinkToTuplet;
```

A sidelink is used in ligatures and spanners, so that each end of the structure can reference the other one.

For example, MusicFormats defines enumeration type `msrLigatureKind` in `src/formats/msr/msrLigatures.h`:

```
1 enum class msrLigatureKind {
2     kLigatureNone,
3     kLigatureStart, kLigatureContinue, kLigatureStop
4 };
```

Class `msrLigature` contains:

```

1 private:
2
3     // sideLinks
4     // -----
5     S_msrLigature      fLigatureSideLinkToOtherEnd; // two-way

```

Enumeration type `is` is declared in `src/formats/msr/msrSpanners.h`:

```

1 enum class msrSpannerKind {
2     kSpannerDashes, kSpannerWavyLine
3 };

```

## 19.11 Printing descriptions

There is a standard set of methods to print the contents of the descriptions in `MusicFormats` to standard output, depending on the granularity of the information to be displayed:

```

1 void          print (std::ostream& os) const override;
2
3 std::string    asString () const override;
4 std::string    asStringShort () const override;

```

There are also more specific methods such as:

```

1 void          printFull (std::ostream& os) const override;
2
3 void          printSummary (std::ostream& os) const override;

```

Note that:

- virtual method `asString ()` produces a rather condensed view of the data to be displayed as part of a single line;
- virtual method `print ()` may produce its output on multiples lines, which always ends with an end of line.

Most classes in `MusicFormats` can be printed with the `<<` operator:

```

1 std::ostream& operator << (std::ostream& os, const S_msrElement& elt)
2 {
3     if (elt) {
4         elt->print (os);
5     }
6     else {
7         os << "[NONE]" << std::endl;
8     }
9
10    return os;
11 }

```

In simple cases, virtual method `print ()` merely calls virtual method `asString ()`:

```

1 void msrElement::print (std::ostream& os) const
2 {
3     os << asString () << std::endl;
4 }

```

All virtual method `asString ()` methods produce an output of the form [...], in order to facilitate selecting the whole with a double click to help the user, since such output can be nested:

```

1 std::string msrTransposition::asString () const
2 {
3     std::stringstream s;
4
5     s <<
6     "[Transposition" <<
7     ", fTranspositionDiatonic = " << fTranspositionDiatonic <<
8     ", fTranspositionChromatic = " << fTranspositionChromatic <<
9     ", fTranspositionOctaveChange = " << fTranspositionOctaveChange <<
10    ", fTranspositionDouble = " << fTranspositionDouble <<
11    ", line " << fInputLineNumber <<
12    ']' ;
13
14    return s.str ();
15 }
```

A typical sequence to produce indented output is:

```

1 void msrTransposition::print (std::ostream& os) const
2 {
3     const int fieldWidth = 22;
4
5     os <<
6     "Transposition" <<
7     ", line " << fInputLineNumber <<
8     std::endl;
9
10    ++gIndenter;
11
12    os << std::left <<
13    std::setw (fieldWidth) <<
14    "fTranspositionDiatonic" << " = " << fTranspositionDiatonic <<
15    std::endl <<
16    std::setw (fieldWidth) <<
17    "fTranspositionChromatic" << " = " << fTranspositionChromatic <<
18    std::endl <<
19    std::setw (fieldWidth) <<
20    "fTranspositionOctaveChange" << " = " << fTranspositionOctaveChange <<
21    std::endl <<
22    std::setw (fieldWidth) <<
23    "fTranspositionDouble" << " = " << fTranspositionDouble <<
24    std::endl << std::endl;
25
26    --gIndenter;
27 }
```

The main indented output streams are:

```

1 #define gOutputStream *gGlobalOutputIndentedOstream
2 #define gLogStream     *gGlobalLogIndentedOstream
```

## 19.12 Pitches

MSR handle diatonic, semitone and quarter tone pitches, defined in `src/formats/msr/msrBasicTypes.h` as shown below. All pitches data is represented internally as quarter tones pitches, and conversions are done wherever needed.

```

1 // diatonic pitches
2 // -----
3 enum class msrDiatonicPitchKind {
4     kDiatonicPitch_UNKNOWN,
5
6     // starting at C for LilyPond relative octave calculations
7     kDiatonicPitchC,
8     kDiatonicPitchD, kDiatonicPitchE, kDiatonicPitchF,
9     kDiatonicPitchG, kDiatonicPitchA, kDiatonicPitchB
10 };

```

```

1 // semi tones pitches
2 // -----
3 enum class msrSemiTonesPitchKind {
4     kSTP_NoSemiTonesPitch,
5
6     kSTP_C_TripleFlat,
7     kSTP_C_DoubleFlat, kSTP_C_Flat,
8     kSTP_C_Natural,
9     kSTP_C_Sharp, kSTP_C_DoubleSharp,
10    kSTP_C_TripleSharp,
11
12    kSTP_D_TripleFlat,
13    kSTP_D_DoubleFlat, kSTP_D_Flat,
14    kSTP_D_Natural,
15    kSTP_D_Sharp, kSTP_D_DoubleSharp,
16    kSTP_D_TripleSharp,
17
18    kSTP_E_TripleFlat,
19    kSTP_E_DoubleFlat, kSTP_E_Flat,
20    kSTP_E_Natural,
21    kSTP_E_Sharp, kSTP_E_DoubleSharp,
22    kSTP_E_TripleSharp,
23
24    kSTP_F_TripleFlat,
25    kSTP_F_DoubleFlat, kSTP_F_Flat,
26    kSTP_F_Natural,
27    kSTP_F_Sharp, kSTP_F_DoubleSharp,
28    kSTP_F_TripleSharp,
29
30    kSTP_G_TripleFlat,
31    kSTP_G_DoubleFlat, kSTP_G_Flat,
32    kSTP_G_Natural,
33    kSTP_G_Sharp, kSTP_G_DoubleSharp,
34    kSTP_G_TripleSharp,
35
36    kSTP_A_TripleFlat,
37    kSTP_A_DoubleFlat, kSTP_A_Flat,
38    kSTP_A_Natural,
39    kSTP_A_Sharp, kSTP_A_DoubleSharp,
40    kSTP_A_TripleSharp,
41
42    kSTP_B_TripleFlat,
43    kSTP_B_DoubleFlat, kSTP_B_Flat,
44    kSTP_B_Natural,
45    kSTP_B_Sharp, kSTP_B_DoubleSharp,
46    kSTP_B_TripleSharp
47 };

```

```

1 // quarter tones pitches
2 // -----
3 enum class msrQuarterTonesPitchKind {
4     kQTP_UNKNOWN,
5
6     kQTP_Rest, kQTP_Skip,

```



```

7
8 kQTP_A_TripleFlat,
9 kQTP_A_DoubleFlat, kQTP_A_SesquiFlat, kQTP_A_Flat, kQTP_A_SemiFlat,
10 kQTP_A_Natural,
11 kQTP_A_SemiSharp, kQTP_A_Sharp, kQTP_A_SesquiSharp, kQTP_A_DoubleSharp,
12 kQTP_A_TripleSharp,
13
14 kQTP_B_TripleFlat,
15 kQTP_B_DoubleFlat, kQTP_B_SesquiFlat, kQTP_B_Flat, kQTP_B_SemiFlat,
16 kQTP_B_Natural,
17 kQTP_B_SemiSharp, kQTP_B_Sharp, kQTP_B_SesquiSharp, kQTP_B_DoubleSharp,
18 kQTP_B_TripleSharp,
19
20 kQTP_C_TripleFlat,
21 kQTP_C_DoubleFlat, kQTP_C_SesquiFlat, kQTP_C_Flat, kQTP_C_SemiFlat,
22 kQTP_C_Natural,
23 kQTP_C_SemiSharp, kQTP_C_Sharp, kQTP_C_SesquiSharp, kQTP_C_DoubleSharp,
24 kQTP_C_TripleSharp,
25
26 kQTP_D_TripleFlat,
27 kQTP_D_DoubleFlat, kQTP_D_SesquiFlat, kQTP_D_Flat, kQTP_D_SemiFlat,
28 kQTP_D_Natural,
29 kQTP_D_SemiSharp, kQTP_D_Sharp, kQTP_D_SesquiSharp, kQTP_D_DoubleSharp,
30 kQTP_D_TripleSharp,
31
32 kQTP_E_TripleFlat,
33 kQTP_E_DoubleFlat, kQTP_E_SesquiFlat, kQTP_E_Flat, kQTP_E_SemiFlat,
34 kQTP_E_Natural,
35 kQTP_E_SemiSharp, kQTP_E_Sharp, kQTP_E_SesquiSharp, kQTP_E_DoubleSharp,
36 kQTP_E_TripleSharp,
37
38 kQTP_F_TripleFlat,
39 kQTP_F_DoubleFlat, kQTP_F_SesquiFlat, kQTP_F_Flat, kQTP_F_SemiFlat,
40 kQTP_F_Natural,
41 kQTP_F_SemiSharp, kQTP_F_Sharp, kQTP_F_SesquiSharp, kQTP_F_DoubleSharp,
42 kQTP_F_TripleSharp,
43
44 kQTP_G_TripleFlat,
45 kQTP_G_DoubleFlat, kQTP_G_SesquiFlat, kQTP_G_Flat, kQTP_G_SemiFlat,
46 kQTP_G_Natural,
47 kQTP_G_SemiSharp, kQTP_G_Sharp, kQTP_G_SesquiSharp, kQTP_G_DoubleSharp,
48 kQTP_G_TripleSharp
49 };

```

## 19.13 Octaves

They are represented with enumeration type :

```

1 // octaves
2 // -----
3 enum class msrOctaveKind {
4     kOctave_UNKNOWN,
5
6     kOctave0, kOctave1, kOctave2, kOctave3,
7     kOctave4, // that of middle C
8     kOctave5, kOctave6, kOctave7, kOctave8, kOctave9
9 };

```

For the needs of LilyPond and MSDL, MSR also contains a description of how to enter octaves:

```

1 // octave entry
2 // -----
3 enum class msrOctaveEntryKind {

```

```

4   kOctaveEntryRelative,
5   kOctaveEntryAbsolute,
6   kOctaveEntryFixed
7 };

```

## 19.14 Durations

MusicFormats represents durations with enumeration type `msrDurationKind`, defined in `src/formats/msr/msrBasic`

```

1  // durations
2  // -----
3  enum class msrDurationKind {
4      kDuration_UNKNOWN,
5
6      // from longest to shortest for the algorithms
7      kDurationMaxima, kDurationLonga, kDurationBreve, kDurationWhole, kDurationHalf,
8      kDurationQuarter,
9      kDurationEighth, kDuration16th, kDuration32nd, kDuration64th, kDuration128th,
10     kDuration256th, kDuration512th, kDuration1024th
11 };

```

## 19.15 Alterations

```

1  // alterations
2  // -----
3  enum class msrAlterationKind {
4      kAlteration_UNKNOWN,
5
6      kAlterationTripleFlat, kAlterationDoubleFlat, kAlterationSesquiFlat,
7      kAlterationFlat, kAlterationSemiFlat,
8      kAlterationNatural,
9      kAlterationSemiSharp, kAlterationSharp, kAlterationSesquiSharp,
10     kAlterationDoubleSharp, kAlterationTripleSharp

```

## 19.16 Accidentals

```

1  // accidentals
2  // -----
3  enum class msrAccidentalKind {
4      kAccidentalNone,
5
6      kAccidentalSharp, kAccidentalNatural,
7      kAccidentalFlat, kAccidentalDoubleSharp,
8      kAccidentalSharpSharp,
9      kAccidentalFlatFlat, kAccidentalNaturalSharp,
10     kAccidentalNaturalFlat, kAccidentalQuarterFlat,
11     kAccidentalQuarterSharp, kAccidentalThreeQuartersFlat,
12     kAccidentalThreeQuartersSharp,
13
14     kAccidentalSharpDown, kAccidentalSharpUp,
15     kAccidentalNaturalDown, kAccidentalNaturalUp,
16     kAccidentalFlatDown, kAccidentalFlatUp,
17     kAccidentalTripleSharp, kAccidentalTripleFlat,
18     kAccidentalSlashQuarterSharp, kAccidentalSlashSharp,
19     kAccidentalSlashFlat, kAccidentalDoubleSlashFlat,
20     kAccidentalSharp_1, kAccidentalSharp_2,
21     kAccidentalSharp_3, kAccidentalSharp_5,

```

```

22 kAccidentalFlat_1, kAccidentalFlat_2,
23 kAccidentalFlat_3, kAccidentalFlat_4,
24 kAccidentalSori, kAccidentalKoron,
25
26 kAccidentalOther
27 };

```

## 19.17 Durations

They are represented in MSR with the enumeration type `msrDurationKind` enumeration type, defined in `src/formats/msr/msrBasicTypes.h`:

```

1 // durations
2 // -----
3 enum class msrDurationKind {
4     kDuration_UNKNOWN,
5
6     // from longest to shortest for the algorithms
7     kDurationMaxima, kDurationLonga, kDurationBreve, kDurationWhole, kDurationHalf,
8     kDurationQuarter,
9     kDurationEighth, kDuration16th, kDuration32nd, kDuration64th, kDuration128th,
10    kDuration256th, kDuration512th, kDuration1024th
11 };

```

Class `msrDottedDuration` contains:

```

1     msrDurationKind      fDurationKind;
2     int                  fDotsNumber;

```

## 19.18 Tempos

There are thus several kinds of tempos in MSR, with variants represented by enumeration type `msrTempoKBeatUnitsKind` in `src/formats/msr/msrTempos.h`:

```

1 class EXP msrTempo : public msrMeasureElement
2 {
3     public:
4
5     // data types
6     // -----
7
8     enum class msrTempoKBeatUnitsKind {
9         kTempoBeatUnits_UNKNOWN,
10        kTempoBeatUnitsWordsOnly,
11        kTempoBeatUnitsPerMinute,
12        kTempoBeatUnitsEquivalence,
13        kTempoNotesRelationship
14    };
15
16    // ... ..
17
18    enum class msrTempoParenthesizedKind {
19        kTempoParenthesizedYes, kTempoParenthesizedNo
20    };
21
22    // ... ..
23
24    enum class msrTempoNotesRelationshipKind {
25        kTempoNotesRelationshipNone, kTempoNotesRelationshipEquals

```

```

26     };
27
28     // ... ..
29 };

```

### 19.18.1 Tempos notes

A tempo indication can contain a note a notes in a tuplet. Such notes are described by class `msrTempoNote`:

```

1  class EXP msrTempoNote : public msrElement
2  {
3      public:
4
5          // creation from MusicXML
6          // -----
7
8          static SMARTP<msrTempoNote> create (
9              int                inputLineNumber ,
10             const Rational&    tempoNoteWholeNotes ,
11             Bool               tempoNoteBelongsToATuplet);
12
13     protected:
14
15         // constructors/destructor
16         // -----
17
18         msrTempoNote (
19             int                inputLineNumber ,
20             const Rational&    tempoNoteWholeNotes ,
21             Bool               tempoNoteBelongsToATuplet);
22
23         // ... ..
24
25     private:
26
27         // private fields
28         // -----
29
30         Rational                fTempoNoteWholeNotes;
31
32         std::list<S_msrBeam>    fTempoNoteBeams;
33
34         Bool                   fTempoNoteBelongsToATuplet;
35 };

```

### 19.18.2 Tempos tuplets

A tuplet in a tempo representation is described by class `msrTempoTuplet`:

```

1  // -----
2  class EXP msrTempoTuplet : public msrElement
3  {
4      public:
5
6          // data types
7          // -----
8
9          enum class msrTempoTupletTypeKind {
10             kTempoTupletTypeNone ,
11             kTempoTupletTypeStart , kTempoTupletTypeStop
12         };
13

```

```

14 // ... ..
15
16 enum class msrTempoTupletBracketKind {
17     kTempoTupletBracketYes, kTempoTupletBracketNo
18 };
19
20 // ... ..
21
22 enum class msrTempoTupletShowNumberKind {
23     kTempoTupletShowNumberActual,
24     kTempoTupletShowNumberBoth,
25     kTempoTupletShowNumberNone
26 };
27
28 // ... ..
29
30 // creation from MusicXML
31 // -----
32
33 static SMARTP<msrTempoTuplet> create (
34     int                inputLineNumber,
35     int                tempoTupletNumber,
36     msrTempoTupletBracketKind tempoTupletBracketKind,
37     msrTempoTupletShowNumberKind tempoTupletShowNumberKind,
38     msrTupletFactor    tempoTupletFactor,
39     Rational           memberNotesDisplayWholeNotes);
40
41 protected:
42
43 // constructors/destructor
44 // -----
45
46 msrTempoTuplet (
47     int                inputLineNumber,
48     int                tempoTupletNumber,
49     msrTempoTupletBracketKind tempoTupletBracketKind,
50     msrTempoTupletShowNumberKind tempoTupletShowNumberKind,
51     msrTupletFactor    tempoTupletFactor,
52     Rational           memberNotesDisplayWholeNotes);
53
54 // ... ..
55
56 private:
57
58 // private fields
59 // -----
60
61 int                fTempoTupletNumber;
62
63 msrTempoTupletBracketKind
64     fTempoTupletBracketKind;
65
66 msrTempoTupletShowNumberKind
67     fTempoTupletShowNumberKind;
68
69 msrTupletFactor    fTempoTupletFactor;
70
71 Rational           fMemberNotesDisplayWholeNotes;
72
73 Rational           fTempoTupletDisplayWholeNotes;
74
75 std::list<S_msrElement>
76     fTempoTupletElements;
77 };

```

### 19.18.3 Tempos description

The private fields in class `msrTempo` are:

```

1 class EXP msrTempo : public msrMeasureElement
2 {
3     // ... ..
4
5     private:
6
7     // private fields
8     // -----
9
10    msrTempoKBeatUnitsKind
11                                fTempoKind;
12
13    std::list<S_msrWords> fTempoWordsList;
14
15    msrDottedDuration        fTempoBeatUnit;
16
17    std::string               fTempoPerMinute; // '90' or '132-156' for example
18    msrDottedDuration        fTempoEquivalentBeatUnit;
19
20    S_msrTempoNotesRelationshipElements
21                                fTempoNotesRelationshipLeftElements;
22    msrTempoNotesRelationshipKind
23                                fTempoNotesRelationshipKind;
24    S_msrTempoNotesRelationshipElements
25                                fTempoNotesRelationshipRightElements;
26
27    msrTempoParenthesizedKind
28                                fTempoParenthesizedKind;
29
30    msrPlacementKind          fTempoPlacementKind;
31 };

```

Among these fields:

- field `msrTempo::fTempoKind` denotes the variant;
- field `msrTempo::fTempoWordsList` contains the words that can be present, such as 'adagio molto';
- field `msrTempo::fTempoBeatUnit` is a dotted duration, as in '4.';
- field `msrTempo::fTempoPerMinute` is a `std::string`, since it can contain ranges indication as in '4. = 60-66';
- field `msrTempo::fTempoEquivalentBeatUnit` is a dotted duration;
- field `msrTempo::fTempoNotesRelationshipLeftElements`, field `msrTempo::fTempoNotesRelationshipKind` and field `msrTempo::fTempoNotesRelationshipRightElements` are used when a relationship is present, such as '2. = 1', in which case field `msrTempo::fTempoNotesRelationshipKind` contains field `msrTempo::kTempoNotesRelationshipKind`;
- field `msrTempo::fTempoParenthesizedKind` indicates whether the tempo indication is parenthesized;
- field `msrTempo::fTempoPlacementKind` tells whether the tempo is to be placed above or below the staff, constant `msrPlacementKind::kPlacementAbove` by default.

## 19.19 Clefs

Clefs are distinguished using enumeration type `msrClefKind`:

```

1 // clefs
2 // -----
3
4 enum class msrClefKind {
5     kClef_UNKNOWN,
6
7     kClef_Treble,
8     kClef_Soprano, kClef_MezzoSoprano, kClef_Alto, kClef_Tenor, kClef_Baritone, kClef_Bass,
9     kClef_TrebleLine1,
10    kClef_TrebleMinus15, kClef_TrebleMinus8, kClef_TreblePlus8, kClef_TreblePlus15,
11
12    kClef_BassMinus15, kClef_BassMinus8, kClef_BassPlus8, kClef_BassPlus15,
13
14    kClef_Varbaritone,
15
16    kClef_Tablature4, kClef_Tablature5, kClef_Tablature6, kClef_Tablature7,
17
18    kClef_Percussion,
19
20    kClef_Jianpu
21 };

```

Class `msrClef` contains:

```

1     msrClefKind      fClefKind;
2     int              fClefStaffNumber;

```

## 19.20 Keys

MSR, as MusicXML, supports Humdrum-Scot keys as well as traditional key such as C and 6/8.

A Humdrum-Scot key is composed of items represented by class `msrHumdrumScotKeyItem`, each containing:

```

1     msrDiatonicPitchKind  fKeyDiatonicPitchKind;
2     msrAlterationKind     fKeyAlterationKind;
3     msrOctaveKind         fKeyOctaveKind;

```

An example is at figure 19.2 [Humdrum-Scot keys], page 180. It has been produced by:

```

1 xml2ly -auto-output-file-name keys/HumdrumScotKeys.xml

```

Class `msrKey` thus contains:

```

1     // private fields
2     // -----
3
4     msrKeyKind      fKeyKind;
5
6     // traditional keys
7
8     msrQuarterTonesPitchKind
9         fKeyTonicQuarterTonesPitchKind;
10    msrModeKind      fModeKind;
11    int              fKeyCancel;
12
13    // Humdrum/Scot keys
14
15    std::vector<S_msHumdrumScotKeyItem>

```

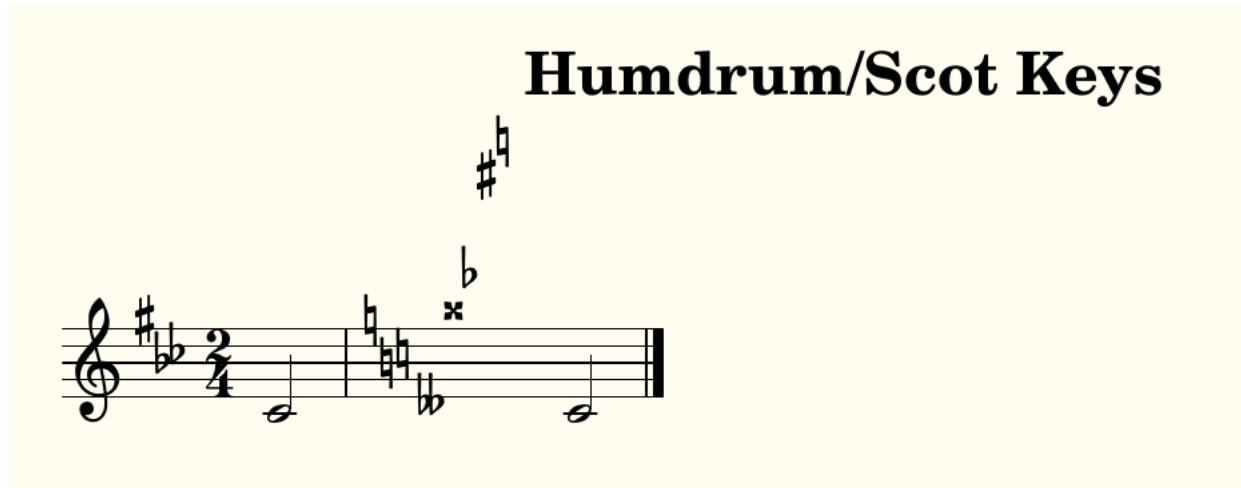


Figure 19.2: Humdrum-Scot keys

```
16         fHumdrumScotKeyItemsVector;  
17     Bool    fKeyItemsOctavesAreSpecified;
```

## 19.21 Time signatures

The variants in time signatures are distinguished by enumeration type `enum type msrTimeSignatureSymbolKind`

```
1 // time symbols
2 // -----
3 enum class msrTimeSignatureSymbolKind {
4     kTimeSignatureSymbolNone,
5     kTimeSignatureSymbolCommon,
6     kTimeSignatureSymbolCut,
7     kTimeSignatureSymbolNote,
8     kTimeSignatureSymbolDottedNote,
9     kTimeSignatureSymbolSingleNumber,
10    kTimeSignatureSymbolSenzaMisura
11 };
```

A time signature can also be structured, and this is described by those two types:

```
1 enum class msrTimeSignatureSeparatorKind {
2     kTimeSignatureSeparatorNone,
3     kTimeSignatureSeparatorHorizontal,
4     kTimeSignatureSeparatorDiagonal,
5     kTimeSignatureSeparatorVertical,
6     kTimeSignatureSeparatorAdjacent
7 };
```

```
1 enum class msrTimeSignatureRelationKind {
2     kTimeSignatureRelationNone,
3     kTimeSignatureRelationParentheses,
4     kTimeSignatureRelationBracket,
5     kTimeSignatureRelationEquals,
6     kTimeSignatureRelationSlash,
7     kTimeSignatureRelationSpace,
8     kTimeSignatureRelationHyphen
9 };
```

A brick that can be used in class `msrTimeSignature` is `msrTimeSignatureItem`, whose private fields are:



```

1  std::vector<int>      fTimeSignatureBeatsNumbersVector; // 5+3+1 is possible
2  int                  fTimeSignatureBeatValue;

```

Class `msrTimeSignature` contains:

```

1  msrTimeSignatureSymbolKind
2      fTimeSignatureSymbolKind;
3
4  std::vector<S_msrTimeSignatureItem>
5      fTimeSignatureItemsVector;
6
7  // a time is compound if it contains several items
8  // or if the only one has several beats numbers
9  // i.e. 3/4 is not, (3+4)/8 is, and 2/4+3/4 is too
10 Bool      fTimeIsCompound;

```

## 19.22 MSR classes inheritance

The picture at figure 19.3 [The MSR classes hierarchy], page 182, shows the hierarchy of the main MSR classes. The colors are used as follows:

The background colors are used as follows:

- **green**: a score element that is expected to be found in a score representation, such as class `msrStaff` and class `msrChord`;
- **pink**: a element needed in MSR to structure the representation, such as class `msrSegment` and class `msrSyllable`;
- **yellow**: a base class with name class `msr*Element` for elements that can be used in another class, such as class `msrVoiceElement`;

The arrows colors have the following meaning:

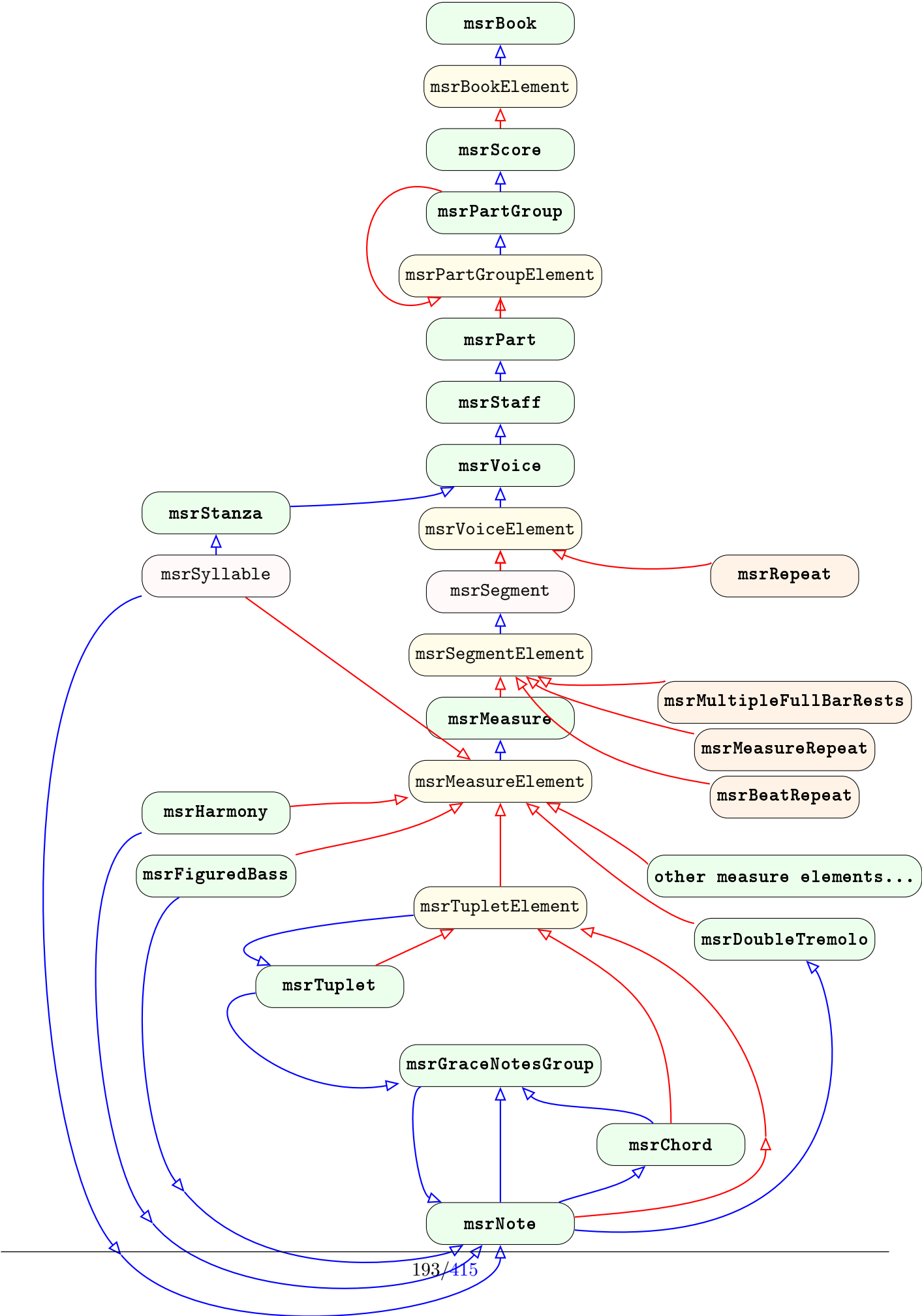
- **red**: a link from a class to its base class. For example, class `msrPart` is derived from class `msrPartGroupElement`, class `msrPartGroup` is derived from class `msrPartGroupElement`, and class `msrChord` is derived from class `msrTupletElement`;
- **blue**: one or more fields of a class are smart pointers to instances of another. For example, an class `msrChords` instance may be an element of a class `msrGraceNotesGroup` instance.

When not shown for clarity, the common base class of all these classes is class `msrElement`, that contains an integer input line number.

The `otherMeasureElements` classes are:

- bars:
  - class `msrBarCheck`
  - class `msrBarNumberCheck`
  - class `msrBarLine`
  - class `msrHiddenMeasureAndBarLine`
- breaks:

Figure 19.3: The MSR classes hierarchy



- class `msrLineBreak`
  - class `msrPageBreak`
- notes:
  - class `msrVoiceStaffChange`
  - class `msrOctaveShift`
- clefs, keys, times, tempo:
  - class `msrClef`
  - class `msrKey`
  - class `msrTime`
  - class `msrTempo`
- instruments:
  - class `msrStaffDetails`
  - class `msrScordatura`
  - class `msrAccordionRegistration`
  - class `msrHarpPedalsTuning`
  - class `msrPedal`
  - class `msrDamp`
  - class `msrDampAll`
- lyrics:
  - class `msrSyllable`
- rehearsals, segno and coda:
  - class `msrRehearsalMark`
  - class `msrSegno`
  - class `msrDalSegno`
  - class `msrCoda`
- others:
  - class `msrPrintLayout`
  - class `msrEyeGlasses`
  - class `msrStaffLevelElement`
  - class `msrTransposition`
  - class `msrTupletElement`

## 19.23 Books

Books handling is presented at section 54 [Books handling], page 320.

LilyPond handles `\book {...}` by placing the scores one after the other in the resulting PDF or SVG files. It will also generate separate MIDI files if a `\markup {...}` block is used.

There is no such concept in MusicXML, but MSR uses it for completeness, creating an implicit class `msrBook` instance if needed.

An class `msrBook` contains a list and a set of `S_msrBookElement`:

```
1 // book elements
2 std::set<S_msrBookElement> fBookElementsSet;
3
4 std::list<S_msrBookElement> fBookElementsList;
```

Currently, the only book element used is the class `msrScore`, but others might come, such as texts, which LilyPond allows as `\markup {...}`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrBook' *
2 formats/msr/msrScores.h:class EXP msrScore : public msrBookElement
```

## 19.24 Scores

Scores handling is presented at section ?? [Scores handling], page ??.

A score in MSR is the usual music score concept. It contains a set and a list of `S_msrPartGroup`:

```
1 // part groups
2 std::set<S_msrPartGroup> fScorePartGroupsSet;
3
4 std::list<S_msrPartGroup> fPartGroupsList;
```

## 19.25 Part groups

Part groups handling is presented at section 52 [Part groups handling], page 317.

A part group in MSR contains parts or other part groups. This concept is recursive, as it is in music score: the winds part group can oboes and horns part group, for example. An implicit part group exists in MSR if the score does not contain explicit part groups.

An class `msrPartGroup` thus contains parts and part groups in any order, as is found in symphonic music scores:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrPartGroupElement' *
2 formats/msr/msrParts.h:class EXP msrPart : public msrPartGroupElement
3 formats/msr/msrPartGroups.h:class EXP msrPartGroup : public msrPartGroupElement
```

which are stored in a list:

```
1 // allowing for both parts and (sub-)part groups as elements
2 std::list<S_msrPartGroupElement>
3 fPartGroupElementsList;
```

## 19.26 Parts

Parts handling is presented at section 51 [Parts handling], page 315.

A part in MSR is composed of voices, stored in:

```

1  // staves
2
3  std::map<int, S_msrStaff>
4      getPartStaveNumbersToStavesMap;
5  std::list<S_msrStaff> fPartAllStavesList;
6
7  // harmonies
8
9  S_msrStaff          fPartHarmoniesStaff;
10 S_msrVoice          fPartHarmoniesVoice;
11
12 // figured bass
13
14 S_msrStaff          fPartFiguredBassStaff;
15 S_msrVoice          fPartFiguredBassVoice;
16
17 // voices
18
19 std::list<S_msrVoice> fPartAllVoicesList;

```

## 19.27 Staves

Staves handling is presented at section ?? [Staves handling], page ??.

A stave contains at most 4 numbered voices, stored in:

```

1  // the mapping of all the voices in the staff,
2  // including harmonies and figured bass voices
3  std::map<int, S_msrVoice> fStaffVoiceNumbersToAllVoicesMap;
4
5  // the mapping of voice numbers to regular voices
6  std::map<int, S_msrVoice> fStaffVoiceNumbersToRegularVoicesMap;
7
8  // we need to handle the regular voice specifically
9  // to assign them sequencing numbers from 1 to gMaxStaffVoices,
10 // needed to set the beams orientation (up or down)
11 int          fStaffRegularVoicesCounter;
12
13 // harmonies and figured bass elements should be placed %%JMI
14 // in the first regular voice of the staff, hence:
15 std::list<S_msrVoice> fStaffRegularVoicesList;
16
17 // we need to sort the voices by increasing voice numbers,
18 // but with harmonies voices right before the corresponding regular voices
19 std::list<S_msrVoice> fStaffAllVoicesList;

```

## 19.28 Voice elements

Voices contain instances of class `msrVoiceElement`, defined in `src/formats/msr/msrVoiceElements.h/.cpp`:

```

1 // -----
2 /*
3  Various elements can found in voices,
4  hence class   msrVoiceElement
5  */
6
7 class EXP msrVoiceElement : public msrElement
8 {
9     public:
10
11     // creation from MusicXML
12     // -----
13
14     // cloning
15     // -----
16
17     protected:
18
19         msrVoiceElement (
20             int inputLineNumber);
21
22     virtual          ~msrVoiceElement ();
23
24     /*
25     The voice uplink is declared in the sub-classes,
26     to allow for separate *.h files, C++ constraint
27     */
28 };

```

The classes derived from class `msrVoiceElement` are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'public msrVoiceElement' *
   *.h
2 msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
3 msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
4 msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
5 msrMultipleFullBarRests.h:class EXP msrMultipleFullBarRests : public msrVoiceElement
6 msrSegments.h:class EXP msrSegment : public msrVoiceElement

```

They are describes in specific sections below.

## 19.29 Voices

Voices handling is presented at section [49](#) [Voices handling], page [313](#).

A voice is conceptually a sequence of `S_msVoiceElement`, that may be:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrVoiceElement' *
2 formats/msr/msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
3 formats/msr/msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
4 formats/msr/msrMultipleFullBarRests.h:class EXP msrMultipleFullBarRests : public
   msrVoiceElement
5 formats/msr/msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
6 formats/msr/msrSegments.h:class EXP msrSegment : public msrVoiceElement

```

More precisely and for technical reasons, an class `msrVoice` contains:

```

1 // voice initial elements list
2
3 std::list<S_msrVoiceElement>
4     fVoiceInitialElementsList;
5
6 // voice first and last segments
7
8 // fVoiceLastSegment contains the music
9 // not yet stored in fVoiceInitialElementsList,
10 // it is thus logically the end of the latter,
11 // and is created implicitly for every voice.
12 // It is needed 'outside' of the 'list<S_msrElement>'
13 // because it is not a mere S_msrElement, but a S_msrSegment
14 S_msrSegment     fVoiceLastSegment;
15
16 // fVoiceFirstSegment is used to work around LilyPond issue #34
17 S_msrSegment     fVoiceFirstSegment;

```

Each voice is described by a field of enumeration type `msrVoiceKind`, defined in `src/formats/msr/msrBasicTypes.h`:

```

1 enum class msrVoiceKind {
2     kVoiceKindRegular,
3     kVoiceKindDynamics,
4     kVoiceKindHarmonies, // for MusicXML <harmony/>, LilyPond ChordNames
5     kVoiceKindFiguredBass // for MusicXML <figured-bass/>, LilyPond FiguredBass
6 };

```

As stated in the comment above, `fVoiceLastSegment` is used because it because `fVoiceInitialElementsList` can contain any class `msrVoiceElement`, whereas all MSR elements appended to the voice are to be placed in a segment.

An class `msrSegment` instance should thus be created and stored in `fVoiceLastSegment` before class `msrVoiceElement` instances can be appended to the voice.

When repeats are handled, an class `msrRepeat` instance is created. Then the contents of field `msrVoice::fVoiceLastSegment` is moved into it and a new segment is created, see section 19.36 [Repeats], page 192.

Whether the last segment should be created right when the voice is created is controlled with enumeration type `msrVoiceCreateInitialLastSegmentKind`, defined in `src/formats/msr/msrVoices.h`:

```

1 enum class msrVoiceCreateInitialLastSegmentKind {
2     kCreateInitialLastSegmentYes,
3     kCreateInitialLastSegmentNo
4 };

```

## 19.30 Measures

Measures handling is presented at section 40 [Measures handling], page 271.

A measure is a linear, flat sequence of class `msrMeasureElements`, some of which are structured, such as class `msrChord`. Class `msrMeasure` is defined in `src/formats/msr/msrMeasure.h/.cpp`.

The measure elements are defined in `src/formats/msr/`:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'public
  msrMeasureElement' *.h
2 msrBars.h:class EXP msrBarCheck : public msrMeasureElement
3 msrBars.h:class EXP msrBarNumberCheck : public msrMeasureElement
4 msrBars.h:class EXP msrBarLine : public msrMeasureElement
5 msrBreaks.h:class EXP msrLineBreak : public msrMeasureElement
6 msrBreaks.h:class EXP msrPageBreak : public msrMeasureElement
7 msrClefs.h:class EXP msrClef : public msrMeasureElement
8 msrCodas.h:class EXP msrCoda : public msrMeasureElement
9 msrDoubleTremolos.h:class EXP msrDoubleTremolo : public msrMeasureElement
10 msrEyeGlasses.h:class EXP msrEyeGlasses : public msrMeasureElement
11 msrFiguredBasses.h:class EXP msrFiguredBass : public msrMeasureElement
12 msrHarmonies.h:class EXP msrHarmony : public msrMeasureElement
13 msrHiddenMeasureAndBarLines.h:class EXP msrHiddenMeasureAndBarLine : public
  msrMeasureElement
14 msrInstruments.h:class EXP msrScordatura : public msrMeasureElement
15 msrInstruments.h:class EXP msrAccordionRegistration : public msrMeasureElement
16 msrInstruments.h:class EXP msrHarpPedalsTuning : public msrMeasureElement
17 msrInstruments.h:class EXP msrPedal : public msrMeasureElement
18 msrInstruments.h:class EXP msrDamp : public msrMeasureElement
19 msrInstruments.h:class EXP msrDampAll : public msrMeasureElement
20 msrKeys.h:class EXP msrKey : public msrMeasureElement
21 msrLyrics.h:class EXP msrSyllable : public msrMeasureElement
22 msrMusicXMLSpecifics.h:class EXP msrPrintLayout : public msrMeasureElement
23 msrRehearsalMarks.h:class EXP msrRehearsalMark : public msrMeasureElement
24 msrSegnos.h:class EXP msrSegno : public msrMeasureElement
25 msrDalSegnos.h:class EXP msrDalSegno : public msrMeasureElement
26 msrStavesDetails.h:class EXP msrStaffDetails : public msrMeasureElement
27 msrTempos.h:class EXP msrTempo : public msrMeasureElement
28 msrTimeSignatures.h:class EXP msrTimeSignature : public msrMeasureElement
29 msrTranspositions.h:class EXP msrOctaveShift : public msrMeasureElement
30 msrTranspositions.h:class EXP msrTransposition : public msrMeasureElement
31 msrVoiceStaffChanges.h:class EXP msrVoiceStaffChange : public msrMeasureElement

```

In order to perform a time-wise analysis of the scores, MSR contains class `msrmeasure` linear flat lists, without the class `msrRepeat` and such being represented.

This is used when identifying rest notes that are not 'heard' simultaneously with other notes or rests: this way, the rest can ignore the current voice number and be placed in the vertical middle of the staff.

Apart from the cloning methods, only one method creates measures, namely method `msrSegment::createAMeasureAndAppendItToSegment ()`, defined in `src/formats/msr/msrSegments.h/.cpp`:

```

1 S_msrMeasure msrSegment::createAMeasureAndAppendItToSegment (
2   int      inputLineNumber,
3   std::string measureNumber,
4   msrMeasureImplicitKind
5     measureImplicitKind)
6 {
7   // ... ..
8
9   ++gIndenter;
10
11   // determine new measure 'first in segment' kind
12   msrMeasureFirstInSegmentKind
13     measureFirstInSegmentKind;
14
15   if (fSegmentElementsList.size () == 0) {
16     // this is the first measure in the segment
17     measureFirstInSegmentKind =
18       msrMeasureFirstInSegmentKind::kMeasureFirstInSegmentKindYes;
19   }
20   else {
21     // this is not the first measure in the segment

```



```

22     measureFirstInSegmentKind =
23         msrMeasureFirstInSegmentKind::kMeasureFirstInSegmentKindNo;
24 }
25
26 // create a measure
27 // ... ..
28
29 S_msrMeasure
30     result =
31         msrMeasure::create (
32             inputLineNumber,
33             measureNumber,
34             this);
35
36 // set result's ordinal number
37 result->
38     setMeasureOrdinalNumberInVoice (
39         fSegmentUpLinkToVoice->
40             incrementVoiceCurrentMeasureOrdinalNumber ());
41
42 // append result to the segment
43 appendMeasureToSegment (result);
44
45 --gIndenter;
46
47 return result;
48 }

```

## 19.31 Repeats patterns and replicas

MSR represents repeated beats and measures this way:

- a pattern describes what is repeated;
- there are as many replicas of the music as needed.

This leads to:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep Pattern *.h | grep
  class
2 msrBeatRepeats.h:class EXP msrBeatRepeatPattern : public msrElement
3 msrMeasureRepeats.h:class EXP msrMeasureRepeatPattern : public msrElement
4 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep Replicas *.h | grep
  class
5 msrBeatRepeats.h:class EXP msrBeatRepeatReplicas : public msrElement
6 msrMeasureRepeats.h:class EXP msrMeasureRepeatReplicas : public msrElement

```

These two repeat cases are described in the sections below.

## 19.32 Beat repeats

Beat repeats handling is presented at section [45](#) [Beat repeats handling], page [303](#).

Class `msrBeatRepeat`, defined in `src/formats/msr/msrBeatRepeats.h/.cpp`, contains a pattern and replicas:

```

1 class EXP msrBeatRepeat : public msrVoiceElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrVoice          fUpLinkToBeatRepeatToVoice;
12
13        // numbers
14        int                  fBeatRepeatMeasuresNumber;
15        int                  fBeatRepeatSlashesNumber;
16
17        // measures repeat pattern
18        S_msrBeatRepeatPattern
19                               fBeatRepeatPattern;
20
21        // measures repeat replicas
22        S_msrBeatRepeatReplicas
23                               fBeatRepeatReplicas;
24
25        // measures repeat build phase, used when building the measures repeat
26        msrBeatRepeatBuildPhaseKind
27                               fCurrentBeatRepeatBuildPhaseKind; // unused??? JMI
28 };

```

Class `msrBeatRepeatPattern` contains a segment and an uplink:

```

1 class EXP msrBeatRepeatPattern : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrBeatRepeat      fUpLinkToBeatRepeat;
12
13        // segment
14        S_msrSegment         fBeatRepeatPatternSegment;
15 };

```

Class `msrBeatRepeatReplicas` contains a segment and an uplink:

```

1 class EXP msrBeatRepeatReplicas : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrBeatRepeat      fUpLinkToBeatRepeat;
12
13        // segment
14        S_msrSegment         fBeatRepeatReplicasSegment;
15 };

```

## 19.33 Measure repeats

Measure repeats handling is presented at section 46 [Measure repeats handling], page 304.

Class `msrMeasureRepeat`, defined in `src/formats/msr/msrMeasureRepeat.h/.cpp`, contains a pattern and replicas:

```

1 class EXP msrMeasureRepeat : public msrVoiceElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrVoice          fUpLinkToMeasureRepeatToVoice;
12
13        // numbers
14        int                 fMeasureRepeatMeasuresNumber;
15        int                 fMeasureRepeatSlashesNumber;
16
17        // measures repeat pattern
18        S_msrMeasureRepeatPattern
19            fMeasureRepeatPattern;
20
21        // measures repeat replicas
22        S_msrMeasureRepeatReplicas
23            fMeasureRepeatReplicas;
24
25        // measures repeat build phase, used when building the measures repeat
26        msrMeasureRepeatBuildPhaseKind
27            fCurrentMeasureRepeatBuildPhaseKind;
28 };

```

Class `msrMeasureRepeatPattern` contains a segment and an uplink:

```

1 class EXP msrMeasureRepeatPattern : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrMeasureRepeat  fUpLinkToMeasureRepeat;
12
13        // segment
14        S_msrSegment        fMeasureRepeatPatternSegment;
15 };

```

Class `msrMeasureRepeatReplicas` contain a segment and an uplink:

```

1 class EXP msrMeasureRepeatReplicas : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9

```

```

10 // upLinks
11 S_msrMeasureRepeat    fUpLinkToMeasureRepeat;
12
13 // segment
14 S_msrSegment          fMeasureRepeatReplicasSegment;
15 };

```

## 19.34 Multiple full-bar rests

Full-bar rests handling is presented at section 47 [Multiple full-bar rests handling], page 305.

Class `msrMultipleFullBarRests`, defined in `src/formats/msr/msrMultipleFullBarRests.h/.cpp`, essentially contains a list of class `_msrMeasure` instances and a multiple full-bar rests number:

```

1 class EXP msrMultipleFullBarRests : public msrSegmentElement
2 {
3     // ... ..
4
5     private:
6
7     // private fields
8     // -----
9
10    S_msrSegment          fMultipleFullBarRestsUpLinkToSegment;
11
12    int                   fMultipleFullBarRestsNumber; // supplied by MusicXML
13    std::list<S_msrMeasure>
14                          fFullBarRestsMeasuresList;
15
16    int                   fMultipleFullBarRestsLastMeasurePuristNumber;
17
18    std::string           fMultipleFullBarRestsNextMeasureNumber;
19 };

```

## 19.35 Barlines

## 19.36 Repeats

Repeats handling is presented at section 48 [Repeats handling], page 306.

Contrary to MusicXML, MusicFormats represents the full structure of repeated music, not just barlines.

The following classes are defined in `src/formats/msr/msrRepeats.h/.cpp`, contains:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep class    msrRepeats.h
2 class    msrRepeat;
3 class    msrMultipleFullBarRests;
4 class    msrMeasureRepeat;
5 class    msrNote;
6 class EXP msrRepeatCommonPart : public msrElement
7 class EXP msrRepeatEnding : public msrElement
8 class EXP msrRepeat : public msrVoiceElement
9 class EXP msrRepeatDescr : public smartable
10 class EXP msrRepeatElement : public msrElement

```

Class `msrRepeat`, defined in `msrBothmsrRepeats`, contains an class `msrRepeatCommonPart`, followed by zero or more instances of class `msrRepeatEnding`:

```

1 class EXP msrRepeat : public msrVoiceElement
2 {
3     public:
4
5         // data types
6         // -----
7
8         enum class msrRepeatExplicitStartKind {
9             kRepeatExplicitStartNo,
10            kRepeatExplicitStartYes
11        };
12
13        // ... ..
14
15        // common part
16        S_msrRepeatCommonPart fRepeatCommonPart;
17
18        // repeat endings
19        std::vector<S_msrRepeatEnding>
20            fRepeatEndings;
21        int
22            fRepeatEndingsInternalCounter;
23
24        // immediately preceding and following repeats
25        // detecting several repeats in a row helps LilyPond code generation
26        // depending on the options JMI
27        S_msrRepeat
28            fImmediatelyPrecedingRepeat;
29        S_msrRepeat
30            fImmediatelyFollowingRepeat;
31    };

```

Class `msrRepeatCommonPart` contains a list of class `msrVoiceElement`:

```

1 private:
2
3     // private fields
4     // -----
5
6     // upLinks
7     S_msrRepeat
8         fRepeatCommonPartUpLinkToRepeat;
9
10    // elements list
11    std::list<S_msrVoiceElement>
12        fRepeatCommonPartElementsList;

```

Enumeration type `msrRepeatEndingKind` is used to distinguish hooked and hookless repeat endings: hookless when the ending is simply overlined, and hooked when there a vertical line at the end of the ending's overline:

```

1 enum class msrRepeatEndingKind {
2     kRepeatEndingHooked,
3     kRepeatEndingHookless
4 };

```

Class `msrRepeatEnding` contains a list of class `msrVoiceElement` too, as well as a enumeration type `msrRepeatEndingKind` field:

```

1 private:
2
3     // private fields
4     // -----
5
6     // upLinks
7     S_msrRepeat
8         fRepeatEndingUpLinkToRepeat;
9
10    // numbers

```

```

10      std::string          fRepeatEndingNumber; // may be "1, 2"
11      int                 fRepeatEndingInternalNumber; // internally assigned
12
13      // kind
14      msrRepeatEndingKind  fRepeatEndingKind;
15
16      // elements list
17      std::list<S_msrVoiceElement>
18                                     fRepeatEndingElementsList;

```

## 19.37 Segments

Segments handling is presented at section 44 [Segments handling], page 298.

Segment are not explicit in music scores, but they are there alright and we have to represent them in MSR:

- it is a sequence of music elements not containing a repeat. This is equivalent to so-called *basic blocs* in compiler technology, that are linear sequences of instructions without jumps, i.e. there is exactly one entry and one exit.

For example, at figure 19.4 [Three segments in a voice], page 194, there are three segments:

- the first one contains the **c1**, and belongs to a first repeat;
- the second one contains the **d1**, and is a member of the voice;
- the last one contains the **e1** and belongs to a second repeat.



Figure 19.4: Three segments in a voice

## 19.38 Notes and rests

Class **msrNote** is complex class: it handles many variants, but using classes to represent the variants would be too cumbersome. As shown at figure 19.3 [The MSR classes hierarchy], page 182:

- a note can be a standalone (regular) note or rest;
- it can belong to a grace notes group;
- it can belong to chord, which can itself belong to a grace notes group or a tuplet;
- it can belong to a tuplet;
- it can belong to double tremolo;
- and finally, a rest can be unpicked.

class `msrNote` thus uses enumeration type `msrNoteKind`, defined in `src/formats/msr/msrBasicType` to distinguish them:

```

1  enum class msrNoteKind {
2      kNote_UNKNOWN,
3
4      // in measures
5      kNoteRegularInMeasure,
6      kNoteRestInMeasure,
7      kNoteSkipInMeasure, // an invisible rest
8      kNoteUnpitchedInMeasure,
9
10     // in chords
11     kNoteRegularInChord,
12
13     // in triplets
14     kNoteRegularInTriplet,
15     kNoteRestInTriplet,
16     kNoteUnpitchedInTriplet,
17
18     // in grace notes groups
19     kNoteRegularInGraceNotesGroup,
20     kNoteSkipInGraceNotesGroup, // used to circumvent LilyPond issue #34
21
22     // in chords in grace notes groups
23     kNoteInChordInGraceNotesGroup,
24
25     // in triplets in grace notes groups
26     kNoteInTripletInGraceNotesGroup,
27
28     // in double-tremolos
29     kNoteInDoubleTremolo
30 };

```

## 19.39 Grace notes groups

Grace notes groups handling is presented at section 60 [Grace notes groups handling], page 326.

## 19.40 Chords

A chord contains notes only, and can occur in measures, triplets and grace notes groups, hence:

```

1  // chords
2  // -----
3
4  enum class msrChordInKind {
5      kChordIn_UNKNOWN,
6
7      kChordInMeasure,
8      kChordInTriplet,
9      kChordInGraceNotesGroup
10 };

```

## 19.41 Tuples

Tuples handling is presented at section [62](#) [Tuples handling], page [328](#).

A tuple can contain:

- notes and rests;
- chords;
- other tuples.

Tuples can occur in measures and other tuples, hence enumeration type `msrTupletInKind`:

```

1 enum class msrTupletInKind {
2     kTupletIn_UNKNOWN,
3
4     kTupletIn_Measure,
5     kTupletIn_Tuplet
6 };

```

Tuples factors are represented by class `msrTupletFactor`, defined in `src/formats/msr/msrBasicTypes.h/.cpp`.

```

1 class EXP msrTupletFactor
2 {
3     // ... ..
4
5     public:
6
7         // public services
8         // -----
9
10        Bool                isEqualToOne () const
11                                {
12                                    return
13                                        fTupletActualNotes == fTupletNormalNotes;
14                                }
15
16        Rational            asRational () const
17                                {
18                                    return
19                                        Rational (
20                                            fTupletActualNotes,
21                                            fTupletNormalNotes);
22                                }
23
24        // ... ..
25
26        private:
27
28            // private fields
29            // -----
30
31            int              fTupletActualNotes;
32            int              fTupletNormalNotes;
33 };

```



## 19.42 Harmonies and figured bass similarities

Harmonies and figured bass handling is presented at section 63 [Harmonies handling], page 329 and section 64 [Figured bass elements handling], page 345, respectively.

In MusicXML, harmonies and figured bass occur at the measure level:

```

1  <harmony print-frame="no">
2    <root>
3      <root-step>C</root-step>
4    </root>
5    <kind text="m">minor</kind>
6  </harmony>
7  <note default-x="75.17" default-y="-35.00">
8    <pitch>
9      <step>F</step>
10     <octave>4</octave>
11   </pitch>
12   <duration>2</duration>
13   <voice>1</voice>
14   <type>quarter</type>
15   <stem>up</stem>
16 </note>

```

```

1  <harmony>
2    <root>
3      <root-step>F</root-step>
4      <root-alter>1</root-alter>
5    </root>
6    <kind>major</kind>
7    <inversion>2</inversion>
8  </harmony>
9  <note>
10    <pitch>
11      <step>C</step>
12      <octave>4</octave>
13    </pitch>
14    <duration>4</duration>
15    <type>whole</type>
16  </note>

```

In MSR, the instances of class `msrHarmony` and class `msrFiguredBass` are present twice:

- each class `msrNote` instance contains the harmonies and figured bass attached to it:

```

1  class EXP msrNote : public msrTupletElement
2  {
3    // ... ..
4
5    private:
6
7      // private fields
8      // -----
9
10     // harmonies
11     // -----
12
13     std::list<S_msrHarmony>
14                           fNoteHarmoniesList;
15
16     // figured bass
17     // -----
18
19     std::list<S_msrFiguredBass>

```

```

20         fNoteFiguredBassesList;
21
22     // ... ..
23 };

```

- each class `msrPart` instance contains a harmonies staff and voice, as well as a figured bass staff and voice:

```

1  class EXP msrPart : public msrPartGroupElement
2  {
3      // ... ..
4
5      private:
6
7          // private fields
8          // -----
9          // harmonies
10
11         S_msrStaff          fPartHarmoniesStaff;
12         S_msrVoice          fPartHarmoniesVoice;
13
14         // figured bass
15
16         S_msrStaff          fPartFiguredBassStaff;
17         S_msrVoice          fPartFiguredBassVoice;
18
19         // ... ..
20 };

```

The way harmonies and figured bass elements are represented in MusicFormats is presented in the next two sections.

## 19.43 Harmonies

Harmonies handling is presented at section [63](#) [Harmonies handling], page [329](#).

## 19.44 Figured bass

Figured bass elements handling is presented at section [64](#) [Figured bass elements handling], page [345](#).

## 19.45 Lyrics

Lyrics handling is presented at section [65](#) [Lyrics handling], page [360](#).

Lyrics are handled in rather a special way in music scores:

- they have a linear structure, independent of the repeats structure of the staff they belong too;
- the can be several lyrics stanzas associated to a given staff;
- the syllables in lyrics can apply to more that one note, and the subdivisions of words have to be handled.

The basic building block for lyrics in MSR is class `msrSyllable`, whose variants are distinguished by enumeration type enumeration type `msrSyllableKind`:

```

1  enum class msrSyllableKind {
2      kSyllableNone,
3      kSyllableSingle,
4      kSyllableBegin, kSyllableMiddle, kSyllableEnd,
5
6      kSyllableOnRestNote,
7      kSyllableSkipRestNote,
8      kSyllableSkipNonRestNote,
9
10     kSyllableMeasureEnd,
11     kSyllableLineBreak, kSyllablePageBreak
12 };

```

Extensions are described by enumeration type :

```

1  enum class msrSyllableExtendKind {
2      kSyllableExtendNone,
3      kSyllableExtendEmpty,
4      kSyllableExtendSingle,
5      kSyllableExtendStart, kSyllableExtendContinue, kSyllableExtendStop
6  };

```

Class `msrSyllable` contains:

```

1  // syllable kind
2  msrSyllableKind      fSyllableKind;
3
4  // texts list
5  std::list<std::string>
6      fSyllableTextsList;
7
8  // extend kind
9  msrSyllableExtendKind fSyllableExtendKind;
10
11 // stanza number, may contain non-digits
12 std::string          fSyllableStanzaNumber;
13
14 // syllable whole notes
15 Rational             fSyllableWholeNotes;
16
17 // syllable tuplet factor
18 msrTupletFactor      fSyllableTupletFactor;

```

Syllables are one case where the data in MSR is denormalized: a given class `msrSyllable` instance belongs both to an class `msrNote` instance and to a lyrics instance of class `msrVoice`.

At the higher level, syllables are organized as instances of class `msrStanza`, which contains:

```

1  // contents
2  std::vector<S_msrSyllable> fSyllables;
3
4  Bool          fStanzaTextPresent;

```

## 19.46 MIDI

MIDI handling is presented at section [66](#) [MIDI handling], page [361](#).

## Chapter 20

# MSR time-oriented representation

In order to represent the music according to simultaneous sounding time, MSR builds:

- a flat list of measures at the voice and staff levels;
- from this, a vector of measures slices at the voice, staff, part, part group and score levels.

The source files are in `src/formats/msr/msrMeasuresSlices.h/.cpp`.

### 20.1 Note events

Notes start and stop are represented by enumeration type `msrNoteEventKind`:

```
1 enum class msrNoteEventKind {  
2     kNoteEventStart,  
3     kNoteEventStop  
4 };
```

A note event is described in class :

```
1 class msrNoteEvent : public smartable  
2 {  
3     // ...  
4  
5     private:  
6  
7         // private fields  
8         // -----  
9  
10        Rational          fNoteEventMeasurePosition;  
11        S_msrNote          fNoteEventNote;  
12        msrNoteEventKind   fNoteEventKind;  
13 };
```

## 20.2 Simultaneous notes chunks

Such a chunk is a set of notes or rests played simultaneously, i.e. that start and stop at the same time. The set is stored as a list actually:

```

1 class    msrSimultaneousNotesChunk : public smartable
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        Rational                fChunkMeasurePosition;
11        std::list<S_msrNote>    fChunkNotesList;
12        Rational                fChunkDurationWholeNotes;
13 };

```

## 20.3 Measures slices

A measures slice, described by class `msrMeasuresSlice`, is a 'vertical' cut in the score across voices: it contains all the measures starting at the same time, one per voice:

```

1 class EXP msrMeasuresSlice : public smartable
2 {
3     // ... ..
4
5     protected:
6
7         // protected fields
8         // -----
9
10        int                    fSlicePuristMeasureNumber;
11        std::string            fSliceMeasureNumber;
12
13        // the measures in the slice
14        std::vector<S_msrMeasure> fSliceMeasuresVector;
15
16        // notes flat list
17        std::list<S_msrNote>    fSliceNotesFlatList;
18
19        // note events list
20        std::list<S_msrNoteEvent>
21                                fSliceNoteEventsList;
22
23        // simultaneous notes chunks list
24        std::list<S_msrSimultaneousNotesChunk>
25                                fSliceSimultaneousNotesChunksList;
26 };

```

## 20.4 Measures slices sequences

A class `msrMeasuresSlicesSequence` contains a vector of `S_msrMeasuresSlice` instances:

```

1 class EXP msrMeasuresSlicesSequence : public smartable
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        std::string          fMeasuresOrigin;
11
12        std::vector<S_msrMeasuresSlice>
13                                fMeasuresSlicesVector;
14 };

```

A smart pointer to an `msrMeasuresSlicesSequence` instance is stored in `msrVoice`, `msrStaff`, `msrPart`, `msrPartGroup` and `msrScore`.

## 20.5 Building the measures slices

### 20.5.1 Part measures slices

At the part level, this is done in method `msrPart::collectPartMeasuresSlices ()`:

```

1 void msrPart::collectPartMeasuresSlices (
2     int inputLineNumber)
3 {
4     // ... ..
5
6     // create the part measures slices sequence
7     fPartMeasuresSlicesSequence =
8         msrMeasuresSlicesSequence::create (
9             fPartName); // origin
10
11    // populate it
12    for (S_msrStaff staff : fPartAllStavesList) {
13        // ... ..
14
15        ++gIndenter;
16
17        S_msrMeasuresSlicesSequence
18            staffMeasuresSlicesSequence =
19                staff->
20                getStaffMeasuresSlicesSequence ();
21
22        if (! staffMeasuresSlicesSequence) {
23            std::stringstream s;
24
25            s <<
26                "The staffMeasuresSlicesSequence of staff \"" <<
27                staff->getStaffName () <<
28                "\" is null";
29
30            musicxmlWarning (
31                gGlobalServiceRunData->getInputSourceName (),
32                inputLineNumber,
33                s.str ());
34        }
35        else {
36            fPartMeasuresSlicesSequence->
37                mergeWithMeasuresSlicesSequence (
38                    inputLineNumber,

```

```

39         getPartCombinedName (),
40         staffMeasuresSlicesSequence);
41     }
42
43     --gIndenter;
44 } // for
45
46 // ... ..
47 }

```

### 20.5.2 Staff measures slices

Method `msrStaff::collectStaffMeasuresSlices ()` builds them:

```

1 void msrStaff::collectStaffMeasuresSlices (
2     int inputLineNumber)
3 {
4     // ... ..
5
6     // create the staff measures slices sequence
7     fStaffMeasuresSlicesSequence =
8         msrMeasuresSlicesSequence::create (
9             fStaffName); // origin
10
11    // populate it
12    for (const S_msrVoice& voice : fStaffAllVoicesList) {
13        // ... ..
14
15        // get the voice measures slices sequence
16        S_msrMeasuresSlicesSequence
17            voiceMeasuresSlicesSequence =
18            voice->
19                getVoiceMeasuresSlicesSequence ();
20
21        // merge it with the voice measures slices sequence
22        if (voiceMeasuresSlicesSequence) { // JMI
23            fStaffMeasuresSlicesSequence =
24                fStaffMeasuresSlicesSequence->
25                    mergeWithMeasuresSlicesSequence (
26                        inputLineNumber,
27                        fStaffName,
28                        voiceMeasuresSlicesSequence);
29        }
30
31        // identify the solo notes and rests in the staff
32        fStaffMeasuresSlicesSequence->
33            identifySoloNotesAndRests ();
34
35        --gIndenter;
36    } // for
37
38    // ... ..
39 }

```

## 20.6 Solo notes and rests

A solo note or rest is one that occurs alone at some point in time for its whole duration, without any other note being played at the same time.

Identifying such solo notes or rests is done in method `msrMeasuresSlicesSequence::identifySoloNotesAndRests` using the measures slices of the staff they occur in, called method `msrStaff::collectStaffMeasuresSlices` () as shown above:

```
1 void msrMeasuresSlicesSequence::identifySoloNotesAndRests ()
2 {
3     // ... ..
4
5     // collect the notes from the sequence's measures slices
6     for (
7         std::vector<S_msrMeasuresSlice>::const_iterator i =
8             fMeasuresSlicesVector.begin ();
9         i != fMeasuresSlicesVector.end ();
10        ++i
11    ) {
12        S_msrMeasuresSlice measuresSlice = (*i);
13
14        measuresSlice->
15            collectNonSkipNotesFromMeasuresSliceMeasures ();
16    } // for
17 }
```

## 20.7 A measures slices example



## Chapter 21

# Path to voice

`src/formats/msr/msrPathToVoice.h.h/.cpp` defines class `msrPathToVoice`, used to create partial clones of class `msrBook` retaining only certain staves and/or voices, or to create new class `msrScore` instances containing each of them only:

```
1 class EXP msrPathToVoice : public smartable
2 {
3     // ... ..
4
5     public:
6
7         // public services
8         // -----
9
10        void                appendPartGroup (const S_msrPartGroup& partGroup)
11                                {
12                                    fPartGroupsList.push_back (partGroup);
13                                }
14
15        // ... ..
16
17        private:
18
19            // private fields
20            // -----
21
22            S_msrBook          fBook;
23
24            S_msrScore          fScore;
25
26            // part groups can be nested
27            std::list<S_msrPartGroup>  fPartGroupsList;
28
29            S_msrPart          fPart;
30
31            S_msrStaff          fStaff;
32
33            S_msrVoice          fVoice;
34 };
```

## Chapter 22

# LilyPond Scores Representation (LPSR)

An LPSR description contains two components:

- the first one is an MSR, thus the whole music score description;
- the second one is a description of the structure of the score mirroring LilyPond's specific blocks such as `\book {...}` and `\layout {...}`.

Class `lpsrScore` thus contains:

```
1 // MSR data
2 S_msrScore          fMsrScore;
3
4 // ... ..
5
6 // LilyPond stuff
7 S_lpsrHeader        fScoreHeader;
8 S_lpsrPaper         fScorePaper;
9 S_lpsrLayout        fScoreLayout;
10
11 // variables, voices and stanzas
12 std::list<S_msrElement>
13                     fScoreElementsList;
14
15 // score LPSR book blocks list
16 std::list<S_lpsrBookBlock> fScoreBookBlocksList;
17 S_lpsrScoreBlock      fScoreScoreBlock; // JMI ???
```

### 22.1 LPSR basic types

Some types used throughout LSPR are defined in `src/formats/lpsr/lpsrEnumTypes.h/.cpp`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/lpsr > egrep -rIn '^// '
  lpsrEnumTypes.h
2 lpsrEnumTypes.h:28:// score output kinds
3 lpsrEnumTypes.h:50:// accidental styles
4 lpsrEnumTypes.h:53:// JMI there are new ones in LilyPond 2.22
5 lpsrEnumTypes.h:87:// chords languages
6 lpsrEnumTypes.h:107:// whole notes
7 lpsrEnumTypes.h:118:// dotted durations
8 lpsrEnumTypes.h:129:// rests measures
9 lpsrEnumTypes.h:135:// texts lists
10 lpsrEnumTypes.h:141:// pitches and octaves
11 lpsrEnumTypes.h:151:// lyrics durations
12 lpsrEnumTypes.h:168:// initialization
```

## 22.2 Adapting LilyPond code generation to the target version number

As of version 2.22, `compressMultipleFullBarRests` has been replaced by `compressFullBarRests` for clarity.

Such is done specific methods:



## Chapter 23

# Braille Scores Representation (BSR)

BSR represents braille scores as composed of lines of 6-dot cells.

### 23.1 BSR basic types

Some types used throughout BSR are defined in [src/formats/bsr/bsrEnumTypes.h/.cpp](#):

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/bsr > egrep -rIn '~/' '
   bsrEnumTypes.h
2 bsrEnumTypes.h:23:// cell kinds
3 bsrEnumTypes.h:107:// lower case letters
4 bsrEnumTypes.h:139:// capitals
5 bsrEnumTypes.h:143:// kCellCapitalsSequenceSign, // { kCellCapitalsSign,
   kCellCapitalsSign };
6 bsrEnumTypes.h:145:// decimal digits
7 bsrEnumTypes.h:160:// lower decimal digits
8 bsrEnumTypes.h:174:// alterations
9 bsrEnumTypes.h:181:// augmentation dots
10 bsrEnumTypes.h:186:// arithmetic operators
11 bsrEnumTypes.h:195:// words
12 bsrEnumTypes.h:205:// braille cells
13 bsrEnumTypes.h:212:// braille output kinds
14 bsrEnumTypes.h:231:// chords languages
15 bsrEnumTypes.h:251:// brailling numbers
16 bsrEnumTypes.h:255:// brailling characters and strings
17 bsrEnumTypes.h:261:// writing UTF-16 to ostream
18 bsrEnumTypes.h:273:// initialization
19 bsrEnumTypes.h:971:// constants
20 bsrEnumTypes.h:975:// computations
```

### 23.2 Representing cells

This is done basically with enumeration type enumeration type `bsrCellKind`, defined in [src/formats/bsr/bsrEnumTy](#)

```
1 // cell kinds
2 // -----
3 enum class bsrCellKind {
4     kCellUnknown,
5
6     // non 6dots values
7     kCellEOL      , // L'\u000a'
8     kCellEOP      , // L'\u000c'
9
10    // 6dots values for Braille
```

```

11 kDotsNone      , // L'\u2800'
12 kDots1         , // L'\u2801'
13 kDots2         , // L'\u2802'
14
15 // ... ..
16
17 kDots23456      , // L'\u283e'
18 kDots123456     , // L'\u283f'
19 };

```

Enumeration type `bsrCellKind` constants are not used throughout the code base: instead, there are enumeration type `bsrCellKind` constants to provide context-specific names for the cells kinds.

Lower-case letters:

```

1 // lower case letters
2 // -----
3 const bsrCellKind
4     kCellA = bsrCellKind::kDots1,
5     kCellB = bsrCellKind::kDots12,
6
7
8     kCellY = bsrCellKind::kDots13456,
9     kCellZ = bsrCellKind::kDots1356;

```

Capital sign:

```

1 // capitals
2 // -----
3 const bsrCellKind
4     kCellCapitalsSign = bsrCellKind::kDots46;

```

Decimal digits:

```

1 // decimal digits
2 // -----
3 const bsrCellKind
4     kCellNumberSign = bsrCellKind::kDots3456,
5     kCell11 = kCellA,
6     kCell12 = kCellB,
7     kCell13 = kCellC,
8     kCell14 = kCellD,
9     kCell15 = kCellE,
10    kCell16 = kCellF,
11    kCell17 = kCellG,
12    kCell18 = kCellH,
13    kCell19 = kCellI,
14    kCell10 = kCellJ;

```

Alterations:

```

1 // alterations
2 // -----
3 const bsrCellKind
4     kCellFlat      = bsrCellKind::kDots126,
5     kCellNatural   = bsrCellKind::kDots16,
6     kCellSharp     = bsrCellKind::kDots146;

```

Augmentation dots:

```

1 // augmentation dots
2 // -----
3 const bsrCellKind
4     kCellAugmentationDot = bsrCellKind::kDots3;

```

Arithmetic operators:

```
1 // arithmetic operators
2 // -----
3 const bsrCellKind
4     kCell_ac_plus      = bsrCellKind::kDots235,
5     kCell_ac_minus     = bsrCellKind::kDots36,
6     kCell_ac_times     = bsrCellKind::kDots35,
7     kCell_ac_dividedBy = bsrCellKind::kDots25,
8     kCell_ac_equals    = bsrCellKind::kDots2356;
```

Words:

```
1 // words
2 // -----
3 const bsrCellKind
4     kCellWordSign      = bsrCellKind::kDots345,
5
6     kCellWordApostrophe = bsrCellKind::kDots6,
7
8     kCellParenthesis    = bsrCellKind::kDots2356,
9     kCellQuestionMark   = bsrCellKind::kDots26;
```

## Chapter 24

# MusicXML Scores Representation (MXSR)

This format is provided by `libmusicxml2`, even though Dominique Fober didn't give it that name. It is a tree of class `mxmlelement` nodes, mapped one to one to the MusicXML markups.

The files in `libmusicxml/src`.

A set of interface functions is contained in `src/formats/mxsr/mxsr.h/.cpp`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/mxsr/  
2 total 48  
3 0 drwxr-xr-x  6 jacquesmenu staff  192 May 26 08:20:55 2021 ./  
4 0 drwxr-xr-x 10 jacquesmenu staff  320 Jun 25 05:39:49 2021 ../  
5 8 -rw-r--r--@ 1 jacquesmenu staff 3292 Jun  6 06:35:19 2021 mxsr.cpp  
6 8 -rw-r--r--@ 1 jacquesmenu staff 1555 Jun  6 06:35:19 2021 mxsrGeneration.h  
7 16 -rw-r--r--@ 1 jacquesmenu staff 7781 Jun  6 06:35:19 2021 mxsr0ah.cpp  
8 16 -rw-r--r--@ 1 jacquesmenu staff 4829 Jun  6 06:35:19 2021 mxsr0ah.h
```

## 24.1 MusicXML elements and attributes

MusicXML data contains so-called elements, written as `<... />` markups, that can be nested:

```
1 <system-margins>  
2   <left-margin>15</left-margin>  
3   <right-margin>0</right-margin>  
4 </system-margins>
```

In the example above, the values of the two margins are 15 and 0, respectively.

MusicXML elements can have attributes, such as `version` below:

```
1 <score-partwise version="3.1">
```

The values of the elements and attributes are strings.

There are two special elements at the beginning of MusicXML data:

- a `<?xml/>` element indicating the characters encoding used;
- a `<“!”DOCTYPE/>` element telling that the contents is in 'score-partwise' mode and containing the URL of the DTD.

An exemple is:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 2.0 Partwise//EN"
3 "http://www.musicxml.org/dtds/partwise.dtd">

```

## 24.2 The xmlelement and xmlattribute types

xmlelementxmlattribute

These two classes are defined in `libmusicxml/src/elements!xml.h/.cpp`:

```

1 class xmlelement;
2 class xmlattribute;
3
4 typedef SMARTP<xmlattribute> Sxmlattribute;
5 typedef SMARTP<xmlelement> Sxmlelement;

```

class xmlattribute contains:

```

1 //-----
2 class EXP xmlattribute : public smartable {
3     //! the attribute name
4     std::string fName;
5     //! the attribute value
6     std::string fValue;
7
8     // ... ..
9
10    //! returns the attribute value as a int
11    operator int () const;
12    //! returns the attribute value as a long
13    operator long () const;
14    //! returns the attribute value as a float
15    operator float () const;

```

class xmlelement contains:

```

1 class EXP xmlelement : public ctree<xmlelement>, public visitable
2 {
3     private:
4         //! the element name
5         std::string fName;
6         //! the element value
7         std::string fValue;
8         //! list of the element attributes
9         std::vector<Sxmlattribute> fAttributes;
10
11     protected:
12         // the element type
13         int fType;
14         // the input line number for messages to the user
15         int fInputLineNumber;
16
17         // ... ..
18
19         //! returns the element value as a long
20         operator long () const;
21         //! returns the element value as a int
22         operator int () const;
23         //! returns the element value as a float
24         operator float () const;
25         //! elements comparison

```



```

26 Bool operator ==(const xmlelement& elt) const;
27 Bool operator !=(const xmlelement& elt) const { return !(*this == elt); }
28
29 /// adds an attribute to the element
30 long add (const Sxmlattribute& attr);
31
32 // ... ..
33 };

```

Type Sxmlelement is a smart pointer to an xmlelement, so it is an xmlelement tree, since xmlelement is a recursive type.

fInputLineNumber is used for example in warning and error messages, to help the user locate the problem.

fType typically contains a value of some enumeration type , more on this below.

## 24.3 Enumeration types for xmlelement's fType

xmlelement

libmusicxml2 uses elements/templates/elements.bash, a Bash script, to generate the enumeration type constants and classes source code from the MusicXML DTD. This is not done in the Makefile, since it is to be run by hand only once.

The DTD files we use as reference are in libmusicxml/dtds/3.1/schema;

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/libmusicxml/dtds/3.1/schema > ls -sal *.mod
2 40 -rwxr-xr-x@ 1 jacquesmenu staff 20238 Apr 22 15:49 attributes.mod
3 16 -rwxr-xr-x 1 jacquesmenu staff 4943 Apr 22 15:49 barLine.mod
4 80 -rwxr-xr-x@ 1 jacquesmenu staff 37932 Apr 22 15:49 common.mod
5 88 -rwxr-xr-x@ 1 jacquesmenu staff 41960 Apr 22 15:49 direction.mod
6 16 -rwxr-xr-x@ 1 jacquesmenu staff 4097 Apr 22 15:49 identity.mod
7 24 -rwxr-xr-x@ 1 jacquesmenu staff 10266 Apr 22 15:49 layout.mod
8 8 -rwxr-xr-x@ 1 jacquesmenu staff 2833 Apr 22 15:49 link.mod
9 104 -rwxr-xr-x@ 1 jacquesmenu staff 51384 Apr 22 15:49 note.mod
10 32 -rwxr-xr-x@ 1 jacquesmenu staff 15476 Apr 22 15:49 score.mod

```

The first result of running libmusicxml/src/elements/templates!elements.bash is an anonymous enumeration type defined in libmusicxml/src/elements!elements.h:

```

1 enum class {
2     kNoElement,
3     kComment,
4     kProcessingInstruction,
5     k_accent,
6     k_accidental,
7     k_accidental_mark,
8     k_accidental_text,
9
10    // ... ..
11
12     k_work,
13     k_work_number,
14     k_work_title,
15     kEndElement
16 };

```

The constants kNoElement, kComment and kProcessingInstruction are added by elements.bash.

## 24.4 Classes for the xmlelements

All the MusicXML classes are instantiated from the `musicxml` template class, defined in `libmusicxml/src/elements!`. This is where `fType` gets its value:

```
1 template <int elt> class musicxml : public xmlelement
2 {
3     protected:
4         musicxml (int inputLineNumber) : xmlelement (inputLineNumber) { fType = elt; }
5 };
```

The smart pointer `s` to the various elements are defined in `libmusicxml/src/elements!typedef.h`, using an anonymous enumeration type :

```
1 typedef SMARTP<musicxml<kComment> > S_comment;
2 typedef SMARTP<musicxml<kProcessingInstruction> > S_processing_instruction;
3
4 typedef SMARTP<musicxml<k_accent> > S_accent;
5 typedef SMARTP<musicxml<k_accidental> > S_accidental;
6 typedef SMARTP<musicxml<k_accidental_mark> > S_accidental_mark;
7 typedef SMARTP<musicxml<k_accidental_text> > S_accidental_text;
8
9 // ... ..
10
11 typedef SMARTP<musicxml<k_work> > S_work;
12 typedef SMARTP<musicxml<k_work_number> > S_work_number;
13 typedef SMARTP<musicxml<k_work_title> > S_work_title;
```

The two-way correspondance of MusicXML elements names to type `Sxmlelement` is stored `fMap` and `fType2Name`, defined in `libmusicxml/src/elements!factory.h`:

```
1 class EXP factory : public singleton<factory>{
2
3     std::map<std::string, functor<Sxmlelement>*> fMap;
4     std::map<int, const char*> fType2Name;
5
6     // ... ..
7 };
```

Those two maps are initialized in `libmusicxml/samples/elements/factory.cpp`:

```
1 factory::factory()
2 {
3     fMap["comment"] = new newElementFunctor<kComment>;
4     fMap["pi"] = new newElementFunctor<kProcessingInstruction>;
5     fType2Name[kComment] = "comment";
6     fType2Name[kProcessingInstruction] = "pi";
7
8     fMap["accent"] = new newElementFunctor<k_accent>;
9     fMap["accidental"] = new newElementFunctor<k_accidental>;
10    fMap["accidental-mark"] = new newElementFunctor<k_accidental_mark>;
11    fMap["accidental-text"] = new newElementFunctor<k_accidental_text>;
12
13    // ... ..
14
15    fMap["work"] = new newElementFunctor<k_work>;
16    fMap["work-number"] = new newElementFunctor<k_work_number>;
17    fMap["work-title"] = new newElementFunctor<k_work_title>;
18
19    fType2Name[k_accent] = "accent";
20    fType2Name[k_accidental] = "accidental";
21    fType2Name[k_accidental_mark] = "accidental-mark";
22    fType2Name[k_accidental_text] = "accidental-text";
23 }
```

```

24 // ... ..
25
26 fType2Name[k_work]   = "work";
27 fType2Name[k_work_number] = "work-number";
28 fType2Name[k_work_title]  = "work-title";
29 }

```

class `newElementFunctor` is defined in `to` to provide call operator as:

```

1 template<int elt>
2 class newElementFunctor : public functor<Sxmlelement>
3 {
4     public:
5
6     Sxmlelement operator ()()
7     { return musicxml<elt>::new_musicxml (libmxmllineneno); }
8 };

```

## 24.5 xmlelement trees

This section describes features supplied by `libmusicxml2`.

An `xmlelement` is the basic brick to represent a MusicXML element.

Smart pointer type `SXMLFile` is defined in `libmusicxml/src/xmlfile.h`:

```

1 // -----
2 class EXP TXMLFile : public smartable
3 {
4     private:
5         TXMLDecl*      fXMLDecl;
6         TDocType*      fDocType;
7         Sxmlelement    fXMLTree;
8
9     protected:
10         TXMLFile () : fXMLDecl(0), fDocType(0) {}
11         virtual ~TXMLFile () { delete fXMLDecl; delete fDocType; }
12
13     public:
14         static SMARTP<TXMLFile> create();
15
16     public:
17         TXMLDecl*      getXMLDecl ()      { return fXMLDecl; }
18         TDocType*      getDocType ()      { return fDocType; }
19         Sxmlelement    elements ()        { return fXMLTree; }
20
21         void           set (Sxmlelement root) { fXMLTree = root; }
22         void           set (TXMLDecl * dec)  { fXMLDecl = dec; }
23         void           set (TDocType * dt)   { fDocType = dt; }
24
25         void           print (std::ostream& s);
26 };
27 typedef SMARTP<TXMLFile> SXMLFile;

```

### 24.5.1 Creating xmlelement trees from textual data

Reading MusicXML data creates instances of `xmlelement`. This is done by an instance of `xmlreader`, defined in `libmusicxml/src/xmlreader.h/.cpp`, which provides methods:

```
1 SXMLFile readbuff(const char* file);
2 SXMLFile read(const char* file);
3 SXMLFile read(FILE* file);
```

These three functions are defined this way:

```
1 // -----
2 SXMLFile xmlreader::readbuff(const char* buffer)
3 {
4     fFile = TXMLFile::create();
5     debug("read buffer", '-');
6     return readbuffer (buffer, this) ? fFile : 0;
7 }
8
9 // -----
10 SXMLFile xmlreader::read(const char* file)
11 {
12     fFile = TXMLFile::create();
13     debug("read", file);
14     return readfile (file, this) ? fFile : 0;
15 }
16
17 // -----
18 SXMLFile xmlreader::read(FILE* file)
19 {
20     fFile = TXMLFile::create();
21     return readstream (file, this) ? fFile : 0;
22 }
```

### 24.5.2 Printing xmlelement trees

An `xmlelement` can be printed by function `printMxsr ()`, defined in `src/formats/mxsr/mxsr.h/.cpp`:

```
1 void printMxsr (const Sxmlelement theMxsr, std::ostream& os)
2 {
3     xmlvisitor v (os);
4     tree_browser<xmlelement> browser (&v);
5     browser.browse (*theMxsr);
6 }
```

This how MusicXML and Guido output are generated.

## 24.6 The SXMLFile type

`SXMLFile` is defined in `libmusicxml/src/factory!musicxmlfactory.h` as a smart pointer to class `TXMLFile`:

```
1 // -----
2 class EXP TXMLFile : public smartable
3 {
4     private:
5         TXMLDecl*          fXMLDecl;
6         TDocType*          fDocType;
7         Sxmlelement       fXMLTree;
8
9     protected:
```

```

10     TXMLFile () : fXMLDecl(0), fDocType(0) {}
11     virtual ~TXMLFile () { delete fXMLDecl; delete fDocType; }
12
13     public:
14         static SMARTP<TXMLFile> create();
15
16     public:
17         TXMLDecl*      getXMLDecl ()      { return fXMLDecl; }
18         TDocType*      getDocType ()      { return fDocType; }
19         Sxmlelement    elements ()        { return fXMLTree; }
20
21         void           set (Sxmlelement root) { fXMLTree = root; }
22         void           set (TXMLDecl * dec)  { fXMLDecl = dec; }
23         void           set (TDocType * dt)   { fDocType = dt; }
24
25         void           print (std::ostream& s);
26 };
27 typedef SMARTP<TXMLFile> SXMLFile;

```

fXMLDecl describes the <?xml/> element and fDocType contains the <"!DOCTYPE"/> element.

---

## Part VII

# Passes

# Chapter 25

## The passes

A pass performs a single translation from one music score description into another, such as from MusicXML to an MXSR, or from an MXSR to an MSR. The name 'pass' comes from the compiler writing field.

### 25.1 Translating MusicXML data to an MXSR format

This is supplied by the `libmusicxml2` library, a version of which is distributed as part of MusicFormats to avoid the need of two installs and the potential associated problems.

#### 25.1.1 MusicXML coverage

`src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.h/.cpp` and `src/passes/mxsr2msr/mxsr2msrTranslator.h/.cpp` handle many of the MusicXML version 3.1 elements. Some of them are handled by both sub-passes, such as `S_voice`, `S_measure` and `S_harmony`.

Among the elements that MusicFormats does not handle are the ones for which there is no occurrence in the corpus in folder `files/musicxml`, such as `beat-unit-tied` and `metronome-tied`.

The elements that are new in MusicXML version 4.0 are not known nor handled yet.

### 25.2 Translating an MXSR to an MSR

This is done by class `mxsr2msrTranslator`.

### 25.3 Translating an MSR to an MXSR

### 25.4 Translating an MSR to another MSR

Such translation is meant to offer an opportunity to modify the score's description depending on options.

### 25.5 Translating an MSR to an LPSR

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

## 25.6 Translating an LPSR to LilyPond code

There are two visiting trace options for the generation of LilyPond code, one for its MSR component, and the other one for its LPSR own part:

```
1 // %%JMI      Bool      fGenerateMsrvVisitingInformation;  
2      Bool      fGenerateLpsrVisitingInformation;
```

## 25.7 Translating an MSR to an BSR

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

## 25.8 Translating a BSR to another BSR

## 25.9 Translating an MXSR to Guido



## Chapter 26

# LilyPond code generation

LilyPond code is produced on standard output, unless options option `-output-file-name`, `-o` or option `-auto-output-file-name`, `-aofn` are used.

### 26.1 Basic principle

Lilypond generation is done in `src/passes/lpsr2lilypond/lpsr2lilypondTranslator.h/.cpp`.

Class `lpsr2lilypondTranslator` provides:

```
1 public:
2
3         lpsr2lilypondTranslator (
4             const S_msrOahGroup& msrOpts,
5             const S_lpsrOahGroup& lpsrOpts,
6             std::ostream& lilypondCodeStream);
7
8     virtual ~lpsr2lilypondTranslator ();
9
10    void translateLpsrToLilypondCode (
11        const S_lpsrScore& lpsrScore);
```

It contains these fields among others:

```
1 private:
2
3     // options
4     // -----
5
6     S_msrOahGroup      fMsrOahGroup;
7     S_lpsrOahGroup     fLpsrOahGroup;
8
9     // the LPSR score we're visiting
10    // -----
11    S_lpsrScore         fVisitedLpsrScore;
12
13    // the output stream
14    // -----
15
16    std::ostream&       fLilypondCodeStream;
```

## 26.2 Generating Scheme functions in the LilyPond output

xml2ly can generate Scheme code that is used by the LilyPond code it generates. This is described in class `lpsrScore` by a number of `*IsNeeded` fields, such as:

```
1 // files includes
2 Bool                fJianpuFileIncludeIsNeeded;
3
4 // Scheme modules
5 Bool                fScmAndAccregSchemeModulesAreNeeded;
6
7 // Scheme functions
8 Bool                fTongueSchemeFunctionIsNeeded;
```

## Chapter 27

# Braille generation

Braille is written to standard output or to files as binary data. Our reference is [http://www.brailleauthority.org/music/Music\\_Braille\\_Code\\_2015.pdf](http://www.brailleauthority.org/music/Music_Braille_Code_2015.pdf).

### 27.1 Basic principle

Lilypond generation is done in `src/passes/bsr2braille/bsr2brailleTranslator.h/.cpp`.

Class `bsr2brailleTranslator` provides:

```
1 public:
2
3         bsr2brailleTranslator (
4             const S_bsrScore&      bsrScore,
5             const S_bsr0ahGroup& bsrOpts,
6             std::ostream&          brailleOutputStream);
7
8     virtual          ~bsr2brailleTranslator ();
9
10    void              translateBsrToBraille ();
```

It contains these fields among others:

```
1 private:
2
3     // options
4     // -----
5
6     S_bsr0ahGroup      fBsr0ahGroup;
7
8     // the BSR score we're visiting
9     // -----
10
11    S_bsrScore          fVisitedBsrScore;
12
13    // the braille generator used
14    // -----
15
16    S_bsrBrailleGenerator fBrailleGenerator;
17
18    // the output stream
19    // -----
20
21    std::ostream&        fBrailleOutputStream;
```

## 27.2 Output files name and contents options

he contents options use the following enumeration types:

```

1 enum class bsrUTFKind {
2     kUTF8, kUTF16
3 };
4
5 enum class bsrByteOrderingKind {
6     kByteOrderingNone,
7     kByteOrderingBigEndian, kByteOrderingSmallEndian
8 };

```

xml2brl supplies a option -files options subgroup:

```

1 jacquesmenu@macmini > xml2brl -query files
2 --- Help for subgroup "files" in group "Files group" ---
3 Files group (-files-group):
4 -----
5 Files (-files):
6     -o, -output-file-name FILENAME
7         Write Braille to file FILENAME instead of standard output.
8     -aofn, -auto-output-file-name
9         This option can only be used when reading from a file.
10        Write MusicXML code to a file in the current working directory.
11        The file name is derived from that of the input file,
12        replacing any suffix after the '.' by 'xml'
13        or adding '.xml' if none is present.
14     -bok, -braille-output-kind OUTPUT_KIND
15        Use OUTPUT_KIND to write the generated Braille to the output.
16        The 4 output kinds available are:
17        ascii, utf16, utf8 and utf8d.
18        'utf8d' leads to every line in the braille score to be generated
19        as a line of cells followed by a line of text showing the contents
20        for debug purposes.
21        The default is 'ascii'.
22     -ueifn, -use-encoding-in-file-name
23        Append a description of the encoding used
24        and the presence of a BOM if any to the file name before the '.'.
25     -bom, -byte-ordering-mark BOM_ENDIAN
26        Generate an initial BOM_ENDIAN byte ordering mark (BOM)
27        ahead of the Braille nusic code,
28        which can be one of 'big' or 'small'.
29        By default, a big endian BOM is generated.

```

## 27.3 Braille generators

The following classes are defined in [src/formatsgeneration/brailleGeneration/brailleGeneration.h/.cpp](#)

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formatsgeneration/brailleGeneration > grep
2     class    brailleGeneration.h
3 enum class bsrUTFKind {
4 enum class bsrByteOrderingKind {
5 class EXP bsrBrailleGenerator : public smartable
6 /* this class is purely virtual
7 class EXP bsrAsciiBrailleGenerator : public bsrBrailleGenerator
8 class EXP bsrUTF8BrailleGenerator : public bsrBrailleGenerator
9 class EXP bsrUTF8DebugBrailleGenerator : public bsrUTF8BrailleGenerator
10 class EXP bsrUTF16BigEndianBrailleGenerator : public bsrBrailleGenerator
11 class EXP bsrUTF16SmallEndianBrailleGenerator : public bsrBrailleGenerator

```

The base class bsrBrailleGenerator contains:

```

1 public:
2
3     // public services
4     // -----
5
6     virtual void          generateCodeForBrailleCell (
7                             bsrCellKind cellKind) = 0;
8
9     void                generateCodeForCellsList (
10                            const S_bsrCellsList& cellsList);
11
12     virtual void          generateCodeForMusicHeading (
13                             const S_bsrMusicHeading& musicHeading);
14
15     virtual void          generateCodeForLineContents (
16                             const S_bsrLineContents& lineContents);
17
18     // ... ..
19
20 protected:
21
22     // protected fields
23     // -----
24
25     std::ostream&          fBrailleOutputStream;

```

## 27.4 Writing braille cells

Braille cells are output to an `std::ostream` as hexadecimal strings by virtual method `generateCodeForBrailleCell` methods in `src/passes/bsr2braille/brailleGeneration.h`, depending on the kind of output chosen.

For example, ASCII braille generation is done by:

```

1 void bsrAsciiBrailleGenerator::generateCodeForBrailleCell (
2     bsrCellKind cellKind)
3 {
4     std::string stringForCell;
5
6     switch (cellKind) {
7     case bsrCellKind::kCellUnknown:
8         {
9             std::stringstream s;
10
11             s <<
12                 "cannot generate code for braille cell '" <<
13                 bsrCellKindAsString (cellKind) <<
14                 "'";
15             msrInternalError (
16                 gGlobalServiceRunData->getInputSourceName (),
17                 -999, // inputLineNumber, TICINO JMI
18                 __FILE__, __LINE__,
19                 s.str ());
20         }
21         break;
22
23     case bsrCellKind::kCellEOL:      stringForCell = "\x0a"; break;
24     case bsrCellKind::kCellEOP:      stringForCell = "\x0c"; break;
25
26     case bsrCellKind::kDotsNone:     stringForCell = "\x20"; break;
27
28     case bsrCellKind::kDots1:        stringForCell = "\x41"; break;
29     case bsrCellKind::kDots2:        stringForCell = "\x31"; break;
30

```

```
31      // ... ..
32
33      case bsrCellKind::kDots23456:  stringForCell = "\x29"; break;
34      case bsrCellKind::kDots123456: stringForCell = "\x3d"; break;
35  } // switch
36
37  fBrailleOutputStream <<
38      stringForCell;
39  }
```

## Chapter 28

# MusicXML generation

MusicXML text is produced on the standard output stream, unless options `'-output-file-name'` or `'-auto-output-f` are used.

### 28.1 Basic principle

MusicXML generation is done in two passes:

- first create and MXSR containing the data;
- then simply write this tree.

### 28.2 Creating an xmlelement

An simple example is:

```
1 // create a direction element
2 Sxmlelement directionElement = createMxmlelement (k_direction, "");
3
4 // set it's "placement" attribute if relevant
5 std::string
6     placementString =
7     msrPlacementKindAsMusicXMLString (placementKind);
8
9 if (placementString.size ()) {
10     directionElement->add (createMxmlAttribute ("placement", placementString));
11 }
```

This one supplies a value to the `xmlelement` it creates:

```
1 void msr2mxsrTranslator::visitStart (S_msrIdentification& elt)
2 {
3     // composers
4     const std::list<std::string>&
5     composersList =
6     elt->getComposersList ();
7
8     for (
9         std::list<std::string>::const_iterator i=composersList.begin ();
10        i!=composersList.end ();
11        ++i
12    ) {
13        std::string variableValue = (*i);
```

```

14
15 // create a creator element
16 Sxmlelement creatorElement = createMxmlelement (k_creator, variableValue);
17
18 // set its "type" attribute
19 creatorElement->add (createMxmlAttribute ("type", "composer"));
20
21 // append it to the composers elements list
22 fComposersElementsList.push_back (creatorElement);
23 } // for
24
25 // ... ..
26 }

```

## 28.3 Creating an xmlelement tree

In , this code:

```

1 void msr2mxsrTranslator::visitStart (S_msrClef& elt)
2 {
3     // ... ..
4
5     Sxmlelement clefElement = createMxmlelement (k_clef, "");
6
7     // set clefElement's "number" attribute if relevant
8     /*
9      0 by default in MSR,
10     1 by default in MusicXML:
11     The optional number attribute refers to staff numbers within the part,
12     from top to bottom on the system.
13     A value of 1 is assumed if not present.
14     */
15
16     int clefStaffNumber =
17         elt->getClefStaffNumber ();
18
19     if (clefStaffNumber > 1) {
20         clefElement->add (
21             createMxmlIntegerAttribute ("number", clefStaffNumber));
22     }
23
24     // populate clefElement
25     switch (elt->getClefKind ()) {
26         // ... ..
27
28         case msrClefKind::kClefTrebleMinus8:
29             {
30                 clefElement->push (
31                     createMxmlelement (
32                         k_sign,
33                         "G"));
34                 clefElement->push (
35                     createMxmlIntegerElement (
36                         k_line,
37                         2));
38                 clefElement->push (
39                     createMxmlIntegerElement (
40                         k_clef_octave_change,
41                         -1));
42             }
43             break;
44
45         // ... ..
46     }

```



creates this MusicXML element depending on the value returned by method `msrClef::getClefStaffNumber ()`:

```

1 <clef number="2">
2   <sign>G</sign>
3   <line>2</line>
4   <clef-octave-change>-1</clef-octave-change>
5 </clef>

```

## 28.4 Browsing the visited MSR score

The creation of the tree is done in `src/passes/msr2mxsr/msr2mxsrTranslator.h/.cpp`.

Class `msr2mxsrTranslator` is defined in those files, it contains:

```

1 public:
2
3         msr2mxsrTranslator (
4             const S_msrScore& visitedMsrScore);
5
6     virtual ~msr2mxsrTranslator ();
7
8     Sxmlelement translateMsrToMxsr ();
9
10    // ... ..
11
12 private:
13
14    // the MSR score we're visiting
15    // -----
16    S_msrScore fVisitedMsrScore;
17
18
19    // the MXSR we're building
20    // -----
21    Sxmlelement fResultingMusicxmlelement;

```

The method `msr2mxsrTranslator::translateMsrToMxsr ()` method does the following:

```

1 // -----
2 Sxmlelement msr2mxsrTranslator::translateMsrToMxsr ()
3 {
4     // sanity check
5     mfAssert (
6         __FILE__, __LINE__,
7         fVisitedMsrScore != nullptr,
8         "fVisitedMsrScore is null");
9
10    // create the current score part-wise element
11    fResultingMusicxmlelement =
12        createMxmlScorePartWiseElement ();
13
14    // create a msrScore browser
15    msrBrowser<msrScore> browser (this);
16
17    // browse the visited score with the browser
18    browser.browse (*fVisitedMsrScore);
19
20    return fResultingMusicxmlelement;
21 }

```

## 28.5 Ancillary functions to create MXSR data

The function `createMxmlScorePartWiseElement ()` is defined in `src/formats/mxsr/mxsr.h/.cpp`:

```
1 //-----
2 Sxmlelement createMxmlScorePartWiseElement ()
3 {
4     Sxmlelement result = factory::instance ().create (k_score_partwise);
5
6     Sxmlelement versionAttribute = createMxmlAttribute("version", "3.1");
7     result->add (versionAttribute);
8
9     return result;
10 }
```

## Chapter 29

# Guido code generation

Guido code is produced on standard output, unless options option `-output-file-name`, `-o` or option `-auto-output-file-name`, `-aofn` are used.

### 29.1 Basic principle

As is done for MusicXML generation, Guido generation is done in two passes:

- first create and `mxsr` containing the data;
- then simply write this tree.

The creation of the tree is done in [src/passes/msr2mxsr/msr2mxsrTranslator.h/.cpp](#). See subsection [28.1](#) [musicxmlGeneration], page [227](#), for more details.

---

# Part VIII

## Generators

## Chapter 30

# The generators

A generator creates a music score ex-nihilo, without any description of the music being input. It's behaviour can be adapted to the users needs with options if needed.

Generators are supplied in the `src/generators/` directory. They don't have any interface in at the time of this writing, even though they could.

### 30.1 MusicAndHarmonies

`MusicAndHarmonies.cpp`

### 30.2 Mikrokosmos3Wandering

This service produces the score for Zoltán Kodály's Mikrokosmos III Wandering score, taking inspiration from the same example in Abjad ([http://abjad.mbrsi.org/literature\\_examples/bartok.html](http://abjad.mbrsi.org/literature_examples/bartok.html)). Is was written in the first place to check the MSR API before writing the MSDDL converter.

The score produced is shown at figure 30.1 [Zoltán Kodály's Mikrokosmos III Wandering], page 233.



Figure 30.1: Zoltán Kodály's Mikrokosmos III Wandering

### 30.3 LilyPondIssue34


This service produces the same score as that obtained by:

```
1 xml2ly -auto-output-file-name gracenotes/LilyPondIssue34.xml
```

The resulting score is shown at figure 30.2 [The LilyPondIssue34 score], page 234.

## Piano Sonata in A Major

Wolfgang A



Piano

Figure 30.2: The LilyPondIssue34 score

The name LilyPondIssue34 stems from the fact that translating this MusicXML file to LilyPond with `musicxml2ly` exhibits the famous LilyPond issue #34.

This example was written to design a LilyPond-oriented interface to LPSR, preparing the grounds for LilyPond export to other formats. This work is in progress at the time of this writing.

---

# Part IX

## Converters

## Chapter 31

# The converters

A multi-pass converter performs a sequence of passes, i.e. a sequence of steps. For example, `xml2ly` performs the following passes:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/musicxml > 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.
```

### 31.1 xml2ly

MusicXML (*Music eXtended Markup Language*) is a specification language meant to represent music scores by texts, readable both by humans and computers. It has been designed by the W3C Music Notation Community Group (<https://www.w3.org/community/music-notation/>) to help sharing music score files between applications, through export and import mechanisms.

The homepage to MusicXML is <https://www.musicxml.com>.

MusicXML data contains very detailed information about the music score, and it is quite verbose by nature. This makes creating such data by hand quite difficult, and this is done by applications actually.

The MusicXML data is not systematically checked for correctness. Checks are done, however, to ensure it won't crash due to missing values.

### 31.2 xml2brl

`xml2brl` is mentioned here, but not described in detail.



**31.3** xml2xml**31.4** xml2gmh**31.5** msdlconverter

---

# Part X

## Interfaces

## Chapter 32

# Library interfaces

## Chapter 33

# Representations interfaces

These interfaces are a set of functions to create formats for various needs.

### 33.1 MSR interfaces

The MSR interfaces are in `interfaces/msrinterfaces/`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll interfaces/msrinterfaces/  
2 total 32  
3 0 drwxr-xr-x  5 jacquesmenu  staff    160 May 26 08:20:55 2021 ./  
4 0 drwxr-xr-x  8 jacquesmenu  staff    256 Jun 25 05:59:13 2021 ../  
5 8 -rw-r--r--@ 1 jacquesmenu  staff     77 Apr 22 15:49:27 2021 README.md  
6 16 -rw-r--r--@ 1 jacquesmenu  staff   5796 Jun 24 17:47:02 2021 msrInterface.cpp  
7 8 -rw-r--r--@ 1 jacquesmenu  staff   1371 Jun 13 07:38:04 2021 msrInterface.h
```

### 33.2 LPSR interfaces

The LSPR interfaces are in `interfaces/lpsrinterfaces/`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll interfaces/lpsrinterfaces/  
2 total 24  
3 0 drwxr-xr-x  5 jacquesmenu  staff    160 Jun 13 07:36:53 2021 ./  
4 0 drwxr-xr-x  8 jacquesmenu  staff    256 Jun 25 05:59:13 2021 ../  
5 8 -rw-r--r--@ 1 jacquesmenu  staff     78 Jun 13 07:37:13 2021 README.md  
6 8 -rw-r--r--@ 1 jacquesmenu  staff    670 Jun 13 07:41:01 2021 lpsrInterface.cpp  
7 8 -rw-r--r--@ 1 jacquesmenu  staff   1450 Jun 13 07:39:29 2021 lpsrInterface.h
```

### 33.3 MSDL interfaces

The MSDL interfaces are in `interfaces/msdlinterfaces/`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll interfaces/msdlinterfaces/  
2 total 8  
3 0 drwxr-xr-x  3 jacquesmenu  staff     96 Jun 25 05:57:39 2021 ./  
4 0 drwxr-xr-x  8 jacquesmenu  staff    256 Jun 25 05:59:13 2021 ../  
5 8 -rw-r--r--@ 1 jacquesmenu  staff   1967 Jun  6 06:38:55 2021 libmsdl.h
```

## Chapter 34

# Passes interfaces

MusicFormats provides its functionality in two ways:

- a set of API functions providing its services to any application, including the ones hosted on the Web;
- a set of CLI tools, to be used in terminals and scripts.

The CLI tool actually use the API functions to do their job. For example, in `mfcli`, the `main ()` function does:

```
1 int main (int argc, char* argv[])
2 // -----
3 {
4     // setup signals catching
5     // -----
6
7     catchSignals ();
8
9     // ... ..
10
11     switch (multiGenerationOutputKind) {
12         case mfMultiGenerationOutputKind::kGeneration_UNKNOWN:
13             // should not occur, unless the run is a pure help one
14             break;
15
16         // ... ..
17
18         case mfMultiGenerationOutputKind::kGeneration_Guido:
19             err =
20                 msrScore2guidoWithHandler (
21                     theMsrScore,
22                     "Pass 2",
23                     "Convert the MSR score into a second MSR",
24                     "Pass 3",
25                     "Convert the second MSR into an MXSR",
26                     "Pass 4",
27                     "Convert the MXSR into Guido text",
28                     std::cout,
29                     std::cerr,
30                     handler);
31             break;
32
33         // ... ..
34     }
```

## **34.1 Translating MusicXML data to an MXSR**

## **34.2 Translating an MXSR to an MSR**

## **34.3 Translating an MSR to an MXSR**

## **34.4 Translating an MSR to another MSR**

Such translation is meant to offer an opportunity to modify the score's description depending on options.

## **34.5 Translating an MSR to an LPSR**

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

## **34.6 Translating an LPSR to LilyPond code**

## **34.7 Translating an MSR to an BSR**

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

## **34.8 Translating a BSR to another BSR**

## **34.9 Translating an MXSR to Guido**

## Chapter 35

# Converters interfaces

These interfaces are a set of functions to run the various converters. They are placed in the corresponding subdirectories of `src/converters/`, such as `src/converters/musicxml2musicxml/musicxml2musicxmlInterface.h`.

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/converters/musicxml2musicxml > cat
   musicxml2musicxmlInterface.h
2 /*
3  MusicFormats Library
4  Copyright (C) Jacques Menu 2016-2022
5
6  This Source Code Form is subject to the terms of the Mozilla Public
7  License, v. 2.0. If a copy of the MPL was not distributed with this
8  file, You can obtain one at http://mozilla.org/MPL/2.0/.
9
10 https://github.com/jacques-menu/musicformats
11 */
12
13 #ifndef __musicxml2musicxmlInterface__
14 #define __musicxml2musicxmlInterface__
15
16 #include "mfMusicformatsErrors.h" // for mfMusicformatsErrorKind
17
18 namespace MusicFormats
19 {
20 /*
21  The API functions with an options and arguments and no handler
22  are declared in libmusicxml.h
23  */
24
25 // -----
26 EXP mfMusicformatsErrorKind convertMusicxmlFile2musicxmlWithHandler (
27     const char*   fileName,
28     std::ostream& out,
29     std::ostream& err,
30     const S_oahHandler& handler);
31
32 // -----
33 EXP mfMusicformatsErrorKind convertMusicxmlFd2musicxmlWithHandler (
34     FILE*         fd,
35     std::ostream& out,
36     std::ostream& err,
37     const S_oahHandler& handler);
38
39 // -----
40 EXP mfMusicformatsErrorKind convertMusicxmlString2musicxmlWithHandler (
41     const char*   buffer,
42     std::ostream& out,
43     std::ostream& err,
```

```
45     const S_oahHandler& handler);  
46  
47  
48 }  
49  
50  
51 #endif
```



---

## Part XI

# Distributions and versions

## Chapter 36

# MusicFormats distributions

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

The distributions Zip archives are supplied with all MusicFormats versions, i.e. the current, most recent version of MusicFormats (the default master branch in the repository), and the earlier versions such as the v0.9.65 branch.

### 36.1 GitHub actions

These actions are defined in .yaml files in .github/workflows/:

```
1 jacquesmenu@macmini:~/musicformats-git-dev/.github/workflows > ls -sal
2 total 24
3 0 drwxr-xr-x  5 jacquesmenu  staff    160 Aug 24 09:35 .
4 0 drwxr-xr-x  4 jacquesmenu  staff    128 Aug 22 08:41 ..
5 8 -rw-r--r--@ 1 jacquesmenu  staff   1366 Aug 23 07:09 build-macos-version.yaml
6 8 -rw-r--r--@ 1 jacquesmenu  staff   1371 Aug 23 07:09 build-ubuntu-version.yaml
7 8 -rw-r--r--@ 1 jacquesmenu  staff   1455 Aug 23 07:08 build-windows-version.yaml
```

For example, the Ubuntu action in file build-ubuntu-version.yaml is shown below. It is executed each time a git push is performed to the master branch:

```
1 # This is a workflow to build MusicFormats and create a distribution of it
2
3 name: Build Ubuntu Version
4
5 # Controls when the action will run.
6 on:
7   # Triggers the workflow on push or pull request events but only for the master branch
8   push:
9     branches: [ master ]
10  pull_request:
11    branches: [ master ]
12
13 # Allows you to run this workflow manually from the Actions tab
14 workflow_dispatch:
15
16 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
17 jobs:
18   # This workflow contains a single job called "build"
19   build:
20     # The type of runner that the job will run on
21     runs-on: ubuntu-latest
```

```

22
23 # Steps represent a sequence of tasks that will be executed as part of the job
24 steps:
25 # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
26 - uses: actions/checkout@v2
27
28 - name: Build MusicFormats for Ubuntu
29   run: make -C build
30
31 - name: Upload libraries and executables for Ubuntu
32   uses: actions/upload-artifact@v2
33   with:
34     name: musicformats-ubuntu-version
35     path: |
36       MusicFormatsVersionNumber.txt
37       MusicFormatsVersionDate.txt
38       build/bin
39       build/lib
40       documentation/IntroductionToMusicXML/IntroductionToMusicXML.pdf
41       documentation/MusicFormatsUserGuide/MusicFormatsUserGuide.pdf

```

After a push to the master branch:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git commit -m "Switched from C++11 to C++17
  for <filesystem>" -a[master 77d3d29] Switched from C++11 to C++17 for <filesystem>
2 7 files changed, 10 insertions(+), 10 deletions(-)
3
4 jacquesmenu@macmini: ~/musicformats-git-dev > git push
5 Enumerating objects: 33, done.
6 Counting objects: 100% (33/33), done.
7 Delta compression using up to 8 threads
8 Compressing objects: 100% (16/16), done.
9 Writing objects: 100% (17/17), 1.47 KiB | 1.47 MiB/s, done.
10 Total 17 (delta 14), reused 0 (delta 0), pack-reused 0
11 remote: Resolving deltas: 100% (14/14), completed with 13 local objects.
12 To https://github.com/jacques-menu/musicformats.git
13 a880063..77d3d29 master -> master

```

we get for example:

github.com/jacques-menu/musicformats/actions

Bienvenue sur e-PJ MusicFormats Qt C++ MusicXML Brother Mac OS 12 LilyPond Mac OS 11 Mac OS X JavaScript Sax Soprano

Actions · jacques-menu/musicformats

Search or jump to...

Pull requests Issues Marketplace Explore

jacques-menu / musicformats Public

Pin Unwatch 2 Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Workflows

New workflow

All workflows

MacOS

Node.js Package

Ubuntu

Windows

All workflows

Showing runs from all workflows

Filter workflow runs

777 workflow runs

Event Status Branch Actor

Switched from C++11 to C++17 for <filesystem>

Ubuntu #8: Commit 38b584f pushed by jacques-menu

master

4 hours ago 37m 47s

Switched from C++11 to C++17 for <filesystem>

Windows #8: Commit 38b584f pushed by jacques-menu

master

4 hours ago 46m 25s

Switched from C++11 to C++17 for <filesystem>

MacOS #8: Commit 38b584f pushed by jacques-menu

master

4 hours ago 20m 22s

Then clicking on the link leads to:

259/415

github.com/jacques-menu/musicformats/actions/runs/1921985888

Bienvenue sur e-PJ MusicFormats Qt C++ MusicXML Brother Mac OS 12 LilyPond Mac OS 11 Mac OS X JavaScript Sax Soprano

Switched from C++11 to C++17 for <filesystem> - jacques-menu/musicformats@38b584f

Search or jump to... Pull requests Issues Marketplace Explore

jacques-menu / musicformats Public

Pin Unwatch 2 Fork 0 Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Switched from C++11 to C++17 for <filesystem> Ubuntu #8 Re-run all jobs

Summary

Jobs

build

Triggered via push 19 hours ago

jacques-menu pushed 38b584f master

Status Success

Total duration 37m 47s

Artifacts 1

ubuntu-check.yml

on: push

build 37m 35s

Artifacts

Produced during runtime

Name	Size
musicformats-ubuntu-distrib	79.3 MB

The `musicformats-ubuntu-distrib` archive has to be clicked to get downloaded, since its URL cannot be guessed by an algorithm (it contains numbers internal to GitHub).

Doing so for the three distributions, we get the following, here in the `Downloads/` folder on Mac OS<sup>TM</sup>, with the Zip archives are automatically uncompressed :

```

1 jacquesmenu@macmini: ~/Downloads > ls -sal musicformats-*-distrib
2 musicformats-macos-distrib:
3 total 8
4 0 drwx-----@ 5 jacquesmenu staff 160 Mar 3 09:18 .
5 0 drwx-----+ 72 jacquesmenu staff 2304 Mar 3 09:18 ..
6 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:10 MusicFormatsVersionNumber.txt
7 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Mar 3 09:18 build
8 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 documentation
9
10 musicformats-ubuntu-distrib:
11 total 8
12 0 drwx-----@ 5 jacquesmenu staff 160 Mar 3 09:18 .
13 0 drwx-----+ 72 jacquesmenu staff 2304 Mar 3 09:18 ..
14 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:31 MusicFormatsVersionNumber.txt
15 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 build
16 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 documentation
17
18 musicformats-windows-distrib:
19 total 8
20 0 drwx-----@ 5 jacquesmenu staff 160 Mar 3 09:18 .
21 0 drwx-----+ 72 jacquesmenu staff 2304 Mar 3 09:18 ..
22 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:43 MusicFormatsVersionNumber.txt
23 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 build

```

```
24 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 documentation
```

```
1 jacquesmenu@macmini: ~/Downloads > ls -sal musicformats-ubuntu-distrib/*
2 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:31 musicformats-ubuntu-distrib/
   MusicFormatsVersionNumber.txt
3
4 musicformats-ubuntu-distrib/build:
5 total 0
6 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 .
7 0 drwx-----@ 5 jacquesmenu staff 160 Mar 3 09:18 ..
8 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Mar 3 09:18 bin
9 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 lib
10
11 musicformats-ubuntu-distrib/documentation:
12 total 0
13 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 .
14 0 drwx-----@ 5 jacquesmenu staff 160 Mar 3 09:18 ..
15 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Mar 3 09:18 IntroductionToMusicXML
16 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Mar 3 09:18 MusicFormatsUserGuide
```

```
1 jacquesmenu@macmini: ~/Downloads > ls -sal musicformats-ubuntu-distrib/*/*
2 musicformats-ubuntu-distrib/build/bin:
3 total 2272
4 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Mar 3 09:18 .
5 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 ..
6 96 -rw-r--r--@ 1 jacquesmenu staff 49008 Mar 3 07:31 LilyPondIssue34
7 96 -rw-r--r--@ 1 jacquesmenu staff 49048 Mar 3 07:31 Mikrokosmos3Wandering
8 96 -rw-r--r--@ 1 jacquesmenu staff 47280 Mar 3 07:31 MusicAndHarmonies
9 96 -rw-r--r--@ 1 jacquesmenu staff 47272 Mar 3 07:31 RandomChords
10 96 -rw-r--r--@ 1 jacquesmenu staff 47272 Mar 3 07:31 RandomMusic
11 72 -rw-r--r--@ 1 jacquesmenu staff 33848 Mar 3 07:31 countnotes
12 40 -rw-r--r--@ 1 jacquesmenu staff 17648 Mar 3 07:31 displayMusicformatsHistory
13 40 -rw-r--r--@ 1 jacquesmenu staff 17648 Mar 3 07:31 displayMusicformatsVersion
14 104 -rw-r--r--@ 1 jacquesmenu staff 50400 Mar 3 07:31 msdlconverter
15 544 -rw-r--r--@ 1 jacquesmenu staff 276024 Mar 3 07:31 partsummary
16 88 -rw-r--r--@ 1 jacquesmenu staff 43768 Mar 3 07:31 readunrolled
17 80 -rw-r--r--@ 1 jacquesmenu staff 39064 Mar 3 07:31 xml2brl
18 80 -rw-r--r--@ 1 jacquesmenu staff 39104 Mar 3 07:31 xml2gmn
19 48 -rw-r--r--@ 1 jacquesmenu staff 23192 Mar 3 07:31 xml2guido
20 72 -rw-r--r--@ 1 jacquesmenu staff 34816 Mar 3 07:31 xml2ly
21 88 -rw-r--r--@ 1 jacquesmenu staff 42928 Mar 3 07:31 xml2midi
22 80 -rw-r--r--@ 1 jacquesmenu staff 39104 Mar 3 07:31 xml2xml
23 88 -rw-r--r--@ 1 jacquesmenu staff 43416 Mar 3 07:31 xmlclone
24 48 -rw-r--r--@ 1 jacquesmenu staff 22616 Mar 3 07:31 xmlfactory
25 160 -rw-r--r--@ 1 jacquesmenu staff 79440 Mar 3 07:31 xmliter
26 56 -rw-r--r--@ 1 jacquesmenu staff 28472 Mar 3 07:31 xmlread
27 64 -rw-r--r--@ 1 jacquesmenu staff 28704 Mar 3 07:31 xmltranspose
28 40 -rw-r--r--@ 1 jacquesmenu staff 17360 Mar 3 07:31 xmlversion
29
30 musicformats-ubuntu-distrib/build/lib:
31 total 158600
32 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 .
33 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 ..
34 113728 -rw-r--r--@ 1 jacquesmenu staff 58227464 Mar 3 07:31 libmusicformats.a
35 44872 -rw-r--r--@ 1 jacquesmenu staff 22971160 Mar 3 07:31 libmusicformats.so
36
37 musicformats-ubuntu-distrib/documentation/IntroductionToMusicXML:
38 total 1704
39 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Mar 3 09:18 .
40 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 ..
41 1704 -rw-r--r--@ 1 jacquesmenu staff 869211 Mar 3 07:31 IntroductionToMusicXML.pdf
42
43 musicformats-ubuntu-distrib/documentation/MusicFormatsUserGuide:
44 total 3000
45 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Mar 3 09:18 .
46 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 ..
```

```
47 3000 -rw-r--r--@ 1 jacquesmenu staff 1532300 Mar 3 07:31 MusicFormatsUserGuide.pdf
```

The contents of `musicformats-windows-distrib/` differs in the `lib/` contents:

```
1 jacquesmenu@macmini: ~/Downloads > ls -sal musicformats-windows-distrib/build/lib/
2 total 37672
3      0 drwxr-xr-x@ 4 jacquesmenu staff      128 Mar 3 09:18 .
4      0 drwxr-xr-x@ 4 jacquesmenu staff      128 Mar 3 09:18 ..
5 14768 -rw-r--r--@ 1 jacquesmenu staff 7558913 Mar 3 07:44 musicformats.exp
6 22904 -rw-r--r--@ 1 jacquesmenu staff 11726392 Mar 3 07:44 musicformats.lib
```

For Mac OS™, there is no `lib/` directory, since the executables in `bin/` are self-sufficient. They can be placed anywhere on a disk except the trash. Usually, they are placed in the `/Applications/` directory.

### 36.1.1 Creating the distributions

The hierarchy in the `musicformats-*-distrib/` directories comes from the MusicFormats repository untouched, which is not convenient for the users.

Their contents is thus re-structured by `scripts/MakeMusicFormatsDistributions.bash`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > scripts/MakeMusicFormatsDistributions.bash
2 ... ..
3
4 ==> final distrib contents:
5 -----
6
7 4208 -rw-r--r-- 1 jacquesmenu staff 2153547 Mar 3 12:55 /Users/jacquesmenu/
8 musicformats-git-dev/distrib/MusicFormatsForWindows.zip
9 34576 -rw-r--r-- 1 jacquesmenu staff 17559638 Mar 3 12:55 /Users/jacquesmenu/
10 musicformats-git-dev/distrib/MusicFormatsForUbuntu.zip
11 109512 -rw-r--r-- 1 jacquesmenu staff 55888914 Mar 3 12:55 /Users/jacquesmenu/
12 musicformats-git-dev/distrib/MusicFormatsForMacOS.zip
13 1704 -rw-r--r--@ 1 jacquesmenu staff 869211 Mar 3 07:10 /Users/jacquesmenu/
14 musicformats-git-dev/distrib/IntroductionToMusicXML.pdf
15 3000 -rw-r--r--@ 1 jacquesmenu staff 1532300 Mar 3 07:10 /Users/jacquesmenu/
16 musicformats-git-dev/distrib/MusicFormatsUserGuide.pdf
17 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:10 /Users/jacquesmenu/
18 musicformats-git-dev/distrib/MusicFormatsVersionNumber.txt
19
20 /Users/jacquesmenu/musicformats-git-dev/distrib/MusicFormatsForWindows:
21 total 8
22 0 drwxr-xr-x 16 jacquesmenu staff 512 Mar 3 12:55 ..
23 0 drwxr-xr-x 5 jacquesmenu staff 160 Mar 3 12:55 .
24 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 lib
25 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Mar 3 09:18 bin
26 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:43 MusicFormatsVersionNumber.txt
27
28 /Users/jacquesmenu/musicformats-git-dev/distrib/MusicFormatsForUbuntu:
29 total 8
30 0 drwxr-xr-x 16 jacquesmenu staff 512 Mar 3 12:55 ..
31 0 drwxr-xr-x 5 jacquesmenu staff 160 Mar 3 12:55 .
32 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 lib
33 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Mar 3 09:18 bin
34 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:31 MusicFormatsVersionNumber.txt
35
36 /Users/jacquesmenu/musicformats-git-dev/distrib/MusicFormatsForMacOS:
37 total 8
38 0 drwxr-xr-x 16 jacquesmenu staff 512 Mar 3 12:55 ..
39 0 drwxr-xr-x 4 jacquesmenu staff 128 Mar 3 12:55 .
40 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Mar 3 07:10 bin
41 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:10 MusicFormatsVersionNumber.txt
```

The contents of `distrib/` is now:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/distrib > ls -sal
2 total 154128
3      0 drwxr-xr-x  16 jacquesmenu  staff          512 Mar  3 13:18 .
4      0 drwxr-xr-x  35 jacquesmenu  staff         1120 Mar  3 07:13 ..
5     24 -rw-r--r--@   1 jacquesmenu  staff         8196 Feb 24 13:33 .DS_Store
6    1704 -rw-r--r--@   1 jacquesmenu  staff        869211 Mar  3 07:10 IntroductionToMusicXML.pdf
7      0 drwxr-xr-x   4 jacquesmenu  staff          128 Mar  3 13:18 MusicFormatsForMacOS
8 109960 -rw-r--r--   1 jacquesmenu  staff    55888914 Mar  3 13:18 MusicFormatsForMacOS.zip
9      0 drwxr-xr-x   5 jacquesmenu  staff          160 Mar  3 13:18 MusicFormatsForUbuntu
10   35216 -rw-r--r--   1 jacquesmenu  staff    17559638 Mar  3 13:18 MusicFormatsForUbuntu.zip
11      0 drwxr-xr-x   5 jacquesmenu  staff          160 Mar  3 13:18 MusicFormatsForWindows
12   4208 -rw-r--r--   1 jacquesmenu  staff    2153547 Mar  3 13:18 MusicFormatsForWindows.zip
13   3000 -rw-r--r--@   1 jacquesmenu  staff    1532300 Mar  3 07:10 MusicFormatsUserGuide.pdf
14      8 -rw-r--r--@   1 jacquesmenu  staff           6 Mar  3 07:10 MusicFormatsVersionNumber.
    txt
15      8 -rwxr-xr-x@   1 jacquesmenu  staff           95 Mar  3 12:54 doClean.bash
16      0 drwx-----@   6 jacquesmenu  staff          192 Mar  3 10:56 musicformats-macos-distrib
17      0 drwx-----@   6 jacquesmenu  staff          192 Mar  3 10:56 musicformats-ubuntu-
    distrib
18      0 drwx-----@   6 jacquesmenu  staff          192 Mar  3 10:56 musicformats-windows-
    distrib

```

### 36.1.2 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 downloading the MusicFormats distributions 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 trouble is that these executables are in *quarantine* by default. To make them usable, they have to quit quarantine and explicitly be made executable.

This is done this way using `chmod` and `xattr` in `scripts/MakeMusicFormatsDistributions.bash`:

```

1 # make the executables actually executable
2 chmod +x bin/*
3 xattr -d com.apple.quarantine bin/*

```

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

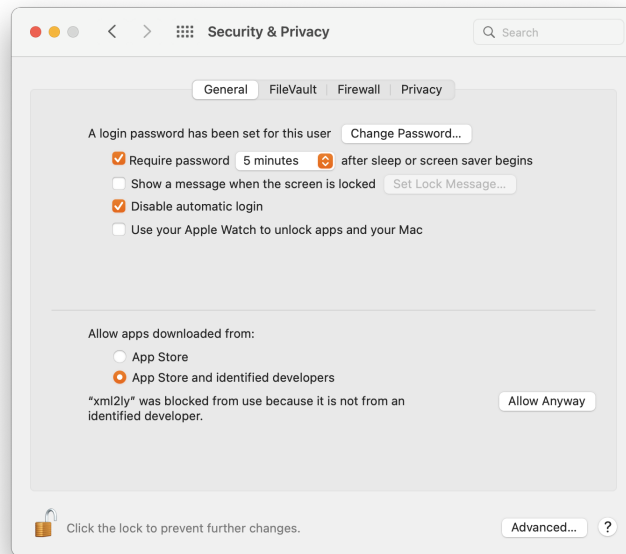
Having to perform the preceding task for each executable is the price to pay for security. And it has to be performed again when installing new versions...

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;

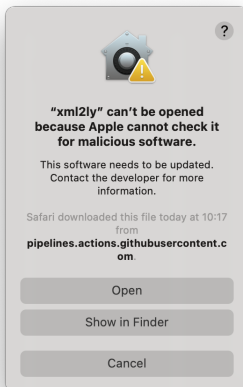


- click on the lock at the bottom left of the dialog to make changes:



- 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.

## Chapter 37

# MusicFormats branches and versions

The MusicFormats repository contains:

- a master branch, that contains the current evolution of the code base, examples and documentation;
- vX.Y.Z branches, created from the master branch where it is in a useful state. An example is v0.9.63.

When a `git push` is performed, the `musicformats-*-distrib` archives are created, but they cannot be added to the MusicFormats repository by GitHub on the fly.

Thus, in order to create a new version of a satisfactory state of the local development repository, one should:

1. check that the version number and date are fine in `MusicFormatsVersionNumber.txt` and `MusicFormatsVersionDate.txt`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > cat MusicFormatsVersionNumber.txt
2 0.9.63jacquesmenu@macmini: ~/musicformats-git-dev > cat MusicFormatsVersionDate.txt
3 June 9, 2022
```

Note that file `MusicFormatsVersionNumber.txt` should not end with an end of line, since that would disturb the creation of the PDF documentation files with L<sup>A</sup>T<sub>E</sub>X;

These informations can be displayed with `scripts/ShowMusicFormatsVersion.bash`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > scripts/ShowMusicFormatsVersion.bash
2 Version number:
3 0.9.63Version date:
4 June 9, 2022
```

2. (re-)create the up-to-date documentation with:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > scripts/CreateDocumentationPDFs.bash
```

3. add all new and/or modified files to the local repository. The `addAll` function is defined for this:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > type addAll
2 addAll is a function
3 addAll ()
4 {
5     git add -f ${MUSIC_FORMATS_DEV}/MusicFormatsVersionNumber.txt;
6     git add -f ${MUSIC_FORMATS_DEV}/MusicFormatsVersionDate.txt;
7     git add -f ${MUSIC_FORMATS_DEV}/src/MusicFormatsVersionNumber.h;
8     git add -f ${MUSIC_FORMATS_DEV}/src/MusicFormatsVersionDate.h;
9     addSrc;
10    addBuild;
11    addScripts;
```

```

12     addDistrib;
13     addDoc;
14     addFXML;
15     addFmfs1
16 }

```

4. commit a first time to the local repository clone with a 'Pre' version number:

```
1 git commit -m "Pre v0.9.63" -a
```

5. push this to the MusicFormats repo with:

```
1 git push
```

6. the actions in the MusicFormats repository perform a build on Linux, Windows™ and Mac OS™. Check that they were executed successfully at <https://github.com/jacques-menu/musicformats/actions>:

The screenshot displays the GitHub Actions interface for the `musicformats` repository. The 'All workflows' section is active, showing a list of workflow runs. The table below summarizes the visible runs:

Workflow Run	Event	Status	Branch	Actor	Duration
Pre v0.9.63 Windows #40: Commit b4b09f8 pushed by jacques-menu	push	Success	master	jacques-menu	1h 0m 14s
Pre v0.9.63 Ubuntu #40: Commit b4b09f8 pushed by jacques-menu	push	Success	master	jacques-menu	34m 14s
Pre v0.9.63 MacOS #40: Commit b4b09f8 pushed by jacques-menu	push	Success	master	jacques-menu	16m 49s

7. when that is the case, download each of the three resulting `musicformats-*-distrib` archives locally in turn:

jacques-menu / musicformats Public

Notifications Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights

✓ Pre v0.9.63 Ubuntu #40 Sign in to view logs

**Summary**

Jobs

✓ build

Triggered via push 2 hours ago

jacques-menu pushed -> b4b09f8 master

Status **Success** Total duration **34m 14s** Artifacts **1**

**ubuntu-check.yml**  
on: push

✓ build 34m 3s

**Artifacts**  
Produced during runtime

Name	Size
musicformats-ubuntu-distrib	83.7 MB

On this authors's machine, they go to `${HOME}/Downloads`:

```

1 jacquesmenu@macmini: ~/Downloads > ls -sal musicformats-*-distrib
2 musicformats-macos-distrib:
3 total 8
4 0 drwx-----@ 5 jacquesmenu staff 160 Jun 9 11:44 .
5 0 drwx-----+ 28 jacquesmenu staff 896 Jun 9 11:44 ..
6 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 07:40 MusicFormatsVersionNumber.txt
7 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Jun 9 11:44 build
8 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Jun 9 11:44 documentation
9
10 musicformats-ubuntu-distrib:
11 total 8
12 0 drwx-----@ 5 jacquesmenu staff 160 Jun 9 11:44 .
13 0 drwx-----+ 28 jacquesmenu staff 896 Jun 9 11:44 ..
14 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 07:57 MusicFormatsVersionNumber.txt
15 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Jun 9 11:44 build
16 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Jun 9 11:44 documentation
17
18 musicformats-windows-distrib:
19 total 8
20 0 drwx-----@ 5 jacquesmenu staff 160 Jun 9 11:43 .
21 0 drwx-----+ 28 jacquesmenu staff 896 Jun 9 11:44 ..
22 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 08:14 MusicFormatsVersionNumber.txt
23 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Jun 9 11:43 build
24 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Jun 9 11:43 documentation

```

8. create the distributions in the local MusicFormats repository clone:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > scripts/MakeMusicFormatsDistributions.
  bash
2 ... ..
3 -----

```

```

4 ==> final distrib contents:
5 -----
6
7 4368 -rw-r--r-- 1 jacquesmenu staff 2234811 Jun 9 11:47 /Users/jacquesmenu/
  musicformats-git-dev/distrib/MusicFormatsForWindows.zip
8 35312 -rw-r--r-- 1 jacquesmenu staff 18076799 Jun 9 11:47 /Users/jacquesmenu/
  musicformats-git-dev/distrib/MusicFormatsForUbuntu.zip
9 127824 -rw-r--r-- 1 jacquesmenu staff 65442854 Jun 9 11:47 /Users/jacquesmenu/
  musicformats-git-dev/distrib/MusicFormatsForMacOS.zip
10 1712 -rw-r--r--@ 1 jacquesmenu staff 872863 Jun 9 07:40 /Users/jacquesmenu/
  musicformats-git-dev/distrib/IntroductionToMusicXML.pdf
11 5328 -rw-r--r--@ 1 jacquesmenu staff 2724130 Jun 9 07:40 /Users/jacquesmenu/
  musicformats-git-dev/distrib/MusicFormatsUserGuide.pdf
12 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 07:40 /Users/jacquesmenu/
  musicformats-git-dev/distrib/MusicFormatsVersionNumber.txt
13
14 /Users/jacquesmenu/musicformats-git-dev/distrib/MusicFormatsForWindows:
15 total 8
16 0 drwxr-xr-x 14 jacquesmenu staff 448 Jun 9 11:47 ..
17 0 drwxr-xr-x 5 jacquesmenu staff 160 Jun 9 11:47 .
18 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Jun 9 11:43 lib
19 0 drwxr-xr-x@ 26 jacquesmenu staff 832 Jun 9 11:43 bin
20 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 08:14 MusicFormatsVersionNumber.txt
21
22 /Users/jacquesmenu/musicformats-git-dev/distrib/MusicFormatsForUbuntu:
23 total 8
24 0 drwxr-xr-x 14 jacquesmenu staff 448 Jun 9 11:47 ..
25 0 drwxr-xr-x 5 jacquesmenu staff 160 Jun 9 11:47 .
26 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Jun 9 11:44 lib
27 0 drwxr-xr-x@ 26 jacquesmenu staff 832 Jun 9 11:44 bin
28 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 07:57 MusicFormatsVersionNumber.txt
29
30 /Users/jacquesmenu/musicformats-git-dev/distrib/MusicFormatsForMacOS:
31 total 8
32 0 drwxr-xr-x 14 jacquesmenu staff 448 Jun 9 11:47 ..
33 0 drwxr-xr-x 4 jacquesmenu staff 128 Jun 9 11:47 .
34 0 drwxr-xr-x@ 26 jacquesmenu staff 832 Jun 9 07:40 bin
35 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 07:40 MusicFormatsVersionNumber.txt

```

Now, the local master branch contains the distribution files of itself:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/distrib > ls -al
2 total 174552
3 drwxr-xr-x 14 jacquesmenu staff 448 Jun 9 11:47 .
4 drwxr-xr-x 29 jacquesmenu staff 928 Jun 9 09:13 ..
5 -rw-r--r--@ 1 jacquesmenu staff 872863 Jun 9 07:40 IntroductionToMusicXML.pdf
6 drwxr-xr-x 4 jacquesmenu staff 128 Jun 9 11:47 MusicFormatsForMacOS
7 -rw-r--r-- 1 jacquesmenu staff 65442854 Jun 9 11:47 MusicFormatsForMacOS.zip
8 drwxr-xr-x 5 jacquesmenu staff 160 Jun 9 11:47 MusicFormatsForUbuntu
9 -rw-r--r-- 1 jacquesmenu staff 18076799 Jun 9 11:47 MusicFormatsForUbuntu.zip
10 drwxr-xr-x 5 jacquesmenu staff 160 Jun 9 11:47 MusicFormatsForWindows
11 -rw-r--r-- 1 jacquesmenu staff 2234811 Jun 9 11:47 MusicFormatsForWindows.zip
12 -rw-r--r--@ 1 jacquesmenu staff 2724130 Jun 9 07:40 MusicFormatsUserGuide.pdf
13 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 07:40 MusicFormatsVersionNumber.
  txt
14 drwx-----@ 5 jacquesmenu staff 160 Jun 9 11:44 musicformats-macos-distrib
15 drwx-----@ 5 jacquesmenu staff 160 Jun 9 11:44 musicformats-ubuntu-distrib
16 drwx-----@ 5 jacquesmenu staff 160 Jun 9 11:43 musicformats-windows-distrib

```

9. *commit and push again* with the new version name, no 'Pre' this time, in the `-m "... .."` message, such as:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git commit -m "v0.9.63" -a
2
3 jacquesmenu@macmini: ~/musicformats-git-dev > git push

```

## 10. create the new version branch locally and remotely:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git push --set-upstream origin v0.9.63
2 Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
3 remote:
4 remote: Create a pull request for 'v0.9.63' on GitHub by visiting:
5 remote:     https://github.com/jacques-menu/musicformats/pull/new/v0.9.63
6 remote:
7 To https://github.com/jacques-menu/musicformats.git
8 * [new branch]      v0.9.63 -> v0.9.63
9 branch 'v0.9.63' set up to track 'origin/v0.9.63'.
10
11 jacquesmenu@macmini: ~/musicformats-git-dev > git branch -r
12 origin/HEAD -> origin/master
13 origin/gh-pages
14 origin/master
15 origin/v0.9.60
16 origin/v0.9.61
17 origin/v0.9.62
18 origin/v0.9.63
19
20 jacquesmenu@macmini: ~/musicformats-git-dev > git branch
21 * master
22 v0.9.63

```

## 11. create a new version number and date, for example:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > scripts/SetMusicFormatsVersionNumber.
  bash 0.9.64
2 -bash: scripts/SetMusicFormatsVersionNumber.: No such file or directory
3 jacquesmenu@macmini: ~/musicformats-git-dev > scripts/SetMusicFormatsVersionNumber.
  bash 0.9.64
4 ==> PWD is:
5 /Users/jacquesmenu/musicformats-git-dev
6
7 ==> Writing MusicFormats version number 0.9.64 to MusicFormatsVersionNumber.txt
8
9 8 -rw-r--r--  1 jacquesmenu  staff   6 Jun  9 12:14:57 2022 MusicFormatsVersionNumber.
  txt
10 0.9.64
11 ==> PWD is:
12 /Users/jacquesmenu/musicformats-git-dev/src
13
14 ==> Writing MusicFormats version number 0.9.64 to MusicFormatsVersionNumber.h
15
16 8 -rw-r--r--  1 jacquesmenu  staff  45 Jun  9 12:14:57 2022 MusicFormatsVersionNumber
  .h
17 #define MUSICFORMATS_VERSION_NUMBER "0.9.64"

```

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > scripts/SetMusicFormatsVersionDate.bash
  "June 9, 2022"
2 ==> PWD is:
3 /Users/jacquesmenu/musicformats-git-dev
4
5 ==> Writing MusicFormats version date June 9, 2022 to MusicFormatsVersionDate.txt
6
7 8 -rw-r--r--  1 jacquesmenu  staff  13 Jun  9 12:15:52 2022 MusicFormatsVersionDate.
  txt
8 June 9, 2022
9
10 ==> PWD is:
11 /Users/jacquesmenu/musicformats-git-dev/src
12
13 ==> Writing MusicFormats version date June 9, 2022 to MusicFormatsVersionDate.h
14
15 8 -rw-r--r--  1 jacquesmenu  staff  49 Jun  9 12:15:52 2022 MusicFormatsVersionDate.h
16 #define MUSICFORMATS_VERSION_DATE "June 9, 2022"

```

Check the result with:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > scripts/ShowMusicFormatsVersion.bash
2 Version number:
3 0.9.64Version date:
4 June 9, 2022
```

---

# Part XII

## Selected topics



## Chapter 38

# Initializations

Some initialization activities in MusicFormats use the OAH facility. OAH should thus be initialized first.

### 38.1 Options and help initializations

There is no initialization of the OAH architecture as such, but there are `create*OahGroup ()` functions to create the various OAH groups.

For example, global variable `gGlobalServiceRunData` is supplied by `src/mflibrary/mfServiceRunData.h/.cpp`:

```
1 EXP extern S_generalOahGroup gGlobalServiceRunData;
2
3 // -----
4 EXP S_generalOahGroup createGlobalGeneralOahGroup ();

1 S_generalOahGroup createGlobalGeneralOahGroup ()
2 {
3 #ifdef TRACING_IS_ENABLED
4     if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
5         gLogStream <<
6             "Creating global general OAH group" <<
7             std::endl;
8     }
9 #endif
10
11 // protect library against multiple initializations
12 if (! gGlobalServiceRunData) {
13     // create the global general options group
14     gGlobalServiceRunData =
15         generalOahGroup::create ();
16     assert (gGlobalServiceRunData != 0);
17 }
18
19 // return the global OAH group
20 return gGlobalServiceRunData;
21 }
```

## 38.2 Representations initializations

There are `initialize*` () functions such as `initializeLPSR` () and `initializeBSR` (). They essentially build global data structures, such as the tables of supported languages and their correspondance with an internal enumeration type both ways.

For example, `initializeMSR` () is defined in `src/formats/msr/msr.h/.cpp`:

```

1 void EXP initializeMSR ();

1 static S_mfcVersions pMsrRepresentationComponent;
2
3 static void initializeMsrRepresentationComponent ()
4 {
5     // create the component
6     pMsrRepresentationComponent =
7         mfcVersions::create ("MSR");
8
9     // populate it
10    pMsrRepresentationComponent->
11        appendVersionDescrToComponent (
12            mfcVersionDescr::create (
13                mfcVersionDescr::create (
14                    mfcVersionNumber::createFromString ("0.9.50"),
15                    "October 6, 2021",
16                    std::list<std::string> {
17                        "Start of sequential versions numbering"
18                    }
19                )))
20 }
21
22 void initializeMSR ()
23 {
24     // protect library against multiple initializations
25     static Bool pPrivateThisMethodHasBeenRun (false);
26
27     if (! pPrivateThisMethodHasBeenRun) {
28         // initialize the history
29         initializeMsrRepresentationComponent ();
30
31         // initialize
32         initializeMsrBasicTypes ();
33
34         pPrivateThisMethodHasBeenRun = true;
35     }
36 }

```

### 38.2.1 MSR initialization

`src/formats/msr/msrBasicTypes.h/.cpp` defines function `initializeMSRBasicTypes` () for this initialization:

```

1 void initializeMsrBasicTypes ()
2 {
3     // protect library against multiple initializations
4     static Bool pPrivateThisMethodHasBeenRun (false);
5
6     if (! pPrivateThisMethodHasBeenRun) {
7 #ifdef TRACING_IS_ENABLED
8         if (gGlobalOahEarlyOptions.getEarlyTracingOah () && ! gGlobalOahEarlyOptions.
9             getEarlyQuietOption ()) {
10             gLogStream <<

```

```

10     "Initializing MSR basic types handling" <<
11     std::endl;
12 }
13 #endif
14
15 // languages handling
16 // -----
17
18 initializeQuarterTonesPitchesLanguageKinds ();
19
20 // clefs handling
21 // -----
22
23 initializeClefKinds ();
24
25 // harmonies handling
26 // -----
27
28 initializeHarmonyKinds ();
29
30 // harmony structures handling
31 // -----
32
33 initializeHarmonyStructuresMap ();
34
35 // MSR lengths handling
36 // -----
37
38 initializeMsrLengthUnitKindsMap ();
39
40 // MSR margins types handling
41 // -----
42
43 initializeMsrMarginTypeKindsMap ();
44
45 pPrivateThisMethodHasBeenRun = true;
46 }
47 }

```

### 38.2.2 LPSR initialization

### 38.2.3 BSR initialization

## 38.3 Passes initializations

## 38.4 Converters initializations

The converters create only the global OAH groups they need. Since the order of initializations is critical, initialization of the formats is done when the latter's insider handler is created.

This is how class `xml2lyInsiderHandler` initializes the MSR and LSPR formats in method `xml2lyInsiderHandler::createTheXml2lyOptionGroups ()` in [src/converters/musicxml2lilypond/musicxml2lilypondInsiderHandler.cpp](#):

```

1 void xml2lyInsiderHandler::createTheXml2lyOptionGroups (
2     std::string serviceName)
3 {
4     // ... ..
5
6     // initialize options handling, phase 1

```

```

7 // -----
8
9 // create the OAH OAH group first
10 appendGroupToHandler (
11     createGlobalOahOahGroup (
12         serviceName));
13
14 // create the WAE OAH group
15 appendGroupToHandler (
16     createGlobalWaeOahGroup ());
17
18 #ifdef TRACING_IS_ENABLED
19 // create the tracing OAH group
20 appendGroupToHandler (
21     createGlobalTracingOahGroup (
22         this));
23 #endif
24
25 // create the output file OAH group
26 appendGroupToHandler (
27     createGlobalOutputFileOahGroup ());
28
29 // initialize the library
30 // -----
31
32 initializeWAE ();
33
34 initializeMSR ();
35 initializeLPSR ();
36
37 // initialize options handling, phase 2
38 // -----
39
40 // create the MXSR OAH group
41 appendGroupToHandler (
42     createGlobalMxsrOahGroup ());
43
44 // create the mxsr2msr OAH group
45 appendGroupToHandler (
46     createGlobalMxsr2msrOahGroup (
47         this));
48
49 // create the MSR OAH group
50 appendGroupToHandler (
51     createGlobalMsrOahGroup ());
52
53 // create the msr2msr OAH group
54 appendGroupToHandler (
55     createGlobalMsr2msrOahGroup ());
56
57 // create the msr2lpsr OAH group
58 appendGroupToHandler (
59     createGlobalMsr2lpsrOahGroup ());
60
61 // create the LPSR OAH group
62 appendGroupToHandler (
63     createGlobalLpsrOahGroup ());
64
65 // create the LilyPond generation OAH group
66 appendGroupToHandler (
67     createGlobalLpsr2lilypondOahGroup ());
68
69 #ifdef EXTRA_OAH_IS_ENABLED
70 // create the extra OAH group
71 appendGroupToHandler (
72     createGlobalHarmoniesExtraOahGroup ());
73 #endif

```

```
74
75 // create the global xml2ly OAH group only now,
76 // after the groups whose options it may use
77 // have been created
78 appendGroupToHandler (
79     createGlobalXml2lyInsiderOahGroup ());
80
81 // ... ..
82 }
```

## Chapter 39

# The OAH atoms collection

These handy general-purpose OAH atoms are used in MusicFormats itself. They are defined in [src/oah/oahAtomsCollection.h](#).

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class    oahAtomsCollection.h
2 class EXP oahAtomAlias : public oahAtom
3 class EXP oahMacroAtom : public oahAtom
4 class EXP oahOptionsUsageAtom : public oahPureHelpAtomWithoutAValue
5 class EXP oahHelpAtom : public oahPureHelpAtomWithoutAValue
6 class EXP oahHelpSummaryAtom : public oahPureHelpAtomWithoutAValue
7 class EXP oahAboutAtom : public oahPureHelpAtomWithoutAValue
8 class EXP oahVersionAtom : public oahPureHelpAtomWithoutAValue
9 class EXP oahContactAtom : public oahPureHelpAtomWithoutAValue
10 class EXP oahBooleanAtom : public oahAtom
11 class EXP oahTwoBooleansAtom : public oahBooleanAtom
12 class EXP oahThreeBooleansAtom : public oahBooleanAtom
13 class EXP oahCombinedBooleansAtom : public oahAtom
14 class EXP oahCommonPrefixBooleansAtom : public oahAtom
15 class EXP oahIntegerAtom : public oahAtomStoringAValue
16 class EXP oahTwoIntegersAtom : public oahIntegerAtom
17 class EXP oahFloatAtom : public oahAtomStoringAValue
18 class EXP oahStringAtom : public oahAtomStoringAValue
19 class EXP oahFactorizedStringAtom : public oahAtom
20 class EXP oahStringWithDefaultValueAtom : public oahStringAtom
21 class EXP oahRationalAtom : public oahAtomStoringAValue
22 class EXP oahNaturalNumbersSetElementAtom : public oahAtomStoringAValue
23 class EXP oahColorRGBAtom : public oahAtomStoringAValue
24 class EXP oahIntSetElementAtom : public oahAtomStoringAValue
25 class EXP oahStringSetElementAtom : public oahAtomStoringAValue
26 class EXP oahStringToIntMapElementAtom : public oahAtomStoringAValue
27 class EXP oahStringAndIntegerAtom : public oahAtomStoringAValue
28 class EXP oahStringAndTwoIntegersAtom : public oahAtomStoringAValue
29 class EXP oahLengthUnitKindAtom : public oahAtomStoringAValue
30 class EXP oahLengthAtom : public oahAtomStoringAValue
31 class EXP oahMidiTempoAtom : public oahAtomStoringAValue
32 class EXP oahOptionNameHelpAtom : public oahStringWithDefaultValueAtom
33 class EXP oahQueryOptionNameAtom : public oahPureHelpAtomExpectingAValue
34 class EXP oahFindStringAtom : public oahPureHelpAtomExpectingAValue
```

### 39.1 OAH macro atoms

A OAH macro atom is a combination, a list of several options under a single name. The `oahMacroAtom` class is defined in [src/oah/oahAtomsCollection.h/.cpp](#):

```

1 class EXP oahMacroAtom : public oahAtom
2 {
3     /*
4      a list of atoms
5     */
6
7     // ... ..
8
9     public:
10
11     // public services
12     // -----
13
14     void                appendAtomToMacro (S_oahAtom atom);
15
16     void                applyElement (std::ostream& os) override;
17
18
19     private:
20
21     // private fields
22     // -----
23
24     std::list<S_oahAtom>    fMacroAtomsList;
25 };
26

```

Populating field `oahMacroAtom::fMacroAtomsList` is straightforward:

```

1 void oahMacroAtom::appendAtomToMacro (S_oahAtom atom)
2 {
3     // sanity check
4     mfAssert (
5         __FILE__, __LINE__,
6         atom != nullptr,
7         "atom is null");
8
9     fMacroAtomsList.push_back (atom);
10 }

```

Applying the macro atom is done in method `oahMacroAtom::applyElement ()`:

```

1 void oahMacroAtom::applyElement (std::ostream& os)
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
5             gLogStream <<
6                 "=> option ' ' << fetchNames () << " ' is a oahMacroAtom" <<
7                 std::endl;
8         }
9     #endif
10
11     for (
12         std::list<S_oahAtom>::const_iterator i =
13             fMacroAtomsList.begin ();
14         i != fMacroAtomsList.end ();
15         ++i
16     ) {
17         S_oahAtom atom = (*i);
18
19         if (
20             // oahAtomStoringAValue?
21             S_oahAtomStoringAValue
22             atomWithVariable =
23                 dynamic_cast<oahAtomStoringAValue*>(&(*atom))

```

```

24     ) {
25     //      atomWithVariable-> JMI ???
26     //      applyAtomWithValue (theString, os);
27     }
28     else {
29     // valueless atom
30     atom->
31     applyElement (os);
32     }
33 } // for
34 }

```

## 39.2 A OAH macro atom example

xml2brl has the `-auto-utf8`, `-au8d` option:

```

1 jacquesmenu@macmini > xml2brl -query auto-utf8d
2 --- Help for atom "auto-utf8d" in subgroup "Files"
3   -auto-utf8d, -au8d
4   Combines -auto-output-file-name, -utf8d and -use-encoding-in-file-name

```

This macro options is defined in `src/formatsgeneration/brailleGeneration/brailleGenerationOah.cpp` the following way:

```

1 void brailleGenerationOahGroup::initializeMacroOptions ()
2 {
3     S_oahSubGroup
4     subGroup =
5     oahSubGroup::create (
6     "Macros",
7     "help-braille-generation-macros", "hbgm",
8     R"()",
9     oahElementVisibilityKind::kElementVisibilityWhole,
10    this);
11
12    appendSubGroupToGroup (subGroup);
13
14    // create the auto utfd8 macro
15
16    S_oahMacroAtom
17    autoUTFd8MacroAtom =
18    oahMacroAtom::create (
19    "auto-utf8d", "au8d",
20    "Combines -auto-output-file-name, -utf8d and -use-encoding-in-file-name");
21
22    subGroup->
23    appendAtomToSubGroup (
24    autoUTFd8MacroAtom);
25
26    // populate it
27    autoUTFd8MacroAtom->
28    appendAtomToMacro (
29    gGlobalOutputFileOahGroup->getAutoOutputFileNameAtom ());
30
31    fBrailleOutputKindAtom->
32    applyAtomWithValue (
33    "utf8d",
34    gLogStream);
35    autoUTFd8MacroAtom->
36    appendAtomToMacro (
37    fBrailleOutputKindAtom);
38
39    autoUTFd8MacroAtom->

```



```

40     appendAtomToMacro (
41         fUseEncodingInFileNameAtom);
42 }

```

### 39.3 LilyPond octave entry

Pass `lpsr2lilypond` has three options to choose this, all controlling one and the same variable:

```

1 jacquesmenu@macmini > xml2ly -query absolute
2 --- Help for atom "absolute" in subgroup "Notes"
3     -abs, -absolute
4         Use absolute octave entry in the generated LilyPond code.

```

```

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

```

```

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

```

This is done in `src/formatsgeneration/lilypondGeneration/lpsr2lilypondOah.h/.cpp` using a single instance of class `msrOctaveEntryVariable`:

```

1 class EXP msrOctaveEntryVariable : public smartable
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        std::string          fVariableName;
11        msrOctaveEntryKind   fOctaveEntryKind;
12
13        Bool                 fSetByAnOption;
14 };

```

The three classes:

- `lilypondAbsoluteOctaveEntryAtom`
- `lilypondRelativeOctaveEntryAtom`
- `lilypondFixedOctaveEntryAtom`

all contain an alias for an class `msrOctaveEntryVariable` variable:

```
1 // private fields
2 // -----
3
4 msrOctaveEntryVariable&
5     fOctaveEntryKindVariable;
```

The `fOctaveEntryVariable` field of class `lpsr2lilypondOahGroup` shared by all three options atoms is:

```
1 // notes
2 // -----
3
4 msrOctaveEntryVariable
5     fOctaveEntryVariable;
```

## Chapter 40

# Measures handling

Measures are presented at section [19.30](#) [Measures], page [187](#).

### 40.1 Voices contents

Class `msrVoice` contains a list of the first elements and a last segment:

```
1  std::list<S_msrVoiceElement>
2      fVoiceInitialElementsList;
3
4  // fVoiceLastSegment contains the music
5  // not yet stored in fVoiceInitialElementsList,
6  // it is thus logically the end of the latter,
7  // and is created implicitly for every voice.
8  // It is needed 'outside' of the 'std::list<S_msrElement>'
9  // because it is not a mere S_msrElement, but a S_msrSegment
10 S_msrSegment      fVoiceLastSegment;
```

### 40.2 Voice elements

The class `msrVoiceElement` sub-classes instances in `fVoiceInitialElementsList` can be of types:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grep 'public msrVoiceElement' formats/
   msr/*.h
2 formats/msr/msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
3 formats/msr/msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
4 formats/msr/msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
5 formats/msr/msrMultipleFullBarRests.h:class EXP msrMultipleFullBarRests : public
   msrVoiceElement
6 formats/msr/msrSegments.h:class EXP msrSegment : public msrVoiceElement
```

Class `msrSegment` contains a list of measures:

```
1  // the measures in the segment contain the mmusic
2  std::list<S_msrMeasure>
3      fSegmentElementsList;
```

class contains a list of measure elements:

```
1  // elements
2
3  std::list<S_msrMeasureElement>
4      fMeasureElementsList;
```

## 40.3 Measure elements

The class `msrMeasureElements` sub-classes instances in can be of types:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'public
  msrMeasureElement' *.h
2 msrBarChecks.h:class EXP msrBarCheck : public msrMeasureElement
3 msrBarLines.h:class EXP msrBarLine : public msrMeasureElement
4 msrBarNumberChecks.h:class EXP msrBarNumberCheck : public msrMeasureElement
5 msrBreaks.h:class EXP msrLineBreak : public msrMeasureElement
6 msrBreaks.h:class EXP msrPageBreak : public msrMeasureElement
7 msrClefs.h:class EXP msrClef : public msrMeasureElement
8 msrCodas.h:class EXP msrCoda : public msrMeasureElement
9 msrDoubleTremolos.h:class EXP msrDoubleTremolo : public msrMeasureElement
10 msrEyeGlasses.h:class EXP msrEyeGlasses : public msrMeasureElement
11 msrFiguredBasses.h:class EXP msrFiguredBass : public msrMeasureElement
12 msrHarmonies.h:class EXP msrHarmony : public msrMeasureElement
13 msrHiddenMeasureAndBarLines.h:class EXP msrHiddenMeasureAndBarLine : public
  msrMeasureElement
14 msrInstruments.h:class EXP msrScordatura : public msrMeasureElement
15 msrInstruments.h:class EXP msrAccordionRegistration : public msrMeasureElement
16 msrInstruments.h:class EXP msrHarpPedalsTuning : public msrMeasureElement
17 msrInstruments.h:class EXP msrPedal : public msrMeasureElement
18 msrInstruments.h:class EXP msrDamp : public msrMeasureElement
19 msrInstruments.h:class EXP msrDampAll : public msrMeasureElement
20 msrKeys.h:class EXP msrKey : public msrMeasureElement
21 msrLyrics.h:class EXP msrSyllable : public msrMeasureElement
22 msrMusicXMLSpecifics.h:class EXP msrPrintLayout : public msrMeasureElement
23 msrRehearsalMarks.h:class EXP msrRehearsalMark : public msrMeasureElement
24 msrSegnos.h:class EXP msrSegno : public msrMeasureElement
25 msrDalSegnos.h:class EXP msrDalSegno : public msrMeasureElement
26 msrStavesDetails.h:class EXP msrStaffDetails : public msrMeasureElement
27 msrTempos.h:class EXP msrTempo : public msrMeasureElement
28 msrTimeSignatures.h:class EXP msrTimeSignature : public msrMeasureElement
29 msrTranspositions.h:class EXP msrOctaveShift : public msrMeasureElement
30 msrTranspositions.h:class EXP msrTransposition : public msrMeasureElement
31 msrVoiceStaffChanges.h:class EXP msrVoiceStaffChange : public msrMeasureElement

```

## 40.4 Appending measure elements to a measure

Appending music elements to a measure is done by method `msrMeasure::appendElementToMeasure ()`, defined in `src/formats/msr/msrMeasures.h/.cpp`:

```

1 void msrMeasure::appendElementToMeasure (const S_msrMeasureElement& elem)
2 {
3     int inputLineNumber =
4         elem->getInputLineNumber ();
5
6     #ifdef TRACING_IS_ENABLED
7         if (gGlobalTracingOahGroup->getTraceMeasures ()) {
8             gLogStream <<
9                 "Appending element " <<
10                 elem->asShortString () <<
11                 " to measure " <<
12                 asShortString () <<
13                 " in voice \"" <<
14                 fetchMeasureUpLinkToVoice ()->
15                 getVoiceName () <<
16                 "\", currentMeasureWholeNotesDuration = " <<
17                 fCurrentMeasureWholeNotesDuration <<
18                 ", line " << inputLineNumber <<
19                 std::endl;
20         }

```

```

21 #endif
22
23 // set elem's measure number
24 elem->
25     setMeasureNumber (
26         fBarLineUpLinkToMeasure->getMeasureNumber ());
27
28 // set elem's measure position
29 elem->
30     setMeasureElementMeasurePosition (
31         this,
32         fCurrentMeasureWholeNotesDuration,
33         "appendElementToMeasure()");
34
35 fMeasureElementsList.push_back (elem);
36
37 // take elem's sounding whole notes into account JMI ???
38 if (false) // JMI CAFE
39     incrementCurrentMeasureWholeNotesDuration (
40         inputLineNumber,
41         elem->
42             getMeasureElementSoundingWholeNotes ());
43 }

```

Here is how a harmony instance is appended to a measure:

```

1 void msrMeasure::appendHarmonyToMeasure (const S_msrHarmony& harmony)
2 {
3     int inputLineNumber =
4         harmony->getInputLineNumber ();
5
6 #ifdef TRACING_IS_ENABLED
7     if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
8         gLogStream <<
9             "Appending harmony " << harmony->asString () <<
10             " to measure " <<
11             this->asShortString () <<
12             " in segment '" <<
13             fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
14             "' in voice \"" <<
15             fMeasureUpLinkToSegment->
16                 getSegmentUpLinkToVoice ()->
17                 getVoiceName () <<
18             "\", currentMeasureWholeNotesDuration = " <<
19             fCurrentMeasureWholeNotesDuration <<
20             ", line " << inputLineNumber <<
21             std::endl;
22     }
23 #endif
24
25 // set harmony's measure number
26 harmony->
27     setMeasureNumber (
28         fBarLineUpLinkToMeasure->getMeasureNumber ());
29
30 // append the harmony to the measure elements list
31 // DON'T call 'appendElementToMeasure (harmony)':
32 // that would override harmony's measure position,
33 // which already has the correct value, thus:
34 fMeasureElementsList.push_back (harmony);
35
36 // get harmony sounding whole notes
37 Rational
38     harmonySoundingWholeNotes =
39         harmony->
40             getMeasureElementSoundingWholeNotes ();

```

```

41
42 // account for harmony duration in measure whole notes
43 incrementCurrentMeasureWholeNotesDuration (
44     inputLineNumber,
45     harmonySoundingWholeNotes);
46
47 // this measure contains music
48 fMeasureContainsMusic = true;
49 }

```

The task is simpler when appending a harmony to a measure clone, because the clone's harmony's measure number comes from the clone's original:

```

1 void msrMeasure::appendHarmonyToMeasureClone (const S_msrHarmony& harmony)
2 {
3     int inputLineNumber =
4         harmony->getInputLineNumber ();
5
6 #ifdef TRACING_IS_ENABLED
7     if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
8         gLogStream <<
9             "Appending harmony " << harmony->asString () <<
10             " to measure clone " <<
11             this->asShortString () <<
12             " in segment clone '" <<
13             fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
14             "' in voice clone '" <<
15             fMeasureUpLinkToSegment->
16                 getSegmentUpLinkToVoice ()->
17                 getVoiceName () <<
18             "\", currentMeasureWholeNotesDuration = " <<
19             fCurrentMeasureWholeNotesDuration <<
20             ", line " << inputLineNumber <<
21             std::endl;
22     }
23 #endif
24
25     ++gIndenter;
26
27     // append the harmony to the measure elements list
28     appendElementToMeasure (harmony);
29
30     // fetch harmony sounding whole notes
31     Rational
32         harmonySoundingWholeNotes =
33         harmony->
34             getMeasureElementSoundingWholeNotes ();
35
36     // account for harmony duration in measure whole notes
37     incrementCurrentMeasureWholeNotesDuration (
38         inputLineNumber,
39         harmonySoundingWholeNotes);
40
41     // this measure contains music
42     fMeasureContainsMusic = true;
43
44     --gIndenter;
45 }

```

## 40.5 Appending measures to a segment

Measures are appended to a segment by method `msrSegment::appendMeasureToSegment ()` in `src/formats/msr/msrSegments.h/.cpp`:

```

1 void msrSegment::appendMeasureToSegment (const S_msrMeasure& measure)
2 {
3     int inputLineNumber =
4         measure->getInputLineNumber ();
5
6     std::string measureNumber =
7         measure->getMeasureNumber ();
8
9     unsigned int segmentElementsListSize =
10         fSegmentElementsList.size ();
11
12     std::string currentMeasureNumber =
13         segmentElementsListSize == 0
14         ? ""
15         : fSegmentElementsList.back ()->getMeasureNumber ();
16
17 #ifdef TRACING_IS_ENABLED
18     if (gGlobalTracingOahGroup->getTraceMeasures ()) {
19         gLogStream <<
20             "Appending measure '" << measureNumber <<
21             "' to segment " << asString ();
22
23         if (fSegmentElementsList.size () == 0)
24             gLogStream <<
25                 ", as first measure";
26         else
27             gLogStream <<
28                 ", after measure number '" << currentMeasureNumber << "'";
29
30         gLogStream <<
31             "' in voice \"" <<
32             fSegmentUpLinkToVoice->getVoiceName () <<
33             "\" " <<
34             ", line " << measure->getInputLineNumber () <<
35             std::endl;
36     }
37 #endif
38
39     if (measureNumber == currentMeasureNumber) {
40         std::stringstream s;
41
42         s <<
43             "appending measure number '" << measureNumber <<
44             "' occurs twice in a row in segment " <<
45             asString () <<
46             " in voice \"" <<
47             fSegmentUpLinkToVoice->getVoiceName () <<
48             "\"";
49
50         // msrInternalWarning ( // JMI
51         msrInternalError (
52             gGlobalServiceRunData->getInputSourceName (),
53             inputLineNumber,
54             __FILE__, __LINE__,
55             s.str ());
56     }
57
58     // is measure the first one this segment?
59     if (segmentElementsListSize == 0) {
60         measure->
61             setMeasureFirstInSegmentKind (
62                 msrMeasureFirstInSegmentKind::kMeasureFirstInSegmentKindYes);
63     }
64     else {
65         measure->
66             setMeasureFirstInSegmentKind (

```

```

67     msrMeasureFirstInSegmentKind::kMeasureFirstInSegmentKindNo);
68 }
69
70 // is measure the first one in the voice?
71 // this is necessary for voice clones,
72 // which don't go down the part-staff-voice-segment hierarchy
73 if (! fSegmentUpLinkToVoice->getVoiceFirstMeasure ()) {
74     // yes, register it as such
75     fSegmentUpLinkToVoice->
76         setVoiceFirstMeasure (measure);
77
78     measure->
79         setMeasureFirstInVoice ();
80 }
81
82 // append measure to the segment
83 fSegmentElementsList.push_back (measure);
84 }

```

Calls to method `msrSegment::appendMeasureToSegment ()` occur in:

- method `msrSegment::createAMeasureAndAppendItToSegment ()` called from:  
method `msrVoice::createAMeasureAndAppendItToVoice ()`
- method `msrVoice::appendMeasureCloneToVoiceClone ()` called from:  
visitor method `msr2msrTranslator::visitStart (S_msrMeasure& elt)`
- method `msrMultipleFullBarRests::appendMeasureCloneToMultipleFullBarRests ()` called from:  
visitor method `msr2msrTranslator::visitStart (S_msrMeasure& elt)`
- method `msrVoice::appendMeasureCloneToVoiceClone ()` called from:  
visitor method `msr2lpsrTranslator::visitStart (S_msrMeasure& elt)`
- method `msrVoice::createNewLastSegmentFromItsFirstMeasureForVoice ()` called from:  
method `msrVoice::handleVoiceLevelRepeatStart ()`,  
method `msrVoice::handleVoiceLevelRepeatEndingStartWithoutExplicitStart ()`,  
method `msrVoice::handleVoiceLevelRepeatEndingStartWithExplicitStart ()`,  
method `msrVoice::createMeasureRepeatFromItsFirstMeasures ()`,  
method `msrVoice::appendPendingMeasureRepeatToVoice ()`,  
method `msrVoice::appendMultipleFullBarRestsToVoice ()`

## 40.6 Appending measures to a voice

Method `msrVoice::appendMeasureCloneToVoiceClone ()` does the job in `src/formats/msr/msrVoices.h/.cpp`.

```

1 S_msrMeasure msrVoice::createAMeasureAndAppendItToVoice (
2     int      inputLineNumber,
3     std::string measureNumber,
4     msrMeasureImplicitKind
5         measureImplicitKind)
6 {
7     fVoiceCurrentMeasureNumber = measureNumber;
8
9     #ifdef TRACING_IS_ENABLED
10     if (gGlobalTracingOahGroup->getTraceMeasures ()) {

```



```

11     gLogStream <<
12     "Creating measure '" <<
13     measureNumber <<
14     "' and appending it to voice \"" << getVoiceName () << "\" " <<
15     "', line " << inputLineNumber <<
16     std::endl;
17 }
18 #endif
19
20 fCallsCounter++;
21
22 if (
23 //     true
24 //     ||
25     false
26     &&
27     (
28         fCallsCounter == 2 && getVoiceName ()
29         ==
30         "Part_POne_HARMONIES_Staff_Voice_Eleven_HARMONIES"
31     )
32 ) { // POUSSE JMI
33     gLogStream <<
34     std::endl <<
35     "++++ createAMeasureAndAppendItToVoice() POUSSE, fCallsCounter: " << fCallsCounter
36     << " ++++" <<
37     std::endl;
38     this->print (gLogStream);
39     gLogStream <<
40     std::endl;
41 }
42
43 #ifdef TRACING_IS_ENABLED
44 if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
45     displayVoice (
46         inputLineNumber,
47         "createAMeasureAndAppendItToVoice() 1");
48 }
49 #endif
50
51 ++gIndenter;
52
53 // create the voice last segment if needed
54 if (! fVoiceLastSegment) {
55     createNewLastSegmentForVoice (
56         inputLineNumber,
57         "createAMeasureAndAppendItToVoice() 2");
58 }
59
60 // append a new measure with given number to voice last segment
61 S_msrMeasure
62 result =
63     fVoiceLastSegment->
64     createAMeasureAndAppendItToSegment (
65         inputLineNumber,
66         measureNumber,
67         measureImplicitKind);
68
69 // result is the new voice last appended measure
70 fVoiceLastAppendedMeasure = result;
71
72 #ifdef TRACING_IS_ENABLED
73 if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
74     displayVoice (
75         inputLineNumber,
76         "createAMeasureAndAppendItToVoice() 3");
77 }

```

```

77 #endif
78
79 --gIndenter;
80
81 return result;
82 }

```

## 40.7 Translating from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

Upon the first visit of `S_measure`, as class `msrMeasure` instance is created and append to the current part:

```

1 void mxsr2msrTranslator::visitStart (S_measure& elt)
2 {
3     // ... ..
4
5     // take this measure into account
6     ++fPartMeasuresCounter;
7
8     // ... ..
9
10    // implicit
11
12    /*
13     Measures with an implicit attribute set to "yes"
14     never display a measure number,
15     regardless of the measure-numbering setting.
16     *
17     The implicit attribute is set to "yes" for measures where
18     the measure number should never appear, such as pickup
19     measures and the last half of mid-measure repeats. The
20     value is "no" if not specified.
21    */
22
23    std::string
24        implicit =
25        elt->getAttributeValue ("implicit");
26
27    msrMeasureImplicitKind
28        measureImplicitKind =
29        msrMeasureImplicitKind::kMeasureImplicitKindNo; // default value
30
31    if (implicit == "yes") {
32        measureImplicitKind =
33        msrMeasureImplicitKind::kMeasureImplicitKindYes;
34    }
35    else if (implicit == "no") {
36        measureImplicitKind =
37        msrMeasureImplicitKind::kMeasureImplicitKindNo;
38    }
39    else {
40        if (implicit.size ()) {
41            std::stringstream s;
42
43            s <<
44            "implicit \"" << implicit <<
45            "\" is unknown";
46
47            musicxmlError (
48                gGlobalServiceRunData->getInputSourceName (),
49                inputLineNumber,
50                __FILE__, __LINE__,

```

```

51         s.str ());
52     }
53 }
54
55 // append a new measure to the current part
56 fCurrentPart->
57     createAMeasureAndAppendItToPart (
58         inputLineNumber,
59         fCurrentMeasureNumber,
60         measureImplicitKind);
61
62 // ... ..
63 }

```

Upon the second visit of `S_measure`, the last appended measure appended to the current part is finalized:

```

1 void mxsr2msrTranslator::visitEnd (S_measure& elt)
2 {
3     // ... ..
4
5     // finalize current measure in the part,
6     // to add skips if necessary and set measure kind
7     fCurrentPart->
8         finalizeLastAppendedMeasureInPart (
9             inputLineNumber);
10
11     // ... ..
12 }

```

## 40.8 Translating from MXSR to MSR

A new class `msrMeasure` instance is created in `src/passes/mxsr2msr/mxsr2msrTranslator.cpp` upon the first visit of `S_measure`:

```

1 // -----
2 void mxsr2msrTranslator::visitStart (S_measure& elt)
3 {
4     // ... ..
5
6     // append a new measure to the current part
7     fCurrentPart->
8         createAMeasureAndAppendItToPart (
9             inputLineNumber,
10            fCurrentMeasureNumber,
11            measureImplicitKind);
12
13     // ... ..
14 }

```

This can lead to several class `msrMeasure` instances being created, depending on the MusicXML data. Hence there is no notion of a current measure in this translator.

Method `msrPart::createAMeasureAndAppendItToPart ()` creates and appends a measure to the part harmonies and figured bass staves if relevant, and then cascade s to the part staves:

```

1 void msrPart::createAMeasureAndAppendItToPart (
2     int inputLineNumber,
3     std::string measureNumber,
4     msrMeasureImplicitKind
5     measureImplicitKind)
6 {
7     // ... ..

```

```
8
9 // set part current measure number
10 fPartCurrentMeasureNumber = measureNumber;
11
12 // create and append measure in all the staves
13 for (S_msrStaff staff : fPartAllStavesList) {
14     staff->
15         createAMeasureAndAppendItToStaff (
16             inputLineNumber,
17             measureNumber,
18             measureImplicitKind);
19 } // for
20
21 // ... ..
```

## 40.9 Translating from MSR to MSR

This is done in [src/passes/msr2msr/](#).

## 40.10 Translating from MSR to LPSR

This is done in [src/passes/msr2lpsr/](#).

## 40.11 Translating from LPSR to LilyPond

This is done in [src/passes/lpsr2lilypond/](#).

## Chapter 41

# Finalizations

### 41.1 Clones vs non-clones finalization

Finalizing clones may be simpler than finalizing a just-created and populated non-clone, due to the information available in the clone's original.

For example, method `msrMeasure::finalizeMeasure ()` delegates part of the job to methods handling the three kinds of voices, respectively:

```
1 void msrMeasure::finalizeMeasure (
2     int inputLineNumber,
3     msrMeasureRepeatContextKind measureRepeatContextKind,
4     std::string context)
5 {
6     // ... ..
7
8     S_msrVoice
9     voice =
10         fMeasureUpLinkToSegment ->
11         getSegmentUpLinkToVoice ();
12
13     // ... ..
14
15     // delegate to voice kind specific methods
16     switch (voice->getVoiceKind ()) {
17     case msrVoiceKind::kVoiceKindRegular:
18         finalizeRegularMeasure (
19             inputLineNumber,
20             measureRepeatContextKind,
21             context);
22         break;
23
24     case msrVoiceKind::kVoiceKindDynamics:
25         break;
26
27     case msrVoiceKind::kVoiceKindHarmonies:
28         finalizeHarmonyMeasure (
29             inputLineNumber,
30             measureRepeatContextKind,
31             context);
32         break;
33
34     case msrVoiceKind::kVoiceKindFiguredBass:
35         finalizeFiguredBassMeasure (
36             inputLineNumber,
37             measureRepeatContextKind,
38             context);
39         break;
```

```

40     } // switch
41
42     // ... ..
43 }

```

In the case of harmony and figured bass voices, padding may have to be added to obtain a complete measure. This does not happen for clones of such voices: the padding skips are in the original voice and will be visited and handled without anything special to be done.

## 41.2 The finalization methods

There is a set of virtual method `finalize* ()` methods in `MusicFormats`. There basic ones are:

- method `msrPart::finalizePart ()` and method `msrPart::finalizePartClone ()`, defined in `src/formats/msr/msrParts.h/.cpp`
- method `msrStaff::finalizeStaff ()`, defined in `src/formats/msr/msrStaves.h/.cpp`
- method `msrVoice::finalizeVoice ()`, defined in `src/formats/msr/msrVoices.h/.cpp`
- method `msrSegment::finalizeAllTheMeasuresOfSegment ()`, defined in `src/formats/msr/msrSegments.h/.cpp`
- method `msrMeasure::finalizeMeasure ()`, method `msrMeasure::finalizeMeasureClone ()` and method `msrMeasure::finalizeRegularMeasure ()`, defined in `src/formats/msr/msrMeasures.h/.cpp`
- method `msrChord::finalizeChord ()`, defined in `src/formats/msr/msrChords.h/.cpp`
- method `msrTuplets::finalizeTuplet ()`, defined in `src/formats/msr/msrTuplets.h/.cpp`
- method `mxsr2msrTranslator::finalizeTupletAndPopItFromTupletsStack ()`, defined in `src/passes/mxsr2msr/mxsr2msrTranslator.h.h/.cpp`
- method `msrMeasure::finalizeFiguredBassMeasure ()`, defined in `src/formats/msr/msrMeasures.h/.cpp`
- method `msrMeasure::finalizeHarmonyMeasure ()`, defined in `src/formats/msr/msrMeasures.h/.cpp`
- method `msr2bsrTranslator::finalizeCurrentMeasureClone ()`, defined in `src/passes/msr2bsr/(.h/.cppmsr2bsrTranslator)`
- method `mxsr2msrTranslator::finalizeCurrentChord ()`, defined in `src/passes/mxsr2msr/mxsr2msrTranslator.h/.cpp`

Handling repeats is rather complex in `MusicFormats`. Repeat ends are finalized with these methods:

- method `msrPart::finalizeRepeatEndInPart ()`,  
defined in `src/formats/msr/msrParts.h/.cpp`
- method `msrStaff::finalizeRepeatEndInStaff ()`,  
defined in `src/formats/msr/msrStaves.h/.cpp`
- method `msrVoice::finalizeRepeatEndInVoice ()`,  
defined in `src/formats/msr/msrVoices.h/.cpp`

There are also 'cascading' finalization methods: they propagate finalization going from class `msrPart` towards class `msrVoice`:

- method `msrPart::finalizeLastAppendedMeasureInPart ()`,  
defined in `src/formats/msr/msrParts.h/.cpp`
- method `msrStaff::finalizeLastAppendedMeasureInStaff ()`,  
defined in `src/formats/msr/msrStaves.h/.cpp`
- method `msrVoice::finalizeLastAppendedMeasureInVoice ()`,  
defined in `src/formats/msr/msrVoices.h/.cpp`
- method `msrPart::finalizePartAndAllItsMeasures ()`,  
defined in `src/formats/msr/msrParts.h/.cpp`
- method `msrVoice::finalizeVoiceAndAllItsMeasures ()`,  
defined in `src/formats/msr/msrVoices.h/.cpp`

A typical example of cascading finalization is method `msrPart::finalizePartAndAllItsMeasures ()`:

```

1 void msrPart::finalizePartAndAllItsMeasures (
2     int inputLineNumber)
3 {
4     #ifdef TRACING_IS_ENABLED
5         if (gGlobalTracingOahGroup->getTraceParts ()) {
6             gLogStream <<
7                 "Finalizing part clone " <<
8                 getPartCombinedName () <<
9                 ", line " << inputLineNumber <<
10                std::endl;
11        }
12    #endif
13
14    #ifdef TRACING_IS_ENABLED
15        if (gGlobalTracingOahGroup->getTraceVoices ()) {
16            gLogStream <<
17                "Finalizing all the measures of part \"" <<
18                getPartCombinedName () <<
19                "\", line " << inputLineNumber <<
20                std::endl;
21        }
22    #endif
23
24    for (
25        std::list<S_msrVoice>::const_iterator i = fPartAllVoicesList.begin ();
26        i != fPartAllVoicesList.end ();
27        ++i
28    ) {
29        const S_msrVoice& voice = (*i);
30
31        voice->
32            finalizeVoiceAndAllItsMeasures (
33                inputLineNumber);
34    } // for
35
36    // collect the part measures slices from the staves
37    collectPartMeasuresSlices (
38        inputLineNumber);
39 }

```



## 41.3 Finalizing parts

Method `msrPart::finalizePart ()` warns if there are no staves in the part, and otherwise handles them, including cascading to method `msrStaff::finalizeStaff ()`:

```

1 void msrPart::finalizePart (
2     int inputLineNumber)
3 {
4     #ifdef TRACING_IS_ENABLED
5         if (gGlobalTracingOahGroup->getTraceParts ()) {
6             gLogStream <<
7                 "Finalizing part " <<
8                 getPartCombinedName () <<
9                 ", line " << inputLineNumber <<
10                std::endl;
11        }
12    #endif
13
14    ++gIndenter;
15
16    if (! getPartStaveNumbersToStavesMap.size ()) {
17        std::stringstream s;
18
19        s <<
20            "Part " <<
21            getPartCombinedName () <<
22            " appears in the part list, but doesn't contain any stave";
23
24        musicxmlWarning (
25            gGlobalServiceRunData->getInputSourceName (),
26            inputLineNumber,
27            s.str ());
28    }
29
30    else {
31        // sort the staves to have harmonies above and figured bass below the part
32        fPartAllStavesList.sort (
33            compareStavesToHaveFiguredBassesBelowCorrespondingPart);
34
35        // finalize the staves
36        for (
37            std::map<int, S_msrStaff>::const_iterator i =
38            getPartStaveNumbersToStavesMap.begin ();
39            i != getPartStaveNumbersToStavesMap.end ();
40            ++i
41        ) {
42            S_msrStaff staff = (*i).second;
43
44            staff->
45                finalizeStaff (
46                    inputLineNumber);
47        } // for
48    }
49
50    // set score instrument names max lengths if relevant
51    setPartInstrumentNamesMaxLengths ();
52
53    // collect the part measures slices from the staves
54    collectPartMeasuresSlices (
55        inputLineNumber);
56
57    --gIndenter;
58 }

```

## 41.4 Finalizing staves

Method `msrStaff::finalizeStaff ()` cascade s to method `msrVoice::finalizeVoice ()` and collects the staff measures slices:

```

1 void msrStaff::finalizeStaff (int inputLineNumber)
2 {
3     #ifndef TRACING_IS_ENABLED
4         if (gGlobalTracingOahGroup->getTraceStaves ()) {
5             gLogStream <<
6                 "Finalizing staff \"" <<
7                 getStaffName () << "\" " <<
8                 ", line " << inputLineNumber <<
9                 std::endl;
10        }
11    #endif
12
13    ++gIndenter;
14
15    // finalize the voices
16    #ifndef TRACING_IS_ENABLED
17        if (gGlobalTracingOahGroup->getTraceVoices ()) {
18            gLogStream <<
19                "Finalizing the voices in staff \"" <<
20                getStaffName () << "\" " <<
21                ", line " << inputLineNumber <<
22                std::endl;
23        }
24    #endif
25
26    for (
27        std::map<int, S_msrVoice>::const_iterator i =
28            fStaffVoiceNumbersToAllVoicesMap.begin ();
29        i != fStaffVoiceNumbersToAllVoicesMap.end ();
30        ++i
31    ) {
32        S_msrVoice
33            voice = (*i).second;
34
35        voice->
36            finalizeVoice (
37                inputLineNumber);
38    } // for
39
40    // collect the staff measures slices from the voices
41    collectStaffMeasuresSlices (
42        inputLineNumber);
43
44    --gIndenter;
45 }

```

## 41.5 Finalizing voices

Method `msrVoice::finalizeVoice ()` handles pending repeats if any and collects the voice measures into a flat list. It does not, however, cascade to finalizing the voice repeats and measures.

```

1 void msrVoice::finalizeVoice (
2     int inputLineNumber)
3 {
4     // ... ..
5
6     if (fVoiceHasBeenFinalized) {

```

```

7      std::stringstream s;
8
9      s <<
10     "Attempting to finalize voice \"" <<
11     asShortString () <<
12     "\" more than once";
13
14     msrInternalError (
15         gGlobalServiceRunData->getInputSourceName (),
16         fInputLineNumber,
17         __FILE__, __LINE__,
18         s.str ());
19 }
20
21 // set part shortest note duration if relevant
22 S_msrPart
23     voicePart =
24         fetchVoiceUpLinkToPart ();
25
26 Rational
27     partShortestNoteDuration =
28         voicePart->
29             getPartShortestNoteDuration ();
30
31 // ... ..
32
33 if (fVoiceShortestNoteDuration < partShortestNoteDuration) {
34     // set the voice part shortest note duration
35     voicePart->
36         setPartShortestNoteDuration (
37             fVoiceShortestNoteDuration);
38
39     // set the voice part shortest note tuplet factor // JMI
40     voicePart->
41         setPartShortestNoteTupletFactor (
42             fVoiceShortestNoteTupletFactor);
43 }
44
45 // is this voice totally empty? this should be rare...
46 if (
47     fVoiceInitialElementsList.size () == 0
48     &&
49     fVoiceLastSegment->getSegmentElementsList ().size () == 0
50 ) {
51     std::stringstream s;
52
53     s <<
54     "Voice \"" <<
55     getVoiceName () <<
56     "\" is totally empty, no contents ever specified for it" <<
57     std::endl;
58
59     musicxmlWarning (
60         gGlobalServiceRunData->getInputSourceName (),
61         inputLineNumber,
62         s.str ());
63 }
64
65 // are there pending repeats in the voice repeats stack???
66 unsigned int voicePendingRepeatDescrsStackSize =
67     fVoicePendingRepeatDescrsStack.size ();
68
69 // ... ..
70
71 // collect the voice measures into the flat list
72 collectVoiceMeasuresIntoFlatList (
73     inputLineNumber);

```

```

74
75     fVoiceHasBeenFinalized = true;
76
77     // ... ..
78 }

```

## 41.6 Finalizing repeats

## 41.7 Finalizing measures

Method `msrMeasure::finalizeMeasure ()` is not cascaded. It delegates finalization to voice kind specific methods presented in the subsections below, handles pending repeats if any, and assigns positions in the measure to the measure's elements:

```

1 void msrMeasure::finalizeMeasure (
2     int                inputLineNumber,
3     msrMeasureRepeatContextKind measureRepeatContextKind,
4     std::string        context)
5 {
6     if (fMeasureHasBeenFinalized) {
7         std::stringstream s;
8
9         s <<
10        "Attempting to finalize measure " <<
11        this->asShortString () <<
12        " more than once in segment '" <<
13        fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
14        "', context: " << context <<
15        "', measureFinalizationContext: " << fMeasureFinalizationContext <<
16        " in voice \"" <<
17        fMeasureUpLinkToSegment->
18        getSegmentUpLinkToVoice ()->
19        getVoiceName () <<
20        "\" (" << context << ")" <<
21        ", line " << inputLineNumber;
22
23        // ... ..
24
25        msrInternalWarning (
26            gGlobalServiceRunData->getInputSourceName (),
27            fInputLineNumber,
28            s.str ());
29    }
30
31    else {
32        S_msrVoice
33        voice =
34            fMeasureUpLinkToSegment->
35            getSegmentUpLinkToVoice ();
36
37        // ... ..
38
39        // delegate to voice kind specific methods
40        switch (voice->getVoiceKind ()) {
41            case msrVoiceKind::kVoiceKindRegular:
42                finalizeRegularMeasure (
43                    inputLineNumber,
44                    measureRepeatContextKind,
45                    context);
46                break;
47
48            case msrVoiceKind::kVoiceKindDynamics:

```

```

49         break;
50
51     case msrVoiceKind::kVoiceKindHarmonies:
52         finalizeHarmonyMeasure (
53             inputLineNumber,
54             measureRepeatContextKind,
55             context);
56         break;
57     case msrVoiceKind::kVoiceKindFiguredBass:
58         finalizeFiguredBassMeasure (
59             inputLineNumber,
60             measureRepeatContextKind,
61             context);
62         break;
63 } // switch
64
65 // position in voice
66 Rational
67 voicePosition =
68     fetchMeasureUpLinkToVoice ()->
69     getCurrentVoicePosition ();
70
71 // assign measure' elements measure position
72 for (
73     std::list<S_msrMeasureElement>::const_iterator i = fMeasureElementsList.begin ();
74     i != fMeasureElementsList.end ();
75     ++i
76 ) {
77     S_msrMeasureElement measureElement = (*i);
78
79     measureElement->
80         setMeasureElementVoicePosition (
81             voicePosition,
82             "finalizeMeasure()");
83 } // for
84
85 // register finalization
86 fMeasureHasBeenFinalized = true;
87 fMeasureFinalizationContext = context;
88 }
89 }

```

### 41.7.1 Finalizing regular measures

```

1 void msrMeasure::finalizeRegularMeasure (
2     int inputLineNumber,
3     msrMeasureRepeatContextKind measureRepeatContextKind,
4     std::string context)
5 {
6     // fetch the regular voice
7     S_msrVoice
8     voice =
9         fMeasureUpLinkToSegment->
10         getSegmentUpLinkToVoice ();
11
12     // fetch the regular voice's part
13     S_msrPart
14     regularPart =
15         voice->
16         fetchVoiceUpLinkToPart ();
17
18     mfAssert (
19         __FILE__, __LINE__,
20         regularPart != nullptr,

```

```

21     "regularPart is null");
22
23     if (false) { // JMI
24         gLogStream <<
25             "---> regularPart: " <<
26             std::endl;
27
28         ++gIndenter;
29         gLogStream <<
30             regularPart <<
31             std::endl;
32         --gIndenter;
33         gLogStream << std::endl;
34     }
35
36     Rational
37     measureWholeNotesDurationFromPartMeasuresVector =
38         regularPart->
39             getPartMeasuresWholeNotesDurationsVector () [
40                 fMeasureOrdinalNumberInVoice - 1 ];
41
42 #ifdef TRACING_IS_ENABLED
43     if (gGlobalTracingOahGroup->getTraceMeasures ()) {
44         gLogStream <<
45             "Finalizing regular measure " <<
46             this->asShortString () <<
47             " in segment '" <<
48             fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
49             "' in regular voice \"" <<
50             voice->getVoiceName () <<
51             "\" (" << context << ")" <<
52             ", measureWholeNotesDurationFromPartMeasuresVector: " <<
53             measureWholeNotesDurationFromPartMeasuresVector <<
54             ", line " << inputLineNumber <<
55             std::endl;
56     }
57 #endif
58
59     ++gIndenter;
60
61 #ifdef TRACING_IS_ENABLED
62     if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
63         displayMeasure (
64             inputLineNumber,
65             "finalizeRegularMeasure() 1");
66     }
67 #endif
68
69     padUpToPositionAtTheEndOfTheMeasure (
70         inputLineNumber,
71         measureWholeNotesDurationFromPartMeasuresVector);
72
73     // register this measures's length in the part
74     S_msrPart
75     part =
76         this->fetchMeasureUpLinkToPart ();
77
78     part->
79         registerOrdinalMeasureNumberWholeNotesDuration (
80             inputLineNumber,
81             fMeasureOrdinalNumberInVoice,
82             fCurrentMeasureWholeNotesDuration);
83
84     // determine the measure kind and purist number
85     determineMeasureKindAndPuristNumber (
86         inputLineNumber,
87         measureRepeatContextKind);

```

```

88
89 // pad measure up to whole measure whole notes high tide JMI ???
90 switch (fMeasureKind) {
91     case msrMeasureKind::kMeasureKindCadenza:
92         break;
93
94     case msrMeasureKind::kMeasureKindOvercomplete:
95     case msrMeasureKind::kMeasureKindAnacrusis:
96     case msrMeasureKind::kMeasureKindRegular:
97     case msrMeasureKind::kMeasureKindIncompleteStandalone: // JMI
98     case msrMeasureKind::kMeasureKindIncompleteLastInRepeatCommonPart: // JMI
99     case msrMeasureKind::kMeasureKindIncompleteLastInRepeatHookedEnding: // JMI
100    case msrMeasureKind::kMeasureKindIncompleteLastInRepeatHooklessEnding: // JMI
101    case msrMeasureKind::kMeasureKindIncompleteNextMeasureAfterCommonPart: // JMI
102    case msrMeasureKind::kMeasureKindIncompleteNextMeasureAfterHookedEnding: // JMI
103    case msrMeasureKind::kMeasureKindIncompleteNextMeasureAfterHooklessEnding: // JMI
104        break;
105
106    case msrMeasureKind::kMeasureKindUnknown:
107        // JMI ???
108        break;
109
110    case msrMeasureKind::kMeasureKindMusicallyEmpty:
111    {
112        /* JMI
113         */
114    }
115    break;
116 } // switch
117
118 // is there a single note or rest occupying the full measure?
119 if (fMeasureLongestNote) {
120     if (
121         fMeasureLongestNote-> getMeasureElementSoundingWholeNotes ()
122         ==
123         fFullMeasureWholeNotesDuration
124     ) {
125 #ifdef TRACING_IS_ENABLED
126     if (gGlobalTracingOahGroup->getTraceMeasures ()) {
127         gLogStream <<
128             "Note '" <<
129             fMeasureLongestNote->asShortString () <<
130             "' occupies measure " <<
131             this->asShortString () <<
132             " fully in segment '" <<
133             fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
134             "' in voice \"" <<
135             voice->getVoiceName () <<
136             "\", line " << inputLineNumber <<
137             std::endl;
138     }
139 #endif
140
141     fMeasureLongestNote->
142         setNoteOccupiesAFullMeasure ();
143     }
144 }
145
146 #ifdef TRACING_IS_ENABLED
147 if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
148     displayMeasure (
149         inputLineNumber,
150         "finalizeRegularMeasure() 2");
151 }
152 #endif
153
154 --gIndenter;

```

155 }

### 41.7.2 Finalizing harmonies measures

```

1 void msrMeasure::finalizeHarmonyMeasure (
2     int                inputLineNumber,
3     msrMeasureRepeatContextKind measureRepeatContextKind,
4     std::string        context)
5 {
6     // fetch the harmonies voice
7     S_msrVoice
8     harmoniesVoice =
9         fMeasureUpLinkToSegment->
10         getSegmentUpLinkToVoice ();
11
12     // fetch the harmonies part
13     S_msrPart
14     harmoniesPart =
15         harmoniesVoice->
16         fetchVoiceUpLinkToPart ();
17
18     mfAssert (
19         __FILE__, __LINE__,
20         harmoniesPart != nullptr,
21         "harmoniesPart is null");
22
23 #ifdef TRACING_IS_ENABLED
24     if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
25         gLogStream <<
26             "Finalizing harmonies measure " <<
27             this->asShortString () <<
28             " in segment '" <<
29             fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
30             "' in harmonies voice \"" <<
31             harmoniesVoice->getVoiceName () <<
32             "\" (" << context << ")" <<
33             ", line " << inputLineNumber <<
34             std::endl;
35     }
36 #endif
37     ++gIndenter;
38
39 #ifdef TRACING_IS_ENABLED
40     if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
41         displayMeasure (
42             inputLineNumber,
43             "finalizeHarmonyMeasure() 1");
44     }
45 #endif
46
47 #ifdef TRACING_IS_ENABLED
48     // get the harmoniesPart number of measures
49     int
50     harmoniesPartNumberOfMeasures =
51         harmoniesPart->
52         getPartNumberOfMeasures ();
53
54     if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
55         gLogStream <<
56             "fMeasureOrdinalNumberInVoice = " <<
57             fMeasureOrdinalNumberInVoice <<
58             ", harmoniesPartNumberOfMeasures = " <<
59             harmoniesPartNumberOfMeasures <<

```



```

61     std::endl;
62 }
63 #endif
64
65 // the measureWholeNotesDuration has to be computed
66 Rational
67     measureWholeNotesDurationFromPartMeasuresVector =
68     harmoniesPart->
69     getPartMeasuresWholeNotesDurationsVector () [
70         fMeasureOrdinalNumberInVoice - 1 ];
71
72 // handle the harmonies in this measure
73 finalizeTheHarmoniesInHarmoniesMeasure (
74     inputLineNumber,
75     context);
76
77 // pad the measure up to measureWholeNotesDurationFromPartMeasuresVector
78 padUpToPositionAtTheEndOfTheMeasure (
79     inputLineNumber,
80     measureWholeNotesDurationFromPartMeasuresVector);
81
82 // determine the measure kind and purist number
83 determineMeasureKindAndPuristNumber (
84     inputLineNumber,
85     measureRepeatContextKind);
86
87 #ifdef TRACING_IS_ENABLED
88     if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
89         displayMeasure (
90             inputLineNumber,
91             "finalizeHarmonyMeasure() 2");
92     }
93 #endif
94
95     --gIndenter;
96 }

```

### 41.7.3 Finalizing figured bass measures

```

1 void msrMeasure::finalizeFiguredBassMeasure (
2     int inputLineNumber,
3     msrMeasureRepeatContextKind measureRepeatContextKind,
4     std::string context)
5 {
6     // fetch the figured bass voice
7     S_msrVoice
8         figuredBassVoice =
9         fMeasureUpLinkToSegment->
10         getSegmentUpLinkToVoice ();
11
12     // fetch the figured bass part
13     S_msrPart
14         figuredBassPart =
15         figuredBassVoice->
16         fetchVoiceUpLinkToPart ();
17
18     mfAssert (
19         __FILE__, __LINE__,
20         figuredBassPart != nullptr,
21         "figuredBassPart is null");
22
23 #ifdef TRACING_IS_ENABLED
24     if (gGlobalTracingOahGroup->getTraceFiguredBasses ()) {
25         gLogStream <<

```

```

26     "Finalizing figured bass measure " <<
27     this->asShortString () <<
28     " in segment '" <<
29     fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
30     "' in figured bass voice \"" <<
31     figuredBassVoice->getVoiceName () <<
32     "\" (" << context << ")" <<
33     ", line " << inputLineNumber <<
34     std::endl;
35 }
36 #endif
37
38 ++gIndenter;
39
40 #ifdef TRACING_IS_ENABLED
41 if (gGlobalTracingOahGroup->getTraceFiguredBassesDetails ()) {
42     displayMeasure (
43         inputLineNumber,
44         "finalizeFiguredBassMeasure() 1");
45 }
46 #endif
47
48 #ifdef TRACING_IS_ENABLED
49 // get the figuredBassPart number of measures
50 int
51 figuredBassPartNumberOfMeasures =
52     figuredBassPart->
53     getPartNumberOfMeasures ();
54
55 if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
56     gLogStream <<
57     "fMeasureOrdinalNumberInVoice = " <<
58     fMeasureOrdinalNumberInVoice <<
59     ", figuredBassPartNumberOfMeasures = " <<
60     figuredBassPartNumberOfMeasures <<
61     std::endl;
62 }
63 #endif
64
65 // the measureWholeNotesDuration has to be computed
66 Rational
67 measureWholeNotesDuration =
68     figuredBassPart->
69     getPartMeasuresWholeNotesDurationsVector () [
70         fMeasureOrdinalNumberInVoice - 1 ];
71
72 // handle the figured bass elements in this measure
73 finalizeFiguredBassesInFiguredBassMeasure (
74     inputLineNumber,
75     context);
76
77 // pad the measure up to fFullMeasureWholeNotesDuration
78 padUpToPositionAtTheEndOfTheMeasure (
79     inputLineNumber,
80     measureWholeNotesDuration);
81
82 // determine the measure kind and purist number
83 determineMeasureKindAndPuristNumber (
84     inputLineNumber,
85     measureRepeatContextKind);
86
87 #ifdef TRACING_IS_ENABLED
88 if (gGlobalTracingOahGroup->getTraceFiguredBassesDetails ()) {
89     displayMeasure (
90         inputLineNumber,
91         "finalizeFiguredBassMeasure() 2");
92 }

```

```
93 #endif
94
95     --gIndenter;
96 }
```

## 41.8 Determining measure positionss

## Chapter 42

# Tempos handling

Tempos are presented at section [19.18](#) [Tempos], page [175](#).

**42.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**42.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**42.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**42.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**42.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 43

# Notes handling

Notes are presented at section ?? [Notes], page ??.

**43.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**43.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**43.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**43.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**43.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 44

# Segments handling

Segments are presented at section [19.37](#) [Segments], page [194](#).

The segments concept used by MusicFormats to describe music scores is not apparent to the users of GUI applications, in which music elements are *drawn* on the page. Their need is inherent to the representation of repeats, which contain music elements sequences (the segments) and even other repeats.

ALL SEGMENTS HANDLING in MusicFormats IS DONE INTERNALLY: the class `msrSegment` instances are created in voices and repeats BEHIND THE CURTAINS.

### 44.1 Segments creation

Instances of class `msrSegment` are created at four places in `src/formats/msr/msrVoices.cpp`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'msrSegment::create ('  
   *.cpp  
2 msrSegments.cpp:S_msrSegment msrSegment::create (  
3 msrSegments.cpp:      msrSegment::create (  
4 msrSegments.cpp:      msrSegment::create (  
5 msrVoices.cpp:      msrSegment::create (  
6 msrVoices.cpp:      msrSegment::create (  
7 msrVoices.cpp:      msrSegment::create (  
8 msrVoices.cpp:      msrSegment::create (  
9
```

Calls to method `msrSegment::createSegmentNewbornClone ()` occurs only when visiting class `msrSegment` instances in passes:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/passes > grep createSegmentNewbornClone  
   */*  
2 msr2bsr/msr2bsrTranslator.cpp:      elt->createSegmentNewbornClone (  
3 msr2lpsr/msr2lpsrTranslator.cpp:      elt->createSegmentNewbornClone (  
4 msr2msr/msr2msrTranslator.cpp:      elt->createSegmentNewbornClone (  
5
```

Method `msrSegment::createSegmentDeepClone ()` is not used at the time of this writing.

Explicit segments creation is thus entirely done in methods inside `src/formats/msr/msrVoices.cpp`: the passes are not aware of this happening.

The first occurrence of method `msrSegment::create ()` is in method `msrVoice::initializeVoice ()`: when a voice is created, a segment is created and stored in its `fVoiceLastSegment` if requested :

```

1 void msrVoice::initializeVoice (
2     msrVoiceCreateInitialLastSegmentKind
3     voiceCreateInitialLastSegmentKind)
4 {
5     // ... ..
6
7     // create the initial last segment if requested
8     switch (voiceCreateInitialLastSegmentKind) {
9         case msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes:
10            // sanity check // JMI LAST
11            mfAssert (
12                __FILE__, __LINE__,
13                fVoiceLastSegment == nullptr,
14                "fVoiceLastSegment is null");
15
16            // create the last segment
17            fVoiceLastSegment =
18                msrSegment::create (
19                    fInputLineNumber,
20                    this);
21
22            if (! fVoiceFirstSegment) {
23                fVoiceFirstSegment = fVoiceLastSegment;
24            }
25            break;
26        case msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentNo:
27            break;
28    } // switch
29
30    // ... ..
31 };

```

Method `msrVoice::createMeasureRepeatFromItsFirstMeasures ()` is presented in section 46 [Measure repeats handling], page 304, and the remaining two are presented in the next sections.

#### 44.1.1 Creating a new last segment for a voice

There is method `msrVoice::createNewLastSegmentForVoice ()`, called at many places in `src/formats/msr/msrVoices.cpp`:

```

1 void msrVoice::createNewLastSegmentForVoice (
2     int     inputLineNumber,
3     const std::string& context)
4 {
5     // ... ..
6
7     // create the last segment
8     fVoiceLastSegment =
9         msrSegment::create (
10             inputLineNumber,
11             this);
12
13     if (! fVoiceFirstSegment) {
14         fVoiceFirstSegment = fVoiceLastSegment;
15     }
16
17     // ... ..
18 }

```

The calls to method `msrVoice::createNewLastSegmentForVoice ()` are in:

- method `msrVoice::createAMeasureAndAppendItToVoice ()`
- method `msrVoice::appendStaffDetailsToVoice ()`
- method `msrVoice::addGraceNotesGroupBeforeAheadOfVoiceIfNeeded ()`
- method `msrVoice::handleVoiceLevelRepeatStart ()`
- method `msrVoice::handleVoiceLevelRepeatEndWithoutStart ()`
- method `msrVoice::handleVoiceLevelContainingRepeatEndWithoutStart ()`
- method `msrVoice::handleVoiceLevelRepeatEndWithStart ()`
- method `msrVoice::handleVoiceLevelRepeatEndingStartWithoutExplicitStart ()`
- method `msrVoice::handleVoiceLevelRepeatEndingStartWithExplicitStart ()`
- method `msrVoice::handleMultipleFullBarRestsStartInVoiceClone ()`
- method `msrVoice::handleHooklessRepeatEndingEndInVoice ()`
- method `msrVoice::appendBarLineToVoice ()`
- method `msrVoice::appendSegnoToVoice ()`
- method `msrVoice::appendCodaToVoice ()`
- method `msrVoice::appendEyeGlassesToVoice ()`
- method `msrVoice::appendPedalToVoice ()`
- method `msrVoice::appendDampToVoice ()`
- method `msrVoice::appendDampAllToVoice ()`

#### 44.1.2 Creating a new last segment for a voice from its first measure

Method `msrVoice::createNewLastSegmentFromItsFirstMeasureForVoice ()` is used at several places in [src/formats/msr/msrVoices.cpp](#):

```

1 void msrVoice::createNewLastSegmentFromItsFirstMeasureForVoice (
2     int          inputLineNumber,
3     S_msrMeasure firstMeasure,
4     std::string  context)
5 {
6     // create the last segment
7     fVoiceLastSegment =
8         msrSegment::create (
9             inputLineNumber,
10            this);
11
12     if (! fVoiceFirstSegment) {
13         fVoiceFirstSegment = fVoiceLastSegment;
14     }
15
16     // ... ..
17

```



```

18 // append firstMeasure to fVoiceLastSegment
19 fVoiceLastSegment->
20     appendMeasureToSegment (firstMeasure);
21
22 // firstMeasure is the new voice last appended measure
23 fVoiceLastAppendedMeasure = firstMeasure;
24
25 // is firstMeasure the first one in the voice?
26 if (! fVoiceFirstMeasure) {
27     // yes, register it as such
28     setVoiceFirstMeasure (
29         firstMeasure);
30
31     firstMeasure->
32         setMeasureFirstInVoice ();
33 }
34
35 // ... ..
36 }

```

All the uses of this method concern repeats (section 48 [Repeats handling], page 306), measure repeats (section 46 [Measure repeats handling], page 304) and multiple full-bar rests (section ?? [Full-bar rests handling], page ??).

## 44.2 Appending measures to a segment

Method `msrSegment::assertSegmentElementsListIsNotEmpty ()` is called as a sanity check by many methods in `src/formats/msr/msrSegments.cpp`:

```

1 void msrSegment::assertSegmentElementsListIsNotEmpty (
2     int inputLineNumber) const
3 {
4     if (! fSegmentElementsList.size ()) {
5 #ifdef TRACING_IS_ENABLED
6         if (
7             gGlobalTracingOahGroup->getTraceMeasuresDetails ()
8             ||
9             gGlobalTracingOahGroup->getTraceSegmentsDetails ()
10            ||
11            gGlobalTracingOahGroup->getTraceRepeatsDetails ()
12        ) {
13            fSegmentUpLinkToVoice->
14                displayVoiceRepeatsStackMultipleFullBarRestsMeasureRepeatAndVoice (
15                inputLineNumber,
16                "assertSegmentElementsListIsNotEmpty()");
17        }
18 #endif
19
20        gLogStream <<
21            "assertSegmentElementsListIsNotEmpty()" <<
22            ", fSegmentElementsList is empty" <<
23            ", segment: " <<
24            this->asString () <<
25            ", in voice \"" <<
26            fSegmentUpLinkToVoice->getVoiceName () <<
27            "\" " <<
28            ", line " << inputLineNumber <<
29            std::endl;
30
31        mfAssert (
32            __FILE__, __LINE__,
33            false,
34            ", fSegmentElementsList is empty");

```

```

35 }
36 }

```

One such call is:

```

1 void msrSegment::appendKeyToSegment (
2     const S_msrKey& key)
3 {
4     #ifdef TRACING_IS_ENABLED
5         if (gGlobalTracingOahGroup->getTraceKeys ()) {
6             gLogStream <<
7                 "Appending key " << key->asString () <<
8                 " to segment " << asString () <<
9                 ", in voice \"" <<
10                fSegmentUpLinkToVoice->getVoiceName () <<
11                "\" " <<
12                std::endl;
13        }
14    #endif
15
16    // sanity check
17    assertSegmentElementsListIsNotEmpty (
18        key->getInputLineNumber ());
19
20    ++gIndenter;
21
22    // register key in segments's current measure
23    fSegmentElementsList.back ()->
24        appendKeyToMeasure (key);
25
26    --gIndenter;
27 }

```

## 44.3 Translating from MXSR to MSR

## 44.4 Translating from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

## 44.5 Translating from MSR to MSR

This is done in `src/passes/msr2msr/`.

## 44.6 Translating from MSR to LPSR

This is done in `src/passes/msr2lpsr/`.

## 44.7 Translating from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

## Chapter 45

# Beat repeats handling

Beat repeats are presented at section [19.32](#) [Beat repeats], page [189](#).

**45.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**45.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**45.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**45.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**45.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 46

# Measure repeats handling

Measure repeats are presented at section [19.33](#) [Measure repeats], page [191](#).

### 46.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

```
method msrVoice::createMeasureRepeatFromItsFirstMeasures ():
```

### 46.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

### 46.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

### 46.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

### 46.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 47

# Multiple full-bar rests handling

Multiple full-bar rests are presented at section [19.34](#) [Multiple full-bar rests], page [192](#).

## Chapter 48

# Repeats handling

Repeats are presented at section [19.36](#) [Repeats], page [192](#).

### 48.1 Translating repeats from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

The tough part is to handle MusicXML `<barLine/>` markups, since they are meant for drawing, and do not structure repeats as such.

Recognizing the structure of repeat relies on the attributes of the barLines. The following enumeration types are defined in `src/formats/msr/msrBarLines.h` local to class `msrBarLine`:

```
1 // location
2 enum class msrBarLineLocationKind {
3     kBarLineLocationNone,
4
5     kBarLineLocationLeft,
6     kBarLineLocationMiddle,
7     kBarLineLocationRight // by default
8 };
```

```
1 // style
2 enum class msrBarLineStyleKind {
3     kBarLineStyleNone,
4
5     kBarLineStyleRegular, // by default
6
7     kBarLineStyleDotted, kBarLineStyleDashed, kBarLineStyleHeavy,
8     kBarLineStyleLightLight, kBarLineStyleLightHeavy,
9     kBarLineStyleHeavyLight, kBarLineStyleHeavyHeavy,
10    kBarLineStyleTick, kBarLineStyleShort
11 };
```

```
1 // repeat direction
2 enum class msrBarLineRepeatDirectionKind {
3     kBarLineRepeatDirectionNone,
4     kBarLineRepeatDirectionForward, kBarLineRepeatDirectionBackward
5 };
```

```

1 // ending type
2 enum class msrBarLineEndingTypeKind {
3     kBarLineEndingTypeNone,
4
5     msrBarLineEndingTypeKind::kBarLineEndingTypeStart,
6     msrBarLineEndingTypeKind::kBarLineEndingTypeStop,
7     msrBarLineEndingTypeKind::kBarLineEndingTypeDiscontinue
8 };

```

```

1 // category
2 enum class msrBarLineCategoryKind {
3     kBarLineCategory_UNKNOWN,
4
5     kBarLineCategoryStandalone,
6
7     kBarLineCategoryRepeatStart, kBarLineCategoryRepeatEnd,
8
9     kBarLineCategoryHookedEndingStart, kBarLineCategoryHookedEndingEnd,
10    kBarLineCategoryHooklessEndingStart, kBarLineCategoryHooklessEndingEnd
11 };

```

```

1 // segno
2 enum class msrBarLineHasSegnoKind {
3     kBarLineHasSegnoYes, kBarLineHasSegnoNo
4 };

```

```

1 // coda
2 enum class msrBarLineHasCodaKind {
3     kBarLineHasCodaYes, kBarLineHasCodaNo
4 };

```

```

1 // repeat winged
2 enum class msrBarLineRepeatWingedKind {
3     kBarLineRepeatWingedNone,
4
5     kBarLineRepeatWingedStraight, kBarLineRepeatWingedCurved,
6     kBarLineRepeatWingedDoubleStraight, kBarLineRepeatWingedDoubleCurved
7 };

```

The attributes of <barline/> are deciphered upon the first visit of S\_barline in [src/passes/mxsr2msr/mxsr2msrTranslator.cpp](#):

```

1 void mxsr2msrTranslator::visitStart ( S_barline& elt )
2 {
3     // ... ..
4
5     // location
6
7     {
8         std::string
9             location =
10             elt->getAttributeValue ( "location" );
11
12         fCurrentBarLineLocationKind =
13             msrBarLineLocationKind::kBarLineLocationRight; // by default
14
15         if (location == "left") {
16             fCurrentBarLineLocationKind = msrBarLineLocationKind::kBarLineLocationLeft;
17         }
18         else if (location == "middle") {
19             fCurrentBarLineLocationKind = msrBarLineLocationKind::kBarLineLocationMiddle;
20         }
21         else if (location == "right") {
22             fCurrentBarLineLocationKind = msrBarLineLocationKind::kBarLineLocationRight;

```

```

23     }
24     else {
25         std::stringstream s;
26
27         s <<
28         "barLine location \"" << location <<
29         "\" is unknown, using 'right' by default";
30
31         // JMI    musicxmlError (
32             musicxmlWarning (
33                 gGlobalServiceRunData->getInputSourceName (),
34                 inputLineNumber,
35                 //    __FILE__, __LINE__,
36                 s.str ());
37     }
38 }
39
40 fOnGoingBarLine = true;
41 }

```

Then the class `msrBarLine` instance is created upon the second visit of `S_barline`:

```

1 void mxsr2msrTranslator::visitEnd ( S_barline& elt )
2 {
3     // ... ..
4
5     // create the barLine
6     S_msrBarLine
7     barLine =
8         msrBarLine::create (
9             inputLineNumber,
10            fCurrentBarLineLocationKind,
11            fCurrentBarLineStyleKind,
12            fCurrentBarLineRepeatDirectionKind,
13            fCurrentBarLineEndingTypeKind,
14            fCurrentBarLineEndingNumber,
15            fCurrentBarLineTimes,
16            msrBarLineCategoryKind::kBarLineCategory_UNKNOWN, // will be set afterwards
17            fCurrentBarLineHasSegnoKind,
18            fCurrentBarLineHasCodaKind,
19            fCurrentBarLineRepeatWingedKind);
20
21     // ... ..
22
23     // wait until its category is defined
24     // to append the barLine to the current segment
25
26     // handle the barLine according to: JMI
27     // http://www.musicxml.com/tutorial/the-midi-compatible-part/repeats/
28
29     Bool barLineHasBeenHandled = false;
30
31     switch (fCurrentBarLineLocationKind) {
32     case msrBarLineLocationKind::kBarLineLocationNone:
33         // should not occur
34         break;
35
36     case msrBarLineLocationKind::kBarLineLocationLeft:
37         if (
38             fCurrentBarLineEndingTypeKind
39             ==
40             msrBarLineEndingTypeKind::kBarLineEndingTypeStart
41         ) {
42             // ending start, don't know yet whether it's hooked or hookless
43             // -----
44             if (! fCurrentBarLineEndingNumber.size ()) {

```



```

45     musicxmlWarning (
46         gGlobalServiceRunData->getInputSourceName (),
47         inputLineNumber,
48         "mandatory ending number is missing, assuming \"1\\\"");
49
50     fCurrentBarLineEndingNumber = "1";
51 }
52
53 // don't know yet whether repeat ending start barLine is hooked or hookless
54 // remember it in fCurrentRepeatEndingStartBarLine,
55 fCurrentRepeatEndingStartBarLine = barLine;
56
57 // handle the repeat ending start
58 handleRepeatEndingStart (barLine);
59
60 barLineHasBeenHandled = true;
61 }
62
63 else if (
64     fCurrentBarLineRepeatDirectionKind
65     ==
66     msrBarLineRepeatDirectionKind::kBarLineRepeatDirectionForward
67 ) {
68     // repeat start
69     // -----
70     // set the barLine category
71     barLine->
72         setBarLineCategory (
73             msrBarLineCategoryKind::kBarLineCategoryRepeatStart);
74
75     // handle the repeat start
76     handleRepeatStart (barLine);
77
78     barLineHasBeenHandled = true;
79 }
80 break;
81
82 case msrBarLineLocationKind::kBarLineLocationMiddle:
83     // JMI ???
84     break;
85
86 case msrBarLineLocationKind::kBarLineLocationRight:
87     {
88         if (
89             fCurrentBarLineEndingTypeKind == msrBarLineEndingTypeKind::
kBarLineEndingTypeStop
90             &&
91             fCurrentBarLineEndingNumber.size () != 0
92         ) {
93             // hooked ending end
94             // -----
95             // set current barLine ending start category
96             fCurrentRepeatEndingStartBarLine->
97                 setBarLineCategory (
98                     msrBarLineCategoryKind::kBarLineCategoryHookedEndingStart);
99
100             // set this barLine's category
101             barLine->
102                 setBarLineCategory (
103                     msrBarLineCategoryKind::kBarLineCategoryHookedEndingEnd);
104
105             // handle the repeat hooked ending end
106             handleRepeatHookedEndingEnd (barLine);
107
108             barLineHasBeenHandled = true;
109         }
110

```

```

111     else if (
112         fCurrentBarLineRepeatDirectionKind
113         ==
114         msrBarLineRepeatDirectionKind::kBarLineRepeatDirectionBackward
115     ) {
116         // repeat end
117         // -----
118
119         // set this barLine's category
120         barLine->
121             setBarLineCategory (
122                 msrBarLineCategoryKind::kBarLineCategoryRepeatEnd);
123
124         // handle the repeat end
125         handleRepeatEnd (barLine);
126
127         barLineHasBeenHandled = true;
128     }
129
130     else if (
131         fCurrentBarLineEndingTypeKind == msrBarLineEndingTypeKind::
132         kBarLineEndingTypeDiscontinue
133         &&
134         fCurrentBarLineEndingNumber.size () != 0
135     ) {
136         // hookless ending end
137         // -----
138         // set current barLine ending start category
139         fCurrentRepeatEndingStartBarLine->
140             setBarLineCategory (
141                 msrBarLineCategoryKind::kBarLineCategoryHooklessEndingStart);
142
143         // set this barLine's category
144         barLine->
145             setBarLineCategory (
146                 msrBarLineCategoryKind::kBarLineCategoryHooklessEndingEnd);
147
148         // handle the repeat hookless ending end
149         handleRepeatHooklessEndingEnd (barLine);
150
151         barLineHasBeenHandled = true;
152     }
153
154     // forget about current repeat ending start barLine
155     fCurrentRepeatEndingStartBarLine = nullptr;
156 }
157 // switch
158
159 // set the barLine category to stand alone if not yet handled
160 if (! barLineHasBeenHandled) {
161     switch (fCurrentBarLineStyleKind) {
162     case msrBarLineStyleKind::kBarLineStyleRegular:
163     case msrBarLineStyleKind::kBarLineStyleDotted:
164     case msrBarLineStyleKind::kBarLineStyleDashed:
165     case msrBarLineStyleKind::kBarLineStyleHeavy:
166     case msrBarLineStyleKind::kBarLineStyleLightLight:
167     case msrBarLineStyleKind::kBarLineStyleLightHeavy:
168     case msrBarLineStyleKind::kBarLineStyleHeavyLight:
169     case msrBarLineStyleKind::kBarLineStyleHeavyHeavy:
170     case msrBarLineStyleKind::kBarLineStyleTick:
171     case msrBarLineStyleKind::kBarLineStyleShort:
172         barLine->
173             setBarLineCategory (
174                 msrBarLineCategoryKind::kBarLineCategoryStandalone);
175
176         // append the bar line to the current part

```

```

177      // ... ..
178
179      fCurrentPart->
180          appendBarLineToPart (barLine);
181
182      barLineHasBeenHandled = true;
183      break;
184
185      case msrBarLineStyleKind::kBarLineStyleNone:
186          std::stringstream s;
187
188          s <<
189              "barLine " <<
190              barLine->asString () <<
191              " has no barLine style";
192
193          musicxmlWarning (
194              gGlobalServiceRunData->getInputSourceName (),
195              inputLineNumber,
196              //      __FILE__, __LINE__,
197              s.str ());
198          break;
199      } // switch
200  }
201
202  // has this barLine been handled?
203  if (! barLineHasBeenHandled) {
204      std::stringstream s;
205
206      s << std::left <<
207          "cannot handle a barLine containing: " <<
208          barLine->asString ();
209
210      msrInternalWarning (
211          gGlobalServiceRunData->getInputSourceName (),
212          inputLineNumber,
213          s.str ());
214  }
215
216  fOnGoingBarLine = false;
217  }

```

## 48.2 Translating repeats from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

## 48.3 Translating repeats from MSR to MSR

This is done in `src/passes/msr2msr/`.

## 48.4 Translating repeats from MSR to LPSR

This is done in `src/passes/msr2lpsr/`.

## 48.5 Translating repeats from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

## Chapter 49

# Voices handling

Voices are presented at section [19.29](#) [Voices], page [186](#).

**49.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**49.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**49.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**49.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**49.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 50

# Staves handling

Staves are presented at section [19.27](#) [Staves], page [185](#).

**50.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**50.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**50.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**50.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**50.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

# Chapter 51

## Parts handling

Parts are presented at section [19.26](#) [Parts], page 185.

### 51.1 Parts browsing

Method `msrPart::browseData ()` defined in `src/formats/msr/msrParts.h/.cpp` is peculiar in that it imposes a *partial order* on the part staves browsing:

```
1 void msrPart::browseData (basevisitor* v)
2 {
3     if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
4         gLogStream <<
5             "% ==> msrPart::browseData ()" <<
6             std::endl;
7     }
8
9     #ifdef TRACING_IS_ENABLED // JMI
10    if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) { // JMI TEMP
11        gLogStream <<
12            "+++++++ fPartAllStavesList.size(): " <<
13            fPartAllStavesList.size () <<
14            std::endl;
15
16        if (fPartAllStavesList.size ()) {
17            for (S_msrStaff staff : fPartAllStavesList) {
18                gLogStream <<
19                    std::endl <<
20                    "+++++++ staff: +++++++" <<
21                    " \"\" << staff->getStaffName () << "\"\" <<
22                    std::endl;
23            } // for
24        }
25
26        gLogStream <<
27            "+++++++ fPartNonHarmoniesNorFiguredBassStavesList.size(): " <<
28            fPartNonHarmoniesNorFiguredBassStavesList.size () <<
29            std::endl;
30
31        if (fPartNonHarmoniesNorFiguredBassStavesList.size ()) {
32            for (S_msrStaff staff : fPartNonHarmoniesNorFiguredBassStavesList) {
33                gLogStream <<
34                    std::endl <<
35                    "+++++++ staff: +++++++" <<
36                    " \"\" << staff->getStaffName () << "\"\" <<
37                    std::endl;
38            } // for
39        }
```

```

40 }
41 #endif
42
43 /* don't enforce any order here, leave it to the client thru sorting JMI */
44
45 // browse the part harmonies staff if any right now, JMI
46 // to place it before the corresponding part
47 if (fPartHarmoniesStaff) {
48     msrBrowser<msrStaff> browser (v);
49     browser.browse (*fPartHarmoniesStaff);
50 }
51
52 // browse all non harmonies and non figured bass staves
53 for (S_msrStaff staff : fPartNonHarmoniesNorFiguredBassStavesList) {
54     // browse the staff
55     msrBrowser<msrStaff> browser (v);
56     browser.browse (*staff);
57 } // for
58
59 // browse the part figured bass staff if any only now, JMI
60 // to place it after the corresponding part
61 if (fPartFiguredBassStaff) {
62     msrBrowser<msrStaff> browser (v);
63     browser.browse (*fPartFiguredBassStaff);
64 }
65
66 // // browse all the part staves JMI
67 // for (S_msrStaff staff : fPartAllStavesList) {
68 //     if (staff != fPartHarmoniesStaff && staff != fPartFiguredBassStaff) {
69 //         // browse the staff
70 //         msrBrowser<msrStaff> browser (v);
71 //         browser.browse (*staff);
72 //     }
73 // } // for
74 }

```

## 51.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

## 51.3 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

## 51.4 Translating from MSR to MSR ([src/passes/msr2msr/](#))

## 51.5 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

## 51.6 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))



## Chapter 52

# Part groups handling

MusicFormats part groups are presented at chapter [19.25](#) [Part groups], page [184](#).

In MusicXML, part groups can overlap, even though no one seems ever to have needed that. That seems to be more a feature in the Finale handling of MusicXMLexport that a true musical need.

MSR does not support overlapping part group. Handling part groups is done in `src/passes/mxsr2msr/mxsr2msrTran` where overlapping groups are identified and rejected:

```
1 jacquesmenu@macmini > xml2ly partgroups/OverlappingPartGroups.xml
2 ### MusicXML ERROR ### partgroups/OverlappingPartGroups.xml:169:
3 There are overlapping part groups, namely:
4   '2' ==> PartGroup_6 ('2', partGroupName "1
5 2"), lines 164..169
6 and
7   '1' ==> PartGroup_2 ('1', partGroupName ""), lines 76..170
8
9 Please contact the maintainers of MusicFormats (see option '-c, -contact'):
10 either you found a bug in the xml2ly converter,
11 or this MusicXML data is the first-ever real-world case
12 of a score exhibiting overlapping part groups.
13 std::exception caught: mfException:
14 There are overlapping part groups, namely:
15   '2' ==> PartGroup_6 ('2', partGroupName "1
16 2"), lines 164..169
17 and
18   '1' ==> PartGroup_2 ('1', partGroupName ""), lines 76..170
19
20 Please contact the maintainers of MusicFormats (see option '-c, -contact'):
21 either you found a bug in the xml2ly converter,
22 or this MusicXML data is the first-ever real-world case
23 of a score exhibiting overlapping part groups.
24
25 Error message(s) were issued for input line 169
26 ### xml2ly gIndenter final value: 1 ###
27 ### Conversion from MusicXML to LilyPond failed ###
```

class `mxmlPartGroupDescr` contains:

```
1 struct mxmlPartGroupDescr : public smartable
2 {
3     /*
4     positions represent the order in which the parts appear in <part-list />
5     */
6
7     // ... ..
8
9     private:
10
```

```

11 // private fields
12 // -----
13
14 int                fStartInputLineNumber;
15 int                fStopInputLineNumber;
16
17 int                fPartGroupName; // may be reused later
18
19 S_msrPartGroup     fPartGroup;
20
21 int                fStartPosition;
22 int                fStopPosition;
23 };

```

Part groups numbers number re-used and they can be nested, so there is an implicit part group at the top of their hierarchy, attached to the class `msrScore`:

```

1 class EXP mxsr2msrSkeletonBuilder :
2 // ... ..
3
4 // an implicit part group has to be created to contain everything,
5 // since there can be parts out of any explicit part group
6 S_mxmlPartGroupDescr    fImplicitPartGroupDescr;
7 S_msrPartGroup          fImplicitPartGroup;
8
9 void                    createImplicitPartGroup ();
10
11 // part groups numbers can be re-used, they're no identifier
12 // we use a map to access them by part group number
13 int                    fPartGroupsCounter;
14 std::vector<S_mxmlPartGroupDescr>
15                        fPartGroupDescrsVector;
16 std::map<int, S_mxmlPartGroupDescr>
17                        fAllPartGroupDescrsMap;
18 std::map<int, S_mxmlPartGroupDescr>
19                        fStartedPartGroupDescrsMap;
20
21 // ... ..

```

## Chapter 53

# Scores handling

Scores are presented at section [19.24](#) [Scores], page [184](#).

**53.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**53.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**53.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**53.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**53.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 54

# Books handling

Books are presented at section [19.23](#) [Books], page [184](#).

**54.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**54.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**54.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**54.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**54.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 55

# Ornaments handling

Ornaments are presented at section ?? [Ornaments], page ??.

**55.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**55.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**55.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**55.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**55.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 56

# Ties handling

Ties are presented at section ?? [Ties], page ??.

**56.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**56.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**56.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**56.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**56.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 57

# Dynamics handling

Dynamics are presented at section ?? [Dynamics], page ??.

**57.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**57.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**57.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**57.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**57.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 58

# Beams handling

Beams are presented at section ?? [Beams], page ??.

**58.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**58.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**58.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**58.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**58.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))



## Chapter 59

# Slurs handling

Slurs are presented at section ?? [Slurs], page ??.

**59.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**59.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**59.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**59.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**59.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 60

# Grace notes groups handling

Grace notes groups are presented at section [19.39](#) [Grace notes groups], page [195](#).

**60.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**60.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**60.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**60.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**60.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 61

# Chords handling

Chords are presented at section [19.40](#) [Chords], page [195](#).

**61.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**61.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**61.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**61.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**61.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 62

# Tuples handling

Tuples are presented at section [19.41](#) [Tuples], page [196](#).

**62.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**62.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**62.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**62.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**62.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 63

# Harmonies handling

Harmonies are presented at section [19.43](#) [Harmonies], page [198](#).

The useful options here are:

- option `-trace-harmonies`, `-tharms`
- option `-display-msr-skeleton`, `-dmsrskel`
- option `-display-msr-1`, `-dmsr1`
- option `-display-msr-1-short`, `-dmsr1s` and option `-display-msr-1-details`, `-dmsr1d`
- option `-display-msr-2msr2`, `-dmsr2`
- option `-display-msr-2-short`, `-msr2s` and option `-display-msr-2-details`, `-dmsr2d`

Harmonies need special treatment since we need to determine their position in a harmony voice. This is different than MusicXML, where they are simply *drawn* at the current music position, so to say.

They are handled this way:

- harmonies are stored in class `msrNote`:
- they are also stored in class `msrPart`, class `msrChord` and class `msrTuplet` (denormalization);

In class `msrNote`, there is:

```
1 // harmonies
2 void                appendHarmonyToNote (
3                     const S_msrHarmony& harmony);
4
5 const std::list<S_msrHarmony>&
6     getNoteHarmoniesList () const
7     { return fNoteHarmoniesList; }
8
9 // ... ..
10
11 // harmonies
12 // -----
13
14 std::list<S_msrHarmony>
15     fNoteHarmoniesList;
16
17 // ... ..
```

## 63.1 Harmonies staves and voices

Every class `msrVoice` instance in `MusicFormats` belongs to an class `msrStaff` instance. Staves are created specifically to hold harmonies voices, using specific numbers defined in `src/formats/msr/msrParts.h`:

```

1 public:
2
3     // constants
4     // -----
5
6     #define msrPart::K_PART_HARMONIES_STAFF_NUMBER      10
7     #define msrPart::K_PART_HARMONIES_VOICE_NUMBER     11

```

In class `msrStaff`, there is:

```

1 void registerHarmoniesVoiceByItsNumber (
2     int inputLineNumber,
3     const S_msrVoice& voice);

```

Class `msrPart` also contains:

```

1 // harmonies
2
3 S_msrVoice createPartHarmoniesVoice (
4     int inputLineNumber,
5     std::string currentMeasureNumber);
6
7 void appendHarmonyToPart (
8     const S_msrVoice& harmonySupplierVoice,
9     const S_msrHarmony& harmony);
10
11 void appendHarmonyToPartClone (
12     const S_msrVoice& harmonySupplierVoice,
13     const S_msrHarmony& harmony);

```

```

1 // harmonies
2
3 S_msrStaff fPartHarmoniesStaff;
4 S_msrVoice fPartHarmoniesVoice;

```

## 63.2 Harmonies staves creation

This is done in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.cpp.h/.cpp`:

```

1 S_msrVoice mxsr2msrSkeletonBuilder::createPartHarmoniesVoiceIfNotYetDone (
2     int inputLineNumber,
3     const S_msrPart& part)
4 {
5     // is the harmonies voice already present in part?
6     S_msrVoice
7     partHarmoniesVoice =
8     part->
9     getPartHarmoniesVoice ();
10
11     if (! partHarmoniesVoice) {
12         // create the harmonies voice and append it to the part
13         partHarmoniesVoice =
14         part->
15         createPartHarmoniesVoice (
16             inputLineNumber,
17             fCurrentMeasureNumber);

```

```

18 }
19
20 return partHarmoniesVoice;
21 }

```

Method `msrPartcreatePartHarmoniesVoice` creates the part harmonies staff and the part harmonies voice, and then registers the latter in the former:

```

1 S_msrVoice msrPart::createPartHarmoniesVoice (
2     int     inputLineNumber,
3     std::string currentMeasureNumber)
4 {
5     // ... ..
6
7     // create the part harmonies staff
8     int partHarmoniesStaffNumber =
9         msrPart::K_PART_HARMONIES_STAFF_NUMBER;
10
11    // ... ..
12
13    fPartHarmoniesStaff =
14        addHarmoniesStaffToPart (
15            inputLineNumber);
16
17    // ... ..
18
19    // create the part harmonies voice
20    int partHarmoniesVoiceNumber =
21        msrPart::K_PART_HARMONIES_VOICE_NUMBER;
22
23    // ... ..
24
25    fPartHarmoniesVoice =
26        msrVoice::create (
27            inputLineNumber,
28            msrVoiceKind::kVoiceKindHarmonies,
29            partHarmoniesVoiceNumber,
30            msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes,
31            fPartHarmoniesStaff);
32
33    // register the part harmonies voice in part harmonies staff
34    fPartHarmoniesStaff->
35        registerVoiceInStaff (
36            inputLineNumber,
37            fPartHarmoniesVoice);
38
39    // ... ..
40
41    return fPartHarmoniesVoice;
42 }

```

## 63.3 Translating harmonies from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

The MSR score skeleton created in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.h/.cpp` contains the part groups, parts, staves and voices, as well as the number of measures. The voices do not contain any music elements yet.

A harmony belongs to a `<part/>` in MusicXML, but we sometimes need to have it attached to a note. When visiting an `S_harmony` element, field `mxsr2msrSkeletonBuilder::fThereAreHarmoniesToBeAttachedToCurrentNote` is used to account for that:

```

1 void mxsr2msrSkeletonBuilder::visitStart ( S_harmony& elt )
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
5             gLogStream <<
6                 "--> Start visiting S_harmony" <<
7                 ", harmoniesVoicesCounter = " << fHarmoniesVoicesCounter <<
8                 ", line " << elt->getInputLineNumber () <<
9                 std::endl;
10        }
11    #endif
12
13    /* JMI ???
14       several harmonies can be attached to a given note,
15       leading to as many harmonies voices in the current part
16    */
17
18    // take harmonies voice into account
19    ++fHarmoniesVoicesCounter; // UNUSED JMI
20
21    fThereAreHarmoniesToBeAttachedToCurrentNote = true;
22 }

```

Upon the second visit of class `msrNote`, the part harmonies voice is created if harmonies are not to be ignored due to option `-ignore-musicxml-harmonies`, `-oharms` and it has not been created yet:

```

1 void mxsr2msrSkeletonBuilder::visitEnd ( S_note& elt )
2 {
3     // ... ..
4
5     // are there harmonies attached to the current note?
6     if (fThereAreHarmoniesToBeAttachedToCurrentNote) {
7         if (gGlobalMxsr2msrOahGroup->getIgnoreHarmonies ()) {
8             #ifdef TRACING_IS_ENABLED
9                 if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
10                    gLogStream <<
11                        "Ignoring the harmonies" <<
12                        ", line " <<
13                        inputLineNumber <<
14                        std::endl;
15                }
16            #endif
17        }
18        else {
19            // create the part harmonies voice if not yet done
20            S_msrVoice
21                partHarmoniesVoice =
22                createPartHarmoniesVoiceIfNotYetDone (
23                    inputLineNumber,
24                    fCurrentPart);
25        }
26
27        fThereAreHarmoniesToBeAttachedToCurrentNote = false;
28    }
29
30    // ... ..
31 }

```

Creating the part harmonies voice is delegated to the part:

```

1 S_msrVoice mxsr2msrSkeletonBuilder::createPartHarmoniesVoiceIfNotYetDone (
2     int inputLineNumber,
3     const S_msrPart& part)
4 {
5     // is the harmonies voice already present in part?

```



```

6   S_msrVoice
7       partHarmoniesVoice =
8           part->
9               getPartHarmoniesVoice ();
10
11   if (! partHarmoniesVoice) {
12       // create the harmonies voice and append it to the part
13       partHarmoniesVoice =
14           part->
15               createPartHarmoniesVoice (
16                   inputLineNumber,
17                   fCurrentMeasureNumber);
18   }
19
20   return partHarmoniesVoice;
21 }

```

## 63.4 Translating harmonies from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

This is where the class `msrHarmony` instances are created.

### 63.4.1 First S\_harmony visit

The first visit of `S_harmony` initializes the fields storing values to be gathered visiting subelements:

```

1   void mxsr2msrTranslator::visitStart ( S_harmony& elt )
2   {
3       int inputLineNumber =
4           elt->getInputLineNumber ();
5
6   #ifdef TRACING_IS_ENABLED
7       if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
8           gLogStream <<
9               "--> Start visiting S_harmony" <<
10              ", line " << inputLineNumber <<
11              std::endl;
12      }
13  #endif
14
15      ++fHarmoniesVoicesCounter;
16
17      fCurrentHarmonyInputLineNumber      = inputLineNumber;
18
19      fCurrentHarmonyRootDiatonicPitchKind = msrDiatonicPitchKind::kDiatonicPitch_UNKNOWN;
20      fCurrentHarmonyRootAlterationKind    = msrAlterationKind::kAlterationNatural;
21      fCurrentHarmonyKind                  = msrHarmonyKind::kHarmony_UNKNOWN;
22      fCurrentHarmonyKindText              = "";
23      fCurrentHarmonyInversion             = K_HARMONY_INVERSION_NONE;
24      fCurrentHarmonyBassDiatonicPitchKind = msrDiatonicPitchKind::kDiatonicPitch_UNKNOWN;
25      fCurrentHarmonyBassAlterationKind    = msrAlterationKind::kAlterationNatural;
26      fCurrentHarmonyDegreeValue           = -1;
27      fCurrentHarmonyDegreeAlterationKind = msrAlterationKind::kAlterationNatural;
28
29      fCurrentHarmonyWholeNotesOffset = Rational (0, 1);
30
31      fOnGoingHarmony = true;
32  }

```

### 63.4.2 Second S\_harmony visit

Upon the second visit of `S_harmony`, a class `msrHarmony` instance is created, populated and appended to `mxsr2msrTranslatorfPendingHarmoniesList`.

The voice uplink will be set later, hence the use of method `msrHarmony::create ()`:

```

1 void mxsr2msrTranslator::visitEnd ( S_harmony& elt )
2 {
3     // ... ..
4
5     if (gGlobalMxsr2msrOahGroup->getIgnoreHarmonies ()) {
6         #ifdef TRACING_IS_ENABLED
7             if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
8                 gLogStream <<
9                     "Ignoring harmony" <<
10                     ", line " <<
11                     inputLineNumber <<
12                     std::endl;
13             }
14         #endif
15     }
16     else {
17         // create the harmony
18         #ifdef TRACING_IS_ENABLED
19             if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
20                 gLogStream <<
21                     "Creating a harmony" <<
22                     ", line " << inputLineNumber << ":" <<
23                     std::endl;
24             }
25             // ... ..
26         }
27         #endif
28
29         S_msrHarmony
30         harmony =
31             msrHarmony::create (
32                 fCurrentHarmonyInputLineNumber,
33                 // no harmoniesUpLinkToVoice yet
34
35                 fCurrentHarmonyRootQuarterTonesPitchKind,
36
37                 fCurrentHarmonyKind,
38                 fCurrentHarmonyKindText,
39
40                 fCurrentHarmonyInversion,
41
42                 fCurrentHarmonyBassQuarterTonesPitchKind,
43
44                 Rational (1, 1),           // harmonySoundingWholeNotes,
45                                           // will be set upon next note handling
46                 Rational (1, 1),           // harmonyDisplayWholeNotes,
47                                           // will be set upon next note handling
48                 fCurrentHarmoniesStaffNumber,
49                 msrTupletFactor (1, 1),    // will be set upon next note handling
50                 fCurrentHarmonyWholeNotesOffset);
51
52         // append pending harmony degrees if any to the harmony
53         if (! fCurrentHarmonyDegreesList.size ()) {
54             #ifdef TRACING_IS_ENABLED
55                 if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
56                     musicxmlWarning (
57                         gGlobalServiceRunData->getInputSourceName (),
58                         inputLineNumber,
59                         "harmony has no degrees contents");
60                 }

```

```

61 #endif
62 }
63
64 else {
65     // handle harmony degrees if any
66     while (fCurrentHarmonyDegreesList.size ()) {
67         S_msrHarmonyDegree
68         harmonyDegree =
69             fCurrentHarmonyDegreesList.front ();
70
71         // ... ..
72
73         // append it to harmony's degrees list
74         harmony->
75             appendHarmonyDegreeToHarmony (
76                 harmonyDegree);
77
78         // remove it from the list
79         fCurrentHarmonyDegreesList.pop_front ();
80     } // while
81 }
82
83 // attach the current frame if any to the harmony
84 if (fCurrentFrame) {
85     harmony->setHarmonyFrame (fCurrentFrame);
86 }
87
88 // append the harmony to the pending harmonies list
89 fPendingHarmoniesList.push_back (harmony);
90 }
91
92 fOnGoingHarmony = false;
93 }

```

### 63.4.3 Attaching msrHarmony instances to notes

#### msrHarmony

The contents of `mxsr2msrTranslatorfPendingHarmoniesList` is attached to the class `msrNote` instance in method `mxsr2msrTranslator::populateNote ()`:

```

1 void mxsr2msrTranslator::populateNote (
2     int         inputLineNumber,
3     const S_msrNote& newNote)
4 {
5     // ... ..
6
7     // handle the pending harmonies if any
8     if (fPendingHarmoniesList.size ()) {
9         // get voice to insert harmonies into
10        S_msrVoice
11        voiceToInsertHarmoniesInto =
12            fCurrentPart->
13                getPartHarmoniesVoice ();
14
15        // ... ..
16
17        handlePendingHarmonies (
18            newNote,
19            voiceToInsertHarmoniesInto);
20
21        // reset harmony counter
22        fHarmoniesVoicesCounter = 0;
23    }

```

24 }

### 63.4.4 Populating msrHarmony instances

msrHarmony

The class `msrHarmony` instances are populated further in `src/formats/msr/mxsr2msrTranslator.cpp` and attached to the note by method `msrNote::appendHarmonyToNote ()`:

```

1 void mxsr2msrTranslator::handlePendingHarmonies (
2     const S_msrNote& newNote,
3     const S_msrVoice& voiceToInsertInto)
4 {
5     // ... ..
6
7     Rational
8         newNoteSoundingWholeNotes =
9         newNote->
10             getMeasureElementSoundingWholeNotes (),
11         newNoteDisplayWholeNotes =
12         newNote->
13             getNoteDisplayWholeNotes ();
14
15     while (fPendingHarmoniesList.size ()) { // recompute at each iteration
16         S_msrHarmony
17             harmony =
18             fPendingHarmoniesList.front ();
19
20         /*
21          MusicXML harmonies don't have a duration,
22          and MSR could follow this line, but LilyPond needs one...
23          So:
24          - we register all harmonies with the duration of the next note
25          - they will be sorted by position in the measure in finalizeMeasure(),
26            at which time their duration may be shortened
27            so that the offsets values are enforced
28            and they don't overflow the measure
29          It is VITAL that harmonies measures be finalized
30          AFTER the corresponding measure in the regular voice,
31          since the current sounding whole notes of the latter is needed for that
32          */
33
34         // set the harmony's sounding whole notes
35         harmony->
36             setMeasureElementSoundingWholeNotes (
37                 newNoteSoundingWholeNotes,
38                 "mxsr2msrTranslator::handlePendingHarmonies()");
39
40         // set the harmony's display whole notes JMI useless???
41         harmony->
42             setHarmonyDisplayWholeNotes (
43                 newNoteDisplayWholeNotes);
44
45         // set the harmony's tuplet factor
46         harmony->
47             setHarmonyTupletFactor (
48                 msrTupletFactor (
49                     fCurrentNoteActualNotes,
50                     fCurrentNoteNormalNotes));
51
52         // attach the harmony to newNote
53         newNote->
54             appendHarmonyToNote (
55                 harmony);

```

```

56
57 // get the part harmonies voice
58 S_msrVoice
59     partHarmoniesVoice =
60         fCurrentPart->
61             getPartHarmoniesVoice ();
62
63 // sanity check
64 mfAssert (
65     __FILE__, __LINE__,
66     partHarmoniesVoice != nullptr,
67     "partHarmoniesVoice is null");
68
69 // set the harmony's voice upLink
70 // only now that we know which harmonies voice will contain it
71 harmony->
72     setHarmoniesUpLinkToVoice (
73         partHarmoniesVoice);
74
75 /* JMI CAFE
76 // append the harmony to the part harmonies voice
77 partHarmoniesVoice->
78     appendHarmonyToVoice (
79         harmony);
80 */
81 // don't append the harmony to the part harmonies voice // BLARK
82 // before the note itself has been appended to the voice
83
84 // remove the harmony from the list
85 fPendingHarmoniesList.pop_front ();
86 } // while
87 }

```

### 63.4.5 First S\_harmony visit

msrHarmony

Method `msrNote::appendHarmonyToNote ()` is where the harmony's note uplink is set:

```

1 void msrNote::appendHarmonyToNote (const S_msrHarmony& harmony)
2 {
3 #ifdef TRACING_IS_ENABLED
4     if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
5         gLogStream <<
6             "Appending harmony " <<
7             harmony->asString () <<
8             " to the harmonies list of " <<
9             asString () <<
10            ", line " << fInputLineNumber <<
11            std::endl;
12     }
13 #endif
14
15 // update the harmony whole notes if it belongs to a triplet ??? utf8.xml JMI
16
17 fNoteHarmoniesList.push_back (harmony);
18
19 // register this note as the harmony note upLink
20 harmony->
21     setHarmonyUpLinkToNote (this);
22 }

```

When a harmony is attached to a note that is a chord member, we have to attach it to the chord too, to facilitate setting its measure position when setting the chord's one.

```

1 void mxsr2msrTranslator::copyNoteHarmoniesToChord (
2     const S_msrNote& note,
3     const S_msrChord& chord)
4 {
5     // copy note's harmony if any from the first note to chord
6
7     const std::list<S_msrHarmony>&
8         noteHarmoniesList =
9         note->getNoteHarmoniesList ();
10
11     if (noteHarmoniesList.size ()) {
12         std::list<S_msrHarmony>::const_iterator i;
13         for (i=noteHarmoniesList.begin (); i!=noteHarmoniesList.end (); ++i) {
14             S_msrHarmony harmony = (*i);
15
16 #ifdef TRACING_IS_ENABLED
17             if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
18                 gLogStream <<
19                     "Copying harmony '" <<
20                     harmony->asString () <<
21                     "' from note " << note->asString () <<
22                     " to chord '" << chord->asString () <<
23                     "'" <<
24                     std::endl;
25             }
26 #endif
27
28             chord->
29                 appendHarmonyToChord (harmony);
30
31         } // for
32     }
33 }

```

### 63.4.6 Inserting msrHarmony instances in the part harmonies voice

msrHarmony

Inserting the harmonies in the part harmonies voice is done in method `msrVoice::appendNoteToVoice ()` in `src/formats/msr/msrNotes.cpp`:

```

1 void msrVoice::appendNoteToVoice (const S_msrNote& note)
2 {
3     // ... ..
4
5     // are there harmonies attached to this note? // BLARK
6     const std::list<S_msrHarmony>&
7         noteHarmoniesList =
8         note->
9         getNoteHarmoniesList ();
10
11     if (noteHarmoniesList.size ()) {
12         // get the current part's harmonies voice
13         S_msrVoice
14             partHarmoniesVoice =
15             part->
16             getPartHarmoniesVoice ();
17
18         for (S_msrHarmony harmony : noteHarmoniesList) {
19             // append the harmony to the part harmonies voice
20             partHarmoniesVoice->
21                 appendHarmonyToVoice (
22                     harmony);
23         }
24     }
25 }

```

```

23     } // for
24 }
25
26 // ... ..
27 };

```

## 63.5 Translating harmonies from MSR to MSR

This is done in `src/passes/msr2msr/`.

In `src/passes/msr2msr/msr2msrTranslator.cpp`, a newborn clone of the harmony is created upon the first visit, stored in `msr2msrTranslatorfCurrentHarmonyClone`, and appended to the current non grace note clone, the current chord clone or to the current voice clone, if the latter is a harmonies voice:

```

1 void msr2msrTranslator::visitStart (S_msrHarmony& elt)
2 {
3     #ifndef TRACING_IS_ENABLED
4         if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5             gLogStream <<
6                 "--> Start visiting msrHarmony '" <<
7                 elt->asString () <<
8                 ", fOnGoingNonGraceNote: " << fOnGoingNonGraceNote <<
9                 ", fOnGoingChord: " << fOnGoingChord <<
10                ", fOnGoingHarmoniesVoice: " << fOnGoingHarmoniesVoice <<
11                ", fOnGoingHarmony: " << fOnGoingHarmony <<
12                "', line " << elt->getInputLineNumber () <<
13                std::endl;
14         }
15     #endif
16
17     // create a harmony newborn clone
18     fCurrentHarmonyClone =
19         elt->
20             createHarmonyNewbornClone (
21                 fCurrentVoiceClone);
22
23     if (fOnGoingNonGraceNote) {
24         // register the harmony in the current non-grace note clone
25         fCurrentNonGraceNoteClone->
26             appendHarmonyToNote (
27                 fCurrentHarmonyClone);
28
29         // don't append the harmony to the part harmony,
30         // this has been done in pass2b // JMI ???
31     }
32
33     else if (fOnGoingChord) {
34         // register the harmony in the current chord clone
35         fCurrentChordClone->
36             appendHarmonyToChord (fCurrentHarmonyClone); // JMI
37     }
38
39     else if (fOnGoingHarmoniesVoice) {
40         /* JMI
41         // get the harmony whole notes offset
42         Rational
43             harmonyWholeNotesOffset =
44                 elt->getHarmonyWholeNotesOffset ();
45
46         // is harmonyWholeNotesOffset not equal to 0?
47         if (harmonyWholeNotesOffset.getNumerator () != 0) {
48             // create skip with duration harmonyWholeNotesOffset
49             S_msrNote

```

```

50     skip =
51         msrNote::createSkipNote (
52             elt->                getInputLineNumber (),
53             "666", // JMI elt->                getHarmoniesMeasureNumber (),
54             elt->                getHarmonyDisplayWholeNotes (), // would be 0/1 otherwise
55         JMI
56             elt->                getHarmonyDisplayWholeNotes (),
57             0, // JMI elt->                getHarmonyDotsNumber (),
58             fCurrentVoiceClone-> getRegularVoiceStaffSequentialNumber (), // JMI
59             fCurrentVoiceClone-> getVoiceNumber ());
60
61     // append it to the current voice clone
62     // to 'push' the harmony aside
63     fCurrentVoiceClone->
64         appendNoteToVoice (skip);
65 }
66
67 // append the harmony to the current voice clone
68 fCurrentVoiceClone->
69     appendHarmonyToVoiceClone (
70         fCurrentHarmonyClone);
71 }
72
73 else {
74     std::stringstream s;
75
76     s <<
77         "harmony is out of context, cannot be handled: " <<
78         elt->asShortString ();
79
80     msrInternalError (
81         gGlobalServiceRunData->getInputSourceName (),
82         elt->getInputLineNumber (),
83         __FILE__, __LINE__,
84         s.str ());
85 }
86
87 fOnGoingHarmony = true;
88 }

```

There are only fields updates upon the second visit:

```

1 void msr2msrTranslator::visitEnd (S_msrHarmony& elt)
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5             gLogStream <<
6                 "--> End visiting msrHarmony '" <<
7                 elt->asString () <<
8                 "' " <<
9                 ", line " << elt->getInputLineNumber () <<
10                std::endl;
11         }
12     #endif
13
14     fCurrentHarmonyClone = nullptr;
15     fOnGoingHarmony = false;
16 }

```

## 63.6 Translating harmonies from MSR to LPSR

This is done in `src/passes/msr2lpsr/`.



The same occurs in `src/passes/msr2lpsr/msr2lpsrTranslator.cpp`: a newborn clone of the harmony is created and appended to the current non grace note clone, the current chord clone or to the current voice clone, if the latter is a harmonies voice: :

```

1 void msr2lpsrTranslator::visitStart (S_msrHarmony& elt)
2 {
3 #ifdef TRACING_IS_ENABLED
4     if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5         gLogStream <<
6             "--> Start visiting msrHarmony '" <<
7             elt->asString () <<
8             ", onGoingNonGraceNote: " << fOnGoingNonGraceNote <<
9             ", onGoingChord: " << fOnGoingChord <<
10            ", onGoingHarmoniesVoice: " << fOnGoingHarmoniesVoice <<
11            ", onGoingHarmony: " << fOnGoingHarmony <<
12            "', line " << elt->getInputLineNumber () <<
13            std::endl;
14     }
15 #endif
16
17     // create a harmony newborn clone
18     fCurrentHarmonyClone =
19         elt->
20             createHarmonyNewbornClone (
21                 fCurrentVoiceClone);
22
23     if (fOnGoingNonGraceNote) {
24         // register the harmony in the current non-grace note clone
25         fCurrentNonGraceNoteClone->
26             appendHarmonyToNote (
27                 fCurrentHarmonyClone);
28
29         // don't append the harmony to the part harmony,
30         // this has been done in pass2b // JMI ???
31     }
32
33     else if (fOnGoingChord) {
34         // register the harmony in the current chord clone
35         fCurrentChordClone->
36             appendHarmonyToChord (fCurrentHarmonyClone); // JMI
37     }
38
39     else if (fOnGoingHarmoniesVoice) {
40         /* JMI
41         // get the harmony whole notes offset
42         Rational
43             harmonyWholeNotesOffset =
44             elt->getHarmonyWholeNotesOffset ();
45
46         // is harmonyWholeNotesOffset not equal to 0?
47         if (harmonyWholeNotesOffset.getNumerator () != 0) {
48             // create skip with duration harmonyWholeNotesOffset
49             S_msrNote
50                 skip =
51                 msrNote::createSkipNote (
52                     elt->
53                         getInputLineNumber (),
54                         "666", // JMI elt->
55                             getHarmoniesMeasureNumber (),
56                             getHarmonyDisplayWholeNotes (), // would be 0/1 otherwise
57
58                     JMI
59                     elt->
60                         getHarmonyDisplayWholeNotes (),
61                         0, // JMI elt->
62                             getHarmonyDotsNumber (),
63                             fCurrentVoiceClone-> getRegularVoiceStaffSequentialNumber (), // JMI
64                             fCurrentVoiceClone-> getVoiceNumber ());
65
66         // append it to the current voice clone
67         // to 'push' the harmony aside
68         fCurrentVoiceClone->

```

```
63         appendNoteToVoice (skip);
64     }
65 */
66
67     // append the harmony to the current voice clone
68     fCurrentVoiceClone->
69         appendHarmonyToVoiceClone (
70             fCurrentHarmonyClone);
71 }
72
73 else {
74     std::stringstream s;
75
76     s <<
77         "harmony is out of context, cannot be handled: " <<
78         elt->asShortString ();
79
80     msrInternalError (
81         gGlobalServiceRunData->getInputSourceName (),
82         elt->getInputLineNumber (),
83         __FILE__, __LINE__,
84         s.str ());
85 }
86
87 fOnGoingHarmony = true;
88 }
```

Here too, there are only fields updates upon the second visit of `S_msrHarmony` instances:

```

1 void msr2lpsrTranslator::visitEnd (S_msrHarmony& elt)
2 {
3 #ifdef TRACING_IS_ENABLED
4     if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5         gLogStream <<
6             "--> End visiting msrHarmony '" <<
7             elt->asString () <<
8             "' " <<
9             ", line " << elt->getInputLineNumber () <<
10            std::endl;
11    }
12 #endif
13
14    fCurrentHarmonyClone = nullptr;
15    fOnGoingHarmony = false;
16 }

```

## 63.7 Translating harmonies from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

There is only one visit of class `msrHarmony` instances in `src/passes/lpsr2lilypond/lpsr2lilypondTranslator.cpp`

The LilyPond code is generated only if the harmony belongs to a voice: this is where denormalization ends in the workflow:

```

1 void lpsr2lilypondTranslator::visitStart (S_msrHarmony& elt)
2 {
3 #ifdef TRACING_IS_ENABLED
4     {
5         Bool
6         traceMsrVisitors =
7             gGlobalMsrOahGroup->
8             getTraceMsrVisitors (),
9         generateMsrVisitingInformation =
10            gGlobalLpsr2lilypondOahGroup->
11            getGenerateMsrVisitingInformation ();
12
13         if (traceMsrVisitors || generateMsrVisitingInformation) {
14             std::stringstream s;
15
16             s <<
17                 "% --> Start visiting msrHarmony '" <<
18                 elt->asString () <<
19                 "' " <<
20                 ", fOnGoingNotesStack.size () = " <<
21                 fOnGoingNotesStack.size () <<
22                 ", fOnGoingChord = " <<
23                 fOnGoingChord <<
24                 ", fOnGoingHarmoniesVoice = " <<
25                 fOnGoingHarmoniesVoice <<
26                 ", line " << elt->getInputLineNumber () <<
27                 std::endl;
28
29             if (traceMsrVisitors) {
30                 gLogStream << s.str ();
31             }
32
33             if (generateMsrVisitingInformation) {
34                 fLilypondCodeStream << s.str ();
35             }
36         }
37     }
38 }

```

```

37     }
38 #endif
39
40     if (fOnGoingNotesStack.size () > 0) {
41         /* JMI
42 #ifdef TRACING_IS_ENABLED
43         if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
44             fLilypondCodeStream <<
45                 "%{ fOnGoingNotesStack.size () S_msrHarmony JMI " <<
46                 elt->asString () <<
47                 " %}" <<
48                 std::endl;
49         }
50 #endif
51 */
52     }
53
54     else if (fOnGoingChord) { // JMI
55     }
56
57     else if (fOnGoingHarmoniesVoice) {
58         // actual LilyPond code generation
59         fLilypondCodeStream <<
60             harmonyAsLilypondString (elt) <<
61             ' ';
62
63         // generate the input line number as comment if relevant
64         if (
65             gGlobalLpsr2lilypondOahGroup->getInputLineNumbers ()
66             ||
67             gGlobalLpsr2lilypondOahGroup->getGenerateMeasurePositions ()
68         ) {
69             generateInputLineNumberAndOrMeasurePositionAsAComment (
70                 elt);
71         }
72     }
73 }

```

## Chapter 64

# Figured bass handling

Figured bass elements are presented at section ?? [Figured bass elements], page ??.

The useful options here are:

- option `-trace-figured-bass`, `-tfigbass`
- option `-display-msr-skeleton`, `-dmsrskel`
- option `-display-msr-1-short`, `-dmsr1short`
- option `-display-msr-1`, `-dmsr1`, an alias for the one above
- option `-display-msr-1-full`, `-dmsr1full`
- option `-display-msr-2-short`, `-dmsr2short`
- option `-display-msr-2`, `-dmsr2`, an alias for the one above
- option `-display-msr-2-full`, `-dmsr2full`
- option `-display-lpsr-short`, `-dlpsrshort`
- option `-display-lpsr`, `-dlpsr`, an alias for the one above
- option `-display-lpsr-full`, `-dlpsrfull`

### 64.1 Figured bass description

Figured bass is represented in MSR by classes defined in `src/formats/msr/msrFiguredBasses.h/.cpp`. There is class `msrFiguredBass`:

```
1 class EXP msrFiguredBass : public msrMeasureElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrNote          fFiguredBassUpLinkToNote;
12        S_msrVoice         fFiguredBassUpLinkToVoice; // for use in figured bass voices JMI
13
14        Rational           fFiguredBassDisplayWholeNotes;
```

```

15
16     msrFiguredBassParenthesesKind
17         fFiguredBassParenthesesKind;
18
19     std::list<S_msrBassFigure> fFiguredBassFiguresList;
20
21     msrTupletFactor           fFiguredBassTupletFactor;
22 };

```

The figured bass figures are defined in:

```

1 class EXP msrBassFigure : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrPart           fFigureUpLinkToPart;
12
13        msrBassFigurePrefixKind
14            fFigurePrefixKind;
15        int           fFigureNumber;
16        msrBassFigureSuffixKind
17            fFigureSuffixKind;
18 };

```

Figured bass elements need special treatment since we need to determine their position in a figured bass voice. This is different than MusicXML, where they are simply *drawn* at the current music position, so to say.

They are handled this way:

- figured bass elements are stored in class `msrNote`:
- they are also stored in class `msrPart` and class `msrChord` and class `msrTuplet` (denormalization);

In class `msrNote`, there is:

```

1     // figured bass
2     void           appendFiguredBassToNoteFiguredBassesList (
3         const S_msrFiguredBass& figuredBass);
4
5     const std::list<S_msrFiguredBass>&
6         getNoteFiguredBassesList () const
7         { return fNoteFiguredBassesList; }
8
9     // ... ..
10
11    // figured bass
12    // -----
13
14    std::list<S_msrFiguredBass>
15        fNoteFiguredBassesList;

```

## 64.2 Figured bass staves and voices

Every class `msrVoice` instance in `MusicFormats` belongs to an class `msrStaff` instance. Staves are created specifically to hold figured bass voices, using specific numbers defined in `src/formats/msr/msrParts.h`:

```

1 public:
2
3     // constants
4     // -----
5
6     // ... ..
7
8     #define msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER  20
9     #define msrPart::K_PART_FIGURED_BASS_VOICE_NUMBER  21

```

In class `msrStaff`, there is:

```

1     void                registerFiguredBassVoiceByItsNumber (
2                         int                inputLineNumber,
3                         const S_msrVoice& voice);

```

Class `msrPart` also contains:

```

1     // figured bass
2
3     S_msrVoice          createPartFiguredBassVoice (
4                         int                inputLineNumber,
5                         std::string currentMeasureNumber);
6
7     void                appendFiguredBassToPart (
8                         const S_msrVoice&          figuredBassSupplierVoice,
9                         S_msrFiguredBass figuredBass);
10
11    void                appendFiguredBassToPartClone (
12                        const S_msrVoice&          figuredBassSupplierVoice,
13                        const S_msrFiguredBass& figuredBass);

```

```

1     // figured bass
2
3     S_msrStaff          fPartFiguredBassStaff;
4     S_msrVoice          fPartFiguredBassVoice;

```

## 64.3 Figured bass staves creation

This is done in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.cpp.h/.cpp`:

```

1 S_msrVoice mxsr2msrSkeletonBuilder::createPartFiguredBassVoiceIfNotYetDone (
2     int                inputLineNumber,
3     const S_msrPart&  part)
4 {
5     // is the figured bass voice already present in part?
6     S_msrVoice
7     partFiguredBassVoice =
8     part->
9     getPartFiguredBassVoice ();
10
11    if (! partFiguredBassVoice) {
12        // create the figured bass voice and append it to the part
13        partFiguredBassVoice =
14        part->
15        createPartFiguredBassVoice (

```

```

16         inputLineNumber,
17         fCurrentMeasureNumber);
18     }
19
20     return partFiguredBassVoice;
21 }

```

Method `msrPart::createPartFiguredBassVoice ()` creates the part figured bass staff and the part figured bass voice, and then registers the latter in the former:

```

1 S_msrVoice msrPart::createPartFiguredBassVoice (
2     int     inputLineNumber,
3     std::string currentMeasureNumber)
4 {
5     // ... ..
6
7     // create the part figured bass staff
8     int partFiguredBassStaffNumber =
9         msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER;
10
11     // ... ..
12
13     fPartFiguredBassStaff =
14         addHFiguredBassStaffToPart (
15             inputLineNumber);
16
17     // ... ..
18
19     // create the figured bass voice
20     int partFiguredBassVoiceNumber =
21         msrPart::K_PART_FIGURED_BASS_VOICE_NUMBER;
22
23     // ... ..
24
25     fPartFiguredBassVoice =
26         msrVoice::create (
27             inputLineNumber,
28             msrVoiceKind::kVoiceKindFiguredBass,
29             partFiguredBassVoiceNumber,
30             msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes,
31             fPartFiguredBassStaff);
32
33     // register the figured bass voice in the part figured bass staff
34     fPartFiguredBassStaff->
35         registerVoiceInStaff (
36             inputLineNumber,
37             fPartFiguredBassVoice);
38
39     // ... ..
40
41     return fPartFiguredBassVoice;
42 }

```

## 64.4 Translating figured bass from MXSR to MSR

This is done in `src/passes/mxsr2msr/`, and this is where the class `msrFiguredBass` instances are created. There several methods for Figured bass elements creation:

```

1 jacquesmenu@macmini:~/musicformats-git-dev/src/representations/msr > grep create
   msrFiguredBasses.h
2     static SMARTP<msrBassFigure> create (
3     SMARTP<msrBassFigure> createFigureNewbornClone (
4     SMARTP<msrBassFigure> createFigureDeepClone ( // JMI ???

```



```

5 static SMARTP<msrFiguredBass> create (
6 static SMARTP<msrFiguredBass> create (
7 SMARTP<msrFiguredBass> createFiguredBassNewbornClone (
8 SMARTP<msrFiguredBass> createFiguredBassDeepClone ();

```

The MSR score skeleton created in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.h/.cpp` contains the part groups, parts, staves and voices, as well as the number of measures. The voices do not contain any music elements yet.

A figured bass element belongs to `<part/>` in MusicXML, but we sometimes need to have it attached to a note.

Field `mxsr2msrSkeletonBuilder::fThereAreFiguredBassToBeAttachedToCurrentNote` is used when visiting an `S_FiguredBass` element to account for that:

```

1 void mxsr2msrSkeletonBuilder::visitStart ( S_figured_bass& elt )
2 {
3 #ifdef TRACING_IS_ENABLED
4     if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
5         gLogStream <<
6             "--> Start visiting S_figured_bass" <<
7             ", figuredBassVoicesCounter = " << fFiguredBassVoicesCounter <<
8             ", line " << elt->getInputLineNumber () <<
9             std::endl;
10    }
11 #endif
12
13    /* JMI
14       several figured bass elements can be attached to a given note,
15       leading to as many figured bass voices in the current part JMI TRUE???
16    */
17
18    // take figured bass voice into account
19    ++fFiguredBassVoicesCounter;
20
21    fThereAreFiguredBassToBeAttachedToCurrentNote = true;
22 }

```

Upon the second visit of class `msrNote`, the part figured bass voice is created if figured bass elements are not to be ignored due to option `-ignore-musicxml-figured-bass`, `-ofigbass` and it has not been created yet:

```

1 void mxsr2msrSkeletonBuilder::visitEnd ( S_note& elt )
2 {
3     // ... ..
4
5     // are there figured bass attached to the current note?
6     if (fThereAreFiguredBassToBeAttachedToCurrentNote) {
7         if (gGlobalMxsr2msrOahGroup->getIgnoreFiguredBasses ()) {
8 #ifdef TRACING_IS_ENABLED
9             if (gGlobalTracingOahGroup->getTraceFiguredBasses ()) {
10                 gLogStream <<
11                     "Ignoring the figured bass elements" <<
12                     ", line " <<
13                     inputLineNumber <<
14                     std::endl;
15             }
16 #endif
17         }
18         else {
19             // create the part figured bass voice if not yet done
20             S_msrVoice
21             partFiguredBassVoice =
22                 createPartFiguredBassVoiceIfNotYetDone (
23                     inputLineNumber,

```

```

24         fCurrentPart);
25     }
26
27     fThereAreFiguredBassToBeAttachedToCurrentNote = false;
28
29     // ... ..
30 }

```

Creating the part figured bass voice is delegated to the part:

```

1 S_msrVoice mxsr2msrSkeletonBuilder::createPartFiguredBassVoiceIfNotYetDone (
2     int         inputLineNumber,
3     const S_msrPart&  part)
4 {
5     // is the figured bass voice already present in part?
6     S_msrVoice
7     partFiguredBassVoice =
8         part->
9         getPartFiguredBassVoice ();
10
11     if (! partFiguredBassVoice) {
12         // create the figured bass voice and append it to the part
13         partFiguredBassVoice =
14             part->
15             createPartFiguredBassVoice (
16                 inputLineNumber,
17                 fCurrentMeasureNumber);
18     }
19
20     return partFiguredBassVoice;
21 }

```

#### 64.4.1 First S\_figured\_bass visit

The first visit of S\_figured\_bass initializes the fields storing values to be gathered visiting subelements:

```

1 void mxsr2msrTranslator::visitStart ( S_figured_bass& elt )
2 {
3     int inputLineNumber =
4         elt->getInputLineNumber ();
5
6     #ifdef TRACING_IS_ENABLED
7         if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
8             gLogStream <<
9                 "--> Start visiting S_figured_bass" <<
10                 ", line " << inputLineNumber <<
11                 std::endl;
12         }
13     #endif
14
15     ++fFiguredBassVoicesCounter;
16
17     std::string parentheses = elt->getAttributeValue ("parentheses");
18
19     fCurrentFiguredBassParenthesesKind =
20         msrFiguredBassParenthesesKind::kFiguredBassParenthesesNo; // default value
21
22     if (parentheses.size ()) {
23         if (parentheses == "yes")
24             fCurrentFiguredBassParenthesesKind =
25                 msrFiguredBassParenthesesKind::kFiguredBassParenthesesYes;
26
27         else if (parentheses == "no")
28             fCurrentFiguredBassParenthesesKind =

```

```

29     msrFiguredBassParenthesesKind::kFiguredBassParenthesesNo;
30
31     else {
32         std::stringstream s;
33
34         s <<
35             "parentheses value " << parentheses <<
36             " should be 'yes' or 'no'";
37
38         musicxmlError (
39             gGlobalServiceRunData->getInputSourceName (),
40             inputLineNumber,
41             __FILE__, __LINE__,
42             s.str ());
43     }
44 }
45
46 fCurrentFiguredBassInputLineNumber    = -1;
47
48 fCurrentFigureNumber = -1;
49
50 fCurrentFigurePrefixKind = msrBassFigurePrefixKind::kBassFigurePrefix_UNKNOWN;
51 fCurrentFigureSuffixKind = msrBassFigureSuffixKind::kBassFigureSuffix_UNKNOWN;
52
53 fCurrentFiguredBassSoundingWholeNotes = Rational (0, 1);
54 fCurrentFiguredBassDisplayWholeNotes  = Rational (0, 1);
55
56 fOnGoingFiguredBass = true;
57 }

```

### 64.4.2 Second S\_figured\_bass visit

Upon the second visit of `S_figured_bass`, the class `msrFiguredBass` instance is created, populated and appended to `mxsr2msrTranslatorfPendingFiguredBassesList`:

```

1 void mxsr2msrTranslator::visitEnd ( S_figured_bass& elt )
2 {
3     int inputLineNumber =
4         elt->getInputLineNumber ();
5
6     #ifdef TRACING_IS_ENABLED
7         if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
8             gLogStream <<
9                 "--> End visiting S_figured_bass" <<
10                 ", line " << inputLineNumber <<
11                 std::endl;
12         }
13     #endif
14
15     // create the figured bass element
16     #ifdef TRACING_IS_ENABLED
17         if (gGlobalTracingOahGroup->getTraceFiguredBasses ()) {
18             gLogStream <<
19                 "Creating a figured bass" <<
20                 ", line " << inputLineNumber << ":" <<
21                 std::endl;
22         }
23     #endif
24
25     // create the figured bass element
26     // if the sounding whole notes is 0/1 (no <duration /> was found), JMI ???
27     // it will be set to the next note's sounding whole notes later
28     S_msFiguredBass
29         figuredBass =

```

```

30     msrFiguredBass::create (
31         inputLineNumber,
32         // JMI         fCurrentPart,
33         fCurrentFiguredBassSoundingWholeNotes,
34         fCurrentFiguredBassDisplayWholeNotes,
35         fCurrentFiguredBassParenthesesKind,
36         msrTupletFactor (1, 1));    // will be set upon next note handling
37
38 // attach pending figures to the figured bass element
39 if (! fPendingFiguredBassFiguresList.size ()) {
40     musicxmlWarning (
41         gGlobalServiceRunData->getInputSourceName (),
42         inputLineNumber,
43         "figured-bass has no figures contents, ignoring it");
44 }
45 else {
46     // append the pending figures to the figured bass element
47     for (S_msrBassFigure bassFigure : fPendingFiguredBassFiguresList) {
48         figuredBass->
49             appendFigureToFiguredBass (bassFigure);
50     } // for
51
52     // forget about those pending figures
53     fPendingFiguredBassFiguresList.clear ();
54
55     // append the figured bass element to the pending figured bass elements list
56     fPendingFiguredBassesList.push_back (figuredBass);
57 }
58
59 fOnGoingFiguredBass = false;
60 }

```

### 64.4.3 Attaching msrFiguredBass instances to notes

#### msrFiguredBass

The contents of `mxsr2msrTranslator.fPendingFiguredBassesList` is attached to the class `msrNote` instance in method

method `mxsr2msrTranslator::populateNote ()`:

```

1 void mxsr2msrTranslator::populateNote (
2     int         inputLineNumber,
3     const S_msrNote& newNote)
4 {
5     // ... ..
6
7     // handle the pending figured bass elements if any
8     if (fPendingFiguredBassesList.size ()) {
9         // get voice to insert figured bass elements into
10        S_msrVoice
11        voiceToInsertFiguredBassesInto =
12            fCurrentPart->
13                getPartFiguredBassVoice ();
14
15        // ... ..
16
17        handlePendingFiguredBasses (
18            newNote,
19            voiceToInsertFiguredBassesInto);
20
21        // reset figured bass counter
22        fFiguredBassVoicesCounter = 0;
23    }
24 }

```

### 64.4.4 Populating msrFiguredBass instances

In `src/formats/msr/mxsr2msrTranslator.cpp`, the class `msrFiguredBass` instances are populated further and attached to the note by method `mxsr2msrTranslator::handlePendingFiguredBasses ()`:

```

1 void mxsr2msrTranslator::handlePendingFiguredBasses (
2     const S_msrNote& newNote,
3     const S_msrVoice& voiceToInsertInto)
4 {
5     // ... ..
6
7     Rational
8     newNoteSoundingWholeNotes =
9     newNote->
10         getMeasureElementSoundingWholeNotes (),
11     newNoteDisplayWholeNotes =
12     newNote->
13         getNoteDisplayWholeNotes ();
14
15     while (fPendingFiguredBassesList.size ()) { // recompute at each iteration
16         S_msrFiguredBass
17         figuredBass =
18             fPendingFiguredBassesList.front ();
19
20         /*
21          Figured bass elements take their position from the first
22          regular note (not a grace note or chord note) that follows
23          in score order. The optional duration element is used to
24          indicate changes of figures under a note.
25          */
26
27         // set the figured bass element's sounding whole notes
28         figuredBass->
29             setMeasureElementSoundingWholeNotes (
30                 newNoteSoundingWholeNotes,
31                 "handlePendingFiguredBasses()");
32
33         // set the figured bass element's display whole notes JMI useless???
34         figuredBass->
35             setFiguredBassDisplayWholeNotes (
36                 newNoteDisplayWholeNotes);
37
38         // set the figured bass element's tuplet factor
39         figuredBass->
40             setFiguredBassTupletFactor (
41                 msrTupletFactor (
42                     fCurrentNoteActualNotes,
43                     fCurrentNoteNormalNotes));
44
45         // append the figured bass to newNote
46         newNote->
47             appendFiguredBassToNoteFiguredBassesList (
48                 figuredBass);
49
50         /* JMI
51          // get the figured bass voice for the current voice
52          S_msrVoice
53          voiceFiguredBassVoice =
54          voiceToInsertInto->
55              getRegularVoiceForwardLinkToFiguredBassVoice ();
56
57          // sanity check
58          mfAssert (
59              __FILE__, __LINE__,
60              voiceFiguredBassVoice != nullptr,
61              "voiceFiguredBassVoice is null");

```

```

62
63 // set the figuredBass's voice upLink
64 // only now that we know which figured bass voice will contain it
65 figuredBass->
66     setFiguredBassUpLinkToVoice (
67         voiceFiguredBassVoice);
68
69 // append the figured bass to the figured bass voice for the current voice
70 voiceFiguredBassVoice->
71     appendFiguredBassToVoice (
72         figuredBass);
73 */
74
75 // don't append the figured bass to the part figured bass voice
76 // before the note itself has been appended to the voice
77
78 // remove the figured bass from the list
79 fPendingFiguredBassesList.pop_front ();
80 } // while
81 }

```

```

1 %void mxsr2msrTranslator::copyNoteHarmoniesToChord (
2 % S_msrNote note, S_msrChord chord)
3 %{
4 % // copy note's harmony if any from the first note to chord
5 %
6 % const std::list<S_msrHarmony>&
7 %     noteHarmoniesList =
8 %     note->getNoteHarmoniesList ();
9 %
10 % if (noteHarmoniesList.size ()) {
11 %     std::list<S_msrHarmony>::const_iterator i;
12 %     for (i=noteHarmoniesList.begin (); i!=noteHarmoniesList.end (); ++i) {
13 %         S_msrHarmony harmony = (*i);
14 %
15 % #ifdef TRACING_IS_ENABLED
16 %         if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
17 %             gLogStream <<
18 %                 "Copying harmony '" <<
19 %                 harmony->asString () <<
20 %                 "' from note " << note->asString () <<
21 %                 " to chord '" << chord->asString () <<
22 %                 "'" <<
23 %                 std::endl;
24 %         }
25 % #endif
26 %
27 %         chord->
28 %             appendHarmonyToChord (harmony);
29 %
30 %     } // for
31 % }
32 %}
33 %

```

#### 64.4.5 Inserting S\_msrFiguredBass instances in the part figured bass voice

Method `msrVoice::appendNoteToVoice ()` in `src/formats/msr/msrNotes.cpp` inserts the figured bass elements in the part figured bass voice:

```

1 void msrVoice::appendNoteToVoice (const S_msrNote& note)
2 {
3     // ... ..
4

```

```

5 // are there figured bass elements attached to this note?
6 const std::list<S_msrFiguredBass>&
7   noteFiguredBassesList =
8     note->
9       getNoteFiguredBassesList ();
10
11 if (noteFiguredBassesList.size ()) {
12   // get the current part's figured bass voice
13   S_msrVoice
14     partFiguredBassVoice =
15       part->
16         getPartFiguredBassVoice ();
17
18   for (S_msrFiguredBass figuredBass : noteFiguredBassesList) {
19     // append the figured bass element to the part figured bass voice
20     partFiguredBassVoice->
21       appendFiguredBassToVoice (
22         figuredBass);
23   } // for
24 }
25 };

```

## 64.5 Translating figured bass from MSR to MSR

In `src/passes/msr2msr/msr2msrTranslator.cpp`, a newborn clone of the figured bass element is created upon the first visit, stored in `msr2msrTranslatorfCurrentFiguredBassClone`, and appended to the current non grace note clone, the current chord clone or to the current voice clone, if the latter is a figured bass voice:

```

1 void msr2msrTranslator::visitStart (S_msrFiguredBass& elt)
2 {
3   #ifdef TRACING_IS_ENABLED
4     if (gGlobalMsr0ahGroup->getTraceMsrVisitors ()) {
5       gLogStream <<
6         "--> Start visiting msrFiguredBass '" <<
7         elt->asString () <<
8         "' " <<
9         ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
10        ", line " << elt->getInputLineNumber () <<
11        std::endl;
12    }
13  #endif
14
15  // create a figured bass element newborn clone
16  fCurrentFiguredBassClone =
17    elt->
18      createFiguredBassNewbornClone (
19        fCurrentVoiceClone);
20
21  if (fOnGoingNonGraceNote) {
22    // append the figured bass to the current non-grace note clone
23    fCurrentNonGraceNoteClone->
24      appendFiguredBassToNoteFiguredBassesList (
25        fCurrentFiguredBassClone);
26
27    // don't append the figured bass to the part figured bass, JMI ???
28    // this will be done below
29  }
30
31  /* JMI
32  else if (fOnGoingChord) {
33    // register the figured bass in the current chord clone
34    fCurrentChordClone->

```

```

35     setChordFiguredBass (fCurrentFiguredBassClone); // JMI
36 }
37 */
38
39 else if (fOnGoingFiguredBassVoice) { // JMI
40     /*
41     // register the figured bass in the part clone figured bass
42     fCurrentPartClone->
43         appendFiguredBassToPartClone (
44             fCurrentVoiceClone,
45             fCurrentFiguredBassClone);
46     */
47     // append the figured bass to the current voice clone
48     fCurrentVoiceClone->
49         appendFiguredBassToVoiceClone (
50             fCurrentFiguredBassClone);
51 }
52
53 else {
54     std::stringstream s;
55
56     s <<
57     "figured bass is out of context, cannot be handled: " <<
58     elt->asShortString ();
59
60     msrInternalError (
61         gGlobalServiceRunData->getInputSourceName (),
62         elt->getInputLineNumber (),
63         __FILE__, __LINE__,
64         s.str ());
65 }
66 }

```

There are only fields updates upon the second visit:

```

1 void msr2msrTranslator::visitEnd (S_msrFiguredBass& elt)
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5             gLogStream <<
6             "--> End visiting msrFiguredBass '" <<
7             elt->asString () <<
8             "' " <<
9             ", line " << elt->getInputLineNumber () <<
10             std::endl;
11         }
12     #endif
13
14     fCurrentFiguredBassClone = nullptr;
15 }

```

## 64.6 Translating figured bass from MSR to LPSR

The same occurs in `src/passes/msr2lpsr/msr2lpsrTranslator.cpp`: a newborn clone of the figured bass element is created and appended to the current non grace note clone, the current chord clone or to the current voice clone, if the latter is a figured bass voice:

```

1 void msr2lpsrTranslator::visitStart (S_msrFiguredBass& elt)
2 {
3     #ifdef TRACING_IS_ENABLED
4         if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5             gLogStream <<
6             "--> Start visiting msrFiguredBass '" <<

```



```

7      elt->asString () <<
8      ">" <<
9      ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
10     ", line " << elt->getInputLineNumber () <<
11     std::endl;
12 }
13 #endif
14
15 // create a figured bass newborn clone
16 fCurrentFiguredBassClone =
17     elt->
18     createFiguredBassNewbornClone (
19         fCurrentVoiceClone);
20
21 if (fOnGoingNonGraceNote) {
22     // append the figured bass to the current non-grace note clone
23     fCurrentNonGraceNoteClone->
24         appendFiguredBassToNoteFiguredBassesList (
25             fCurrentFiguredBassClone);
26
27     // don't append the figured bass to the part figured bass, JMI ???
28     // this will be done below
29 }
30
31 /* JMI
32 else if (fOnGoingChord) {
33     // register the figured bass in the current chord clone
34     fCurrentChordClone->
35         setChordFiguredBass (fCurrentFiguredBassClone); // JMI
36 }
37 */
38
39 else if (fOnGoingFiguredBassVoice) { // JMI
40     /*
41     // register the figured bass in the part clone figured bass
42     fCurrentPartClone->
43         appendFiguredBassToPartClone (
44             fCurrentVoiceClone,
45             fCurrentFiguredBassClone);
46     */
47     // append the figured bass to the current voice clone
48     fCurrentVoiceClone->
49         appendFiguredBassToVoiceClone (
50             fCurrentFiguredBassClone);
51 }
52
53 else {
54     std::stringstream s;
55
56     s <<
57     "figured bass is out of context, cannot be handled: " <<
58     elt->asShortString ();
59
60     msrInternalError (
61         gGlobalServiceRunData->getInputSourceName (),
62         elt->getInputLineNumber (),
63         __FILE__, __LINE__,
64         s.str ());
65 }
66 }

```

Here too, there are only fields updates upon the second visit of `S_msrFiguredBass` instances:

```

1 void msr2lpsrTranslator::visitEnd (S_msrFiguredBass& elt)
2 {
3 #ifdef TRACING_IS_ENABLED

```

```

4   if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5       gLogStream <<
6       "--> End visiting msrFiguredBass '" <<
7       elt->asString () <<
8       "' " <<
9       ", line " << elt->getInputLineNumber () <<
10      std::endl;
11  }
12  #endif
13
14  fCurrentFiguredBassClone = nullptr;
15  }

```

## 64.7 Translating figured bass from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

There is only one visit of class `msrFiguredBass` instances in `src/passes/lpsr2lilypond/lpsr2lilypondTranslator.cpp`.

The LilyPond code is generated only if the figured bass element belongs to a figured bass voice: this is where denormalization ends in the workflow:

```

1  void msr2lpsrTranslator::visitStart (S_msrFiguredBass& elt)
2  {
3      #ifdef TRACING_IS_ENABLED
4          if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5              gLogStream <<
6              "--> Start visiting msrFiguredBass '" <<
7              elt->asString () <<
8              "' " <<
9              ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
10             ", line " << elt->getInputLineNumber () <<
11             std::endl;
12         }
13     #endif
14
15     // create a figured bass newborn clone
16     fCurrentFiguredBassClone =
17         elt->
18         createFiguredBassNewbornClone (
19             fCurrentVoiceClone);
20
21     if (fOnGoingNonGraceNote) {
22         // append the figured bass to the current non-grace note clone
23         fCurrentNonGraceNoteClone->
24         appendFiguredBassToNoteFiguredBassesList (
25             fCurrentFiguredBassClone);
26
27         // don't append the figured bass to the part figured bass, JMI ???
28         // this will be done below
29     }
30
31     /* JMI
32     else if (fOnGoingChord) {
33         // register the figured bass in the current chord clone
34         fCurrentChordClone->
35         setChordFiguredBass (fCurrentFiguredBassClone); // JMI
36     }
37     */
38
39     else if (fOnGoingFiguredBassVoice) { // JMI
40         /*

```

```
41 // register the figured bass in the part clone figured bass
42 fCurrentPartClone->
43   appendFiguredBassToPartClone (
44     fCurrentVoiceClone,
45     fCurrentFiguredBassClone);
46   */
47 // append the figured bass to the current voice clone
48 fCurrentVoiceClone->
49   appendFiguredBassToVoiceClone (
50     fCurrentFiguredBassClone);
51 }
52
53 else {
54   std::stringstream s;
55
56   s <<
57     "figured bass is out of context, cannot be handled: " <<
58     elt->asShortString ();
59
60   msrInternalError (
61     gGlobalServiceRunData->getInputSourceName (),
62     elt->getInputLineNumber (),
63     __FILE__, __LINE__,
64     s.str ());
65 }
66 }
```

## Chapter 65

# Lyrics handling

Lyrics are presented at section [19.45](#) [Lyrics], page [198](#).

**65.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**65.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**65.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**65.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**65.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

## Chapter 66

# MIDI handling

MIDI is presented at section [19.46](#) [MIDI], page [199](#).

At the day of this writing, MIDI handling is partial, i.e. not all MIDI elements present in MusicXML are incorporated in MSR and no MIDI data can generated generated by MusicFormats.

**66.1** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**66.2** Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

**66.3** Translating from MSR to MSR ([src/passes/msr2msr/](#))

**66.4** Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

**66.5** Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

---

## Part XIII

# MusicFormats Scripting Language (MFSL)

## Chapter 67

# MFSL (MusicFormats Scripting Language)

### 67.1 A script example

This script illustrates the basic features of MFSL:

```
1 #!//Users/jacquesmenu/musicformats-git-dev/build/bin/mfsl
2
3 # the MusicFormats tool to be used
4 tool : xml2ly
5
6 # the input file
7 input :test.mfsl
8
9 # parts
10 -keep-musicxml-part-id P1
11
12 # the voices choice
13 choice VOICES_CHOICE : voice1Only | voice2Only ;
14 # could be : choice VOICES_CHOICE : ... .. ;
15
16 set VOICES_CHOICE = voice1Only ;
17 # change this to voice2Only to switch to another subset of options
18 # could even be parameter to the script such a $1
19
20 # choose which options to use according to VOICES_CHOICE
21 case VOICES_CHOICE :
22   voice1Only:
23     -title "Joli morceau - voix 1"
24     -ignore-msr-voice Part_P0ne_Staff_One_Voice_Two
25   ;
26
27   voice2Only:
28     -title "Joli morceau - voix 2"
29     --ignore-msr-voice Part_P0ne_Staff_One_Voice_One
30
31     -display-options-values
32
33     -global-staff-size 25.5
34   ;
35 ;
```

This first line of an MFSL script is the so-called *shebang* containing the path to the interpreter, allow for running such scripts by their name provided they are made executable.

## 67.2 Implementation principles

MFSL is implemented with the `flex` and `bison` C++ code generators:

- `src/interpreters/mfsl/mfslScanner.ll` contains the `flex` lexical description of MFSL.  
It is used to create `src/interpreters/mfsl/mfslScanner.cpp`;
- `src/interpreters/mfsl/mfslParser.yy` is the syntax and semantics description of MFSL.  
From it, `bison` creates `src/interpreters/mfsl/mfslParser.h`, `src/interpreters/mfsl/mfslParser.cpp` and `src/interpreters/mfsl/mfslParser.output`.  
The latter file can be used to check the grammar, in particular if LR conflicts are detected;
- communication between the code generated this way is done by a so-called *driver*, along the lines of the C++-calc example provided by `bison` v3.8.1;
- the way the tokens description is shared by the scanner and parser is described at section 67.7 [Tokens description], page 366;
- the whole power of OAH is used to handle the contents of MFSL scripts as well as the options to the MFSL interpreter itself.

Only the predefined `bool` type is used, since the generated C++ code relies on this. This is why `getValue ()` is used in `src/clisamples/mfsl.cpp`:

```

1  std::string      theMfTool;
2  std::string      theInputFile;
3  oahOptionsAndArguments optionsAndArguments;
4
5  err =
6      launchMfslInterpreter (
7          inputSourceName,
8          traceScanning.getValue (),
9          traceParsing.getValue (),
10         displayTokens.getValue (),
11         displayNonTerminals.getValue (),
12         theMfTool,
13         theInputFile,
14         optionsAndArguments);

```

## 67.3 The contents of the MFSL folder

`src/interpreters/mfsl/location.hh` defines the `yy::location` class, that contains the script file name and input line number:

```

1  jacquesmenu@macmini: ~/musicformats-git-dev/src/interpreters/mfsl > ls -sal
2  total 776
3   0 drwxr-xr-x  28 jacquesmenu  staff      896 Mar 15 05:16 .
4   0 drwxr-xr-x@  4 jacquesmenu  staff     128 Mar 13 00:47 ..
5  16 -rw-r--r--@  1 jacquesmenu  staff    6148 Mar 14 10:18 .DS_Store
6   8 -rw-r--r--@  1 jacquesmenu  staff    1266 Mar 15 05:15 Makefile
7  16 -rw-r--r--@  1 jacquesmenu  staff    7864 Mar 15 05:16 location.hh
8  24 -rw-r--r--@  1 jacquesmenu  staff   10106 Mar 14 18:26 mfslBasicTypes.cpp
9  16 -rw-r--r--@  1 jacquesmenu  staff    4568 Mar 14 18:16 mfslBasicTypes.h
10   8 -rw-r--r--@  1 jacquesmenu  staff    1585 Mar 15 05:12 mfslDriver.cpp
11   8 -rw-r--r--@  1 jacquesmenu  staff    3413 Mar 15 05:12 mfslDriver.h
12   8 -rw-r--r--@  1 jacquesmenu  staff    3041 Mar  9 07:35 mfslInterpreterComponent.cpp
13   8 -rw-r--r--@  1 jacquesmenu  staff     661 Mar  9 07:02 mfslInterpreterInterface.h

```



```

14 24 -rw-r--r--@ 1 jacquesmenu staff 11981 Mar 10 11:38 mfs1InterpreterInsiderHandler.
    cpp
15 16 -rw-r--r--@ 1 jacquesmenu staff 5270 Mar 10 07:11 mfs1InterpreterInsiderHandler.h
16 8 -rw-r--r--@ 1 jacquesmenu staff 1161 Mar 15 05:13 mfs1InterpreterInterface.h
17 16 -rw-r--r--@ 1 jacquesmenu staff 7116 Mar 14 15:53 mfs1Interpreter0ah.cpp
18 16 -rw-r--r--@ 1 jacquesmenu staff 4692 Mar 14 15:51 mfs1Interpreter0ah.h
19 24 -rw-r--r--@ 1 jacquesmenu staff 10070 Mar 14 15:53 mfs1InterpreterRegularHandler.
    cpp
20 8 -rw-r--r--@ 1 jacquesmenu staff 3533 Mar 9 08:22 mfs1InterpreterRegularHandler.h
21 88 -rw-r--r-- 1 jacquesmenu staff 43880 Mar 15 05:16 mfs1Parser.cpp
22 96 -rw-r--r-- 1 jacquesmenu staff 45868 Mar 15 05:16 mfs1Parser.h
23 24 -rw-r--r--@ 1 jacquesmenu staff 10722 Mar 13 16:57 mfs1Parser.output
24 16 -rw-r--r--@ 1 jacquesmenu staff 5930 Mar 14 18:19 mfs1Parser.yy
25 136 -rw-r--r-- 1 jacquesmenu staff 68514 Mar 15 05:16 mfs1Scanner.cpp
26 24 -rw-r--r--@ 1 jacquesmenu staff 11251 Mar 15 05:12 mfs1Scanner.ll
27 144 -rw-r--r--@ 1 jacquesmenu staff 71091 Mar 15 05:14 mfs1Scanner.log
28 8 -rw-r--r--@ 1 jacquesmenu staff 2047 Mar 9 11:45 mfs1Wae.cpp
29 8 -rw-r--r--@ 1 jacquesmenu staff 3681 Mar 9 11:44 mfs1Wae.h
30 8 -rwxr-xr-x@ 1 jacquesmenu staff 817 Mar 14 18:20 test.mfs1

```

## 67.4 The MFSL basic types

## 67.5 The MFSL Makefile

This Makefile is quite simple: the options to flex and bison are placed in `src/interpreters/mfs1/mfs1Scanner.ll` and `src/interpreters/mfs1/mfs1Parser.yy`, respectively:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/interpreters/mfs1 > cat Makefile
2 # ... ..
3
4 # variables
5 # -----
6
7 MAKEFILE = Makefile
8
9 GENERATED_FILES = mfs1Parser.h mfs1Scanner.cpp mfs1Parser.cpp
10
11 BISON = bison
12 FLEX = flex
13
14 CXXFLAGS = -I.. -DMAIN
15
16
17 # implicit target
18 # -----
19
20 all : $(GENERATED_FILES)
21
22
23 # generation rules
24 # -----
25
26 mfs1Scanner.cpp : $(MAKEFILE) mfs1Scanner.ll
27     $(FLEX) -omfs1Scanner.cpp mfs1Scanner.ll
28
29
30 mfs1Parser.h mfs1Parser.cpp : $(MAKEFILE) mfs1Parser.yy
31     $(BISON) --defines=mfs1Parser.h -o mfs1Parser.cpp mfs1Parser.yy
32
33
34 # clean
35 # -----

```

```

36
37 clean:
38   rm -f $(GENERATED_FILES)

```

## 67.6 Locations handling

## 67.7 Tokens description

The tokens are described in `src/interpreters/mfsl/mfslParser.yy`, such as:

```
1 %token <std::string> OPTION "option"
```

Both `OPTION` and `"option"` can be used in the productions, but the grammar is more readable if the capitalized name is used:

```

1 Option
2   : OPTION
3     {
4       if (drv.getDisplayNonTerminals ()) {
5         gLogStream <<
6           " ==> option " << $1 <<
7           std::endl << std::endl;
8       }
9
10      $$ = oahOptionNameAndValue::create ($1, "");
11    }
12
13 | OPTION OptionValue
14   {
15     if (drv.getDisplayNonTerminals ()) {
16       gLogStream <<
17         " ==> option " << $1 << ' ' << $2 <<
18         std::endl << std::endl;
19     }
20
21     $$ = oahOptionNameAndValue::create ($1, $2);
22   }
23 ;

```

In case of error, `"option"` is used to display a message to the user.

The name `OPTION` is used in `src/interpreters/mfsl/mfslScanner.ll` prefixed by `yy::parser::make_`:

```

1 "--{name} |
2 "--{name} {
3   if (drv.getTraceTokens ()) {
4     gLogStream << "--> " << drv.getScannerLocation () <<
5     ": option [" << yytext << ']' <<
6     std::endl;
7   }
8   return yy::parser::make_OPTION (yytext, loc);
9 }

```

The suffix after `make_` has to be defined in the `src/interpreters/mfsl/mfslParser.yy` for this to do the link between the Flex-generated and Bison-generated code:

```
1 %token <std::string> OPTION "option"
```

In `src/interpreters/mfsl/mfslParser.cpp`, this becomes:

```
1 case symbol_kind::S_OPTION: // "option"
```

We don't have to create method `yy::parser::make_OPTION ()` ourselves, though: it is taken care of by Bison itself, since it returns a type `char*`.

The `calc++` example in the `bison` documentation contains the case of numbers:

```
1 %{
2 // A number symbol corresponding to the value in S.
3 yy::parser::symbol_type
4 make_NUMBER (const std::string &s, const yy::parser::location_type& loc);
5 %}
6
7 // ... ..
8
9 yy::parser::symbol_type
10 make_NUMBER (const std::string &s, const yy::parser::location_type& loc)
11 {
12     errno = 0;
13     long n = strtol (s.c_str(), NULL, 10);
14     if (! (INT_MIN <= n && n <= INT_MAX && errno != ERANGE))
15         throw yy::parser::syntax_error (loc, "integer is out of range: " + s);
16     return yy::parser::make_NUMBER ((int) n, loc);
17 }
```

## 67.8 The driver

Class `mfsldriver` contains everything needed to let the code generated by `flex` and `bison` communicate with each other, as well as any work variables needed during the analysis of MFSL input. This latter point allows for multiple analyzers to coexist.

`src/interpreters/mfsldriver.h` contains a prototype of function `yylex ()`:

```
1 // -----
2 // Give Flex the prototype of yylex we want ...
3 # define YY_DECL \
4     yy::parser::symbol_type yylex (mfsldriver& drv)
5 // ... and declare it for the parser's sake.
6 YY_DECL;
```

Then it contains the declaration of class `mfsldriver`:

```
1 // Conducting the whole scanning and parsing of MFSL
2 class mfsldriver
3 {
4     public:
5
6         // constants
7         // -----
8
9         static const std::string K_ALL_PSEUDO_LABEL_NAME;
10
11         // // constructor/destructor
12         // -----
13
14         mfsldriver ();
15
16         virtual ~mfsldriver ();
17
18         // ... ..
19 }
```

```

20 public:
21
22     // public services
23     // -----
24
25     // run the parser, return 0 on success
26     int                parseInput_Pass1 ();
27
28     // handling the scanner
29     void                scanBegin ();
30     void                scanEnd ();
31
32     // ... ..
33
34 private:
35
36     // private fields
37     // -----
38
39     // the name of the MusicFormats tool
40     std::string          fTool;
41
42     // the name of the MusicFormats script
43     std::string          fScriptName;
44
45     // the names of the input sources
46     std::list<std::string>
47         fInputSoucesList;
48
49
50     // scanning
51     bool                fTraceScanning;
52     mfs::location        fScannerLocation;
53
54     // ... ..
55 };

```

The definitions are placed in two files due to the specificity of the sharing of variables and function in the **flex** and **bison**-generated code:

- `src/interpreters/mfs/mfsDriver.cpp` contains method `mfsDriver::parseInput_Pass1 ()`, that runs the parser:

```

1 int mfsDriver::parseInput_Pass1 ()
2 {
3     // initialize scanner location
4     fScannerLocation.initialize (
5         &fScriptName);
6
7     // begin scan
8     scanBegin ();
9
10    if (fScriptName.empty () || fScriptName == "-") {
11        fScriptName = "stdin"; // nicer for warning and error messages
12    }
13
14    // do the parsing
15    mfs::parser theParser (*this);
16
17    theParser.set_debug_level (
18        fTraceParsing);
19
20    int parseResult = theParser ();
21
22    // end scan

```

```

23 |     scanEnd ();
24 |
25 |     // ... ..
26 |
27 |     // do the final semantics check
28 |     finalSemanticsCheck ();
29 |
30 |     return parseResult;
31 | }

```

- the remaining code is placed in the third part (service code) of `src/interpreters/mfsl/mfslScanner.ll`, since it needs to access variables in the code generated by `flex`:

```

1 | void mfslDriver::scanBegin ()
2 | {
3 |     yy_flex_debug = fTraceScanning;
4 |
5 |     if (fScriptName.empty () || fScriptName == "-") {
6 |         yyin = stdin;
7 |     }
8 |
9 |     else if (!(yyin = fopen (fScriptName.c_str (), "r")))
10 |     {
11 |         std::stringstream s;
12 |
13 |         char*
14 |             errorString =
15 |                 strerror (errno);
16 |
17 |         if (errorString != nullptr) {
18 |             s <<
19 |                 "cannot open " <<
20 |                 fScriptName << ": " <<
21 |                 errorString <<
22 |                 std::endl;
23 |
24 |             mfslFileError (
25 |                 fScriptName,
26 |                 s.str ());
27 |         }
28 |     }
29 | }
30 |
31 | void mfslDriver::scanEnd ()
32 | {
33 |     fclose (yyin);
34 | }

```

## 67.9 Lexical analysis

The lexical definition of MFSL in `src/interpreters/mfsl/mfslScanner.ll` is described below.

### 67.9.1 Flex options

The `prefix` is used to allow for multiple `flex`-generated analyzers to coexist:

```

1 | %option prefix="mfsl"
2 |
3 | %option yylineno
4 |
5 | %option noyywrap

```

```

6
7 %option nounput noinput debug interactive

```

### 67.9.2 Flex regular expressions

The basic ones are:

```

1 blank          [ \t\r]
2 endOfLine      [\n]
3 character      .
4
5 letter         [A-Za-zéèëääöðùí]
6 digit          [0-9]
7
8 name           {letter}(_|-|\.|{letter}|{digit})*
9 integer        {digit}+
10 exponent       [eE][+-]?{integer}
11
12 singleQuote    [']
13 doubleQuote    ["]
14 tabulator      [\t]
15 backSlash      [\\]
16
17 ... ..
18
19 %{
20     // Code run each time a pattern is matched.
21     # define YY_USER_ACTION  loc.columns (yyleng);
22 %}

```

Some exclusive modes are used for strings and comments:

```

1 %x          SINGLE_QUOTED_STRING_MODE
2 %x          DOUBLE_QUOTED_STRING_MODE
3
4 %x          COMMENT_TO_END_OF_LINE_MODE
5 %x          PARENTHEZIZED_COMMENT_MODE

```

Strings must be stored in a private buffer:

```

1 /* strings */
2
3 #define          STRING_BUFFER_SIZE 1024
4 char            pStringBuffer [STRING_BUFFER_SIZE];
5
6 // A handy shortcut to the location held by the mfsldriver
7 mfsll::location& loc = drv.getScannerLocationNonConst ();

```

Locating the tokens in the the MFSL input text is done with:

```

1 // Code run each time yylex() is called
2 loc.step ();

```

This lead for example to:

```

1 {blank} {
2     loc.step ();
3 }
4
5 {endOfLine} {
6     loc.lines (yyleng); loc.step ();
7 }

```

The numbers are handled by:

```

1 {integer} "." {integer} ({exponent})? |
2 {integer}{exponent} {
3   if (drv.getTraceTokens ()) {
4     gLogStream <<
5       "--> " << drv.getScannerLocation () <<
6       " double: " << yytext <<
7       std::endl;
8   }
9   return yy::parser::make_DOUBLE (yytext, loc);
10 }
11
12 {integer} {
13   if (drv.getTraceTokens ()) {
14     gLogStream <<
15       "--> " << drv.getScannerLocation () <<
16       " integer: " << yytext <<
17       std::endl;
18   }
19   return yy::parser::make_INTEGER (yytext, loc);
20 }

```

The MFSL keywords are handled with the make\_... facility:

```

1 "tool" {
2   if (drv.getTraceTokens ()) {
3     gLogStream <<
4       "--> " << drv.getScannerLocation () << ": " << yytext <<
5       std::endl;
6   }
7   return yy::parser::make_TOOL (loc);
8 }

```

The names and the options are handled by:

```

1 {name} {
2   if (drv.getDisplayTokens ()) {
3     gLogStream << "--> " << drv.getScannerLocation () <<
4       ": name [" << yytext << ']' <<
5       std::endl;
6   }
7
8   loc.begin.column += yyleng;
9   loc.step ();
10
11   return
12     mfs1::parser::make_NAME (yytext, loc);
13 }
14
15
16 "--{name} |
17 "--{name} {
18   if (drv.getTraceTokens ()) {
19     gLogStream << "--> " << drv.getScannerLocation () <<
20       ": option [" << yytext << ']' <<
21       std::endl;
22   }
23   return yy::parser::make_OPTION (yytext, loc);
24 }
25
26
27
28 "(" {
29   if (drv.getTraceTokens ()) {

```

```

31     gLogStream <<
32     "--> " << drv.getScannerLocation () << ": " << yytext <<
33     std::endl;
34 }
35 return yy::parser::make_LEFT_PARENTHESIS (loc);
36 }

```

The catchall rule issues an error message:

```

1 . {
2     throw mfs1::parser::syntax_error (
3         loc,
4         "### invalid character: " + std::string (yytext));
5 }

```

And the end of the MFSL input is handled this way:

```

1 <<EOF>> {
2     return
3     mfs1::parser::make_YEEOF (loc);
4 }

```

## 67.10 Syntax and semantic analysis

### 67.10.1 Bison options for MFSL

Setting `api.prefix` allows for multiple analyzers to coexist:

```

1 %skeleton "lalr1.cc" // -*- C++ -*-
2 %require "3.8.1"
3 %defines
4
5 %define api.prefix {mfs1}
6
7 %define api.token.raw
8
9 %define api.token.constructor
10 %define api.value.type variant
11 %define parse.assert
12
13 %code requires {
14     #include <string>
15
16     class mfs1Driver;
17 }
18
19 // the parsing context
20 %param { mfs1Driver& drv } // declaration, any parameter name is fine
21
22 %verbose // to produce mfs1Parser.output
23
24 %locations
25
26 // other Bison options
27 %define parse.trace
28 %define parse.error detailed
29 %define parse.lac full
30 // %define api.pure full
31
32 %printer { yyo << $$; } <*>;
33
34

```



```

35 %code {
36     #include "mfsBasicTypes.h"
37 }

```

### 67.10.2 The MFSL tokens

The MFSL tokens are:

```

1 %define api.token.prefix {MFSL_TOK_}
2
3 %token
4     BAR            "|"
5     AMPERSAND      "&"
6     EQUAL          "="
7     SEMICOLON      ";"
8     COLON          ":"
9     COMMA          ","
10
11     TOOL           "tool"
12     INPUT          "input"
13
14     CHOICE          "choice"
15     DEFAULT         "default"
16
17     CASE            "case"
18
19     SELECT          "select"
20     ALL             "all"
21 ;
22
23 %code {
24     #include "mfsDriver.h"
25 }
26
27 %token <std::string> INTEGER "integer number"
28 %token <std::string> DOUBLE  "double number"
29
30 %token <std::string> SINGLE_QUOTED_STRING "single quoted_string"
31 %token <std::string> DOUBLE_QUOTED_STRING "double quoted_string"
32
33 %token <std::string> NAME "name"
34
35 %token <std::string> OPTION "option"

```

### 67.10.3 The MFSL non-terminals and axiom

They are:

```

1 // the MFSL non-terminals
2 // -----
3
4 %nterm <std::string> Number
5
6 %nterm <std::string> SingleString
7 %nterm <std::string> std::string
8
9 %nterm <std::string> OptionValue
10
11 %nterm <std::string> LabelName
12
13
14 // the MFSL axiom

```

```

15 //-----
16
17 %start Script

```

## 67.11 Interface to the MFSL parser

This is provided by `src/interpreters/mfsl/mfslInterpreterInterface.h`:

```

1 EXP extern mfMusicformatsErrorKind launchMfslInterpreter (
2     const std::string&      inputSourceName,
3     bool                    traceScanning,
4     bool                    traceParsing,
5     bool                    displayTokens,
6     bool                    displayNonTerminals,
7     std::string&            theMfTool,
8     std::string&            theInputFile,
9     oahOptionsAndArguments& optionsAndArguments);

```

The definition of this function is placed in `src/interpreters/mfsl/mfslScanner.ll`:

```

1 mfMusicformatsErrorKind launchMfslInterpreter (
2     const std::string&      inputSourceName,
3     bool                    traceScanning,
4     bool                    traceParsing,
5     bool                    displayTokens,
6     bool                    displayNonTerminals,
7     std::string&            theMfTool,
8     std::string&            theInputFile,
9     oahOptionsAndArguments& optionsAndArguments)
10 {
11     mfMusicformatsErrorKind
12         result =
13         mfMusicformatsErrorKind::kMusicformatsError_NONE;
14
15     mfslDriver
16         theDriver (
17             traceScanning,
18             traceParsing,
19             displayTokens,
20             displayNonTerminals);
21
22     int parseResult =
23         theDriver.parseFile (inputSourceName);
24
25     gLogStream <<
26         "--> parseResult: " << parseResult <<
27         std::endl;
28
29     if (! parseResult) {
30         result =
31             mfMusicformatsErrorKind::kMusicformatsErrorInvalidFile;
32     }
33
34     gLogStream <<
35         "inputFileName: " << theDriver.getInputFileName () <<
36         std::endl <<
37         "toolName: " << theDriver.getToolName () <<
38         std::endl;
39
40     theMfTool      = theDriver.getToolName ();
41     theInputFile   = theDriver.getInputFileName ();
42
43     return result;
44 }

```

## 67.12 Running the example MFSL script

Let's show show the MFSL interpreter uses the options above:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/interpreters/mfsl > ./test.mfsl -display-
  tokens -display-non-terminals -display-options-values
2 The options values for //Users/jacquesmenu/musicformats-git-dev/build/bin/mfsl are:
3 MFSL group (-help-mfsl-group, -hmfsl-group), 2 atoms chosen:
4 -----
5 MFSL (-help-mfsl, -hmfsl), 2 atoms chosen:
6 fTraceTokens : true, set by user
7 fDisplayNonTerminals : true, set by user
8
9 Options and help group (-help-oah-group, -hoah-group), 1 atom chosen:
10 -----
11 Options and help (-help-oah, -hoah), 1 atom chosen:
12 fDisplayOptionsValues : true, set by user
13
14
15 --> ./test.mfsl:2.1-39: tool
16 --> ./test.mfsl:2.41: :
17 --> ./test.mfsl:2.43-48: name [xml2ly]
18 ==> tool: xml2ly
19
20 --> ./test.mfsl:4.1-22: input
21 --> ./test.mfsl:4.24: :
22 --> ./test.mfsl:4.25-33: name [test.mfsl]
23 ==> input: test.mfsl
24
25 --> test.mfsl:6.11-32: option [-keep-musicxml-part-id]
26 --> test.mfsl:6.34-35: name [P1]
27 ==> option -keep-musicxml-part-id P1
28
29 --> test.mfsl:8.1-26: choice
30 --> test.mfsl:8.28-40: name [VOICES_CHOICE]
31 --> test.mfsl:8.42: :
32 --> test.mfsl:8.44-53: name [voice10only]
33 --> test.mfsl:8.55: |
34 --> test.mfsl:8.57-66: name [voice20only]
35 --> test.mfsl:8.68: ;
36 ==> ChoiceDeclaration VOICES_CHOICE : ...
37
38 --> test.mfsl:10.1-3: set
39 --> test.mfsl:10.5-17: name [VOICES_CHOICE]
40 --> test.mfsl:10.19: =
41 --> test.mfsl:10.21-30: name [voice10only]
42 --> test.mfsl:10.32: ;
43 ==> ChoiceSetting, set VOICES_CHOICE = voice10only
44
45 --> test.mfsl:12.1-61: case
46 --> test.mfsl:12.63-75: name [VOICES_CHOICE]
47 --> test.mfsl:12.77: :
48 --> test.mfsl:13.2-11: name [voice10only]
49 --> test.mfsl:13.12: :
50 --> test.mfsl:14.5-10: option [-title]
51 --> test.mfsl:14.12-34: double quoted std::string ["]
52 ==> option -title "
53
54 --> test.mfsl:15.5-21: option [-ignore-msr-voice]
55 --> test.mfsl:15.23-51: name [Part_P0ne_Staff_One_Voice_Two]
56 ==> option -ignore-msr-voice Part_P0ne_Staff_One_Voice_Two
57
58 --> test.mfsl:16.3: ;
59 ==> Case voice10only : ...
60
61 --> test.mfsl:18.2-11: name [voice20only]

```

```

62 --> test.mfsl:18.12: :
63 --> test.mfsl:19.5-10: option [-title]
64 --> test.mfsl:19.12-34: double quoted std::string ["]
65 ==> option -title "
66
67 --> test.mfsl:20.5-22: option [--ignore-msr-voice]
68 --> test.mfsl:20.24-52: name [Part_P0ne_Staff_One_Voice_One]
69 ==> option --ignore-msr-voice Part_P0ne_Staff_One_Voice_One
70
71 --> test.mfsl:22.5-27: option [-display-options-values]
72 --> test.mfsl:24.5-22: option [-global-staff-size]
73 ==> option -display-options-values
74
75 --> test.mfsl:24.24-27 double: 25.5
76 ==> option -global-staff-size 25.5
77
78 --> test.mfsl:25.3: ;
79 ==> Case voice20nly : ...
80
81 --> test.mfsl:26.1: ;
82 ==> CaseStatement, VOICES_CHOICE : ...
83
84 --> parseResult: 0
85 ==> inputFileName: test.mfsl
86 ==> toolName:      xml2ly
87 jacquesmenu@macmini: ~/musicformats-git-dev/src/interpreters/mfsl >

```

### 67.12.1 Error recovery

The MFSL interpreter uses a variant of the *stopper sets* method that was present in the early Pascal and Pascal-S converters. The latter passed a set of tokens not to be overtaken to the procedures in charge of accepting the various statements in the language. Strangely enough, this was not done for declarations.

We use a stack of tokens sets that grows and shrinks in parallel with the accepting functions, to know more contextual informations when deciding wether to consume a token or not. The corresponding term is it shift when building the analysis tables in LR technology.

---

## Part XIV

# Music Scores Description Language (MSDL)

## Chapter 68

# MSDL (Music Scores Description Language)

MSDL is an attempt at a description of music score in a non-linear way, much like a painter puts touches of paint on his work. This is also what users do with GUI music scoring applications, but scores textual descriptions such as LilyPond and Guido impose a linear, left to right, writing of the scores contents.

Contrary to LilyPond, the `|` token in MSDL is not the end of a measure. Writing `|2` means that the music that follows will be placed in a new layer in measure 2.

### 68.1 Main features of MSDL

They are:

- note are written much like in LilyPond such as `b2...`;
- the keywords such as `pitches` and `music`, are reserved;
- they are available in a number of languages such as english, french, german and italian. It is easy to add other languages;

A first, limited converter is provided by MusicFormats with service `msdl`. It also performs reserved keywords translation from one language to another:

### 68.2 MSDL basic types

Some types used throughout MSR are defined in `src/formats/msdl/msdlEnumTypes.h/.cpp`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msdl > egrep -rIn  '^// '
    msdlEnumTypes.h
2 msdlEnumTypes.h:28:// user languages
3 msdlEnumTypes.h:52:// comments types
4 msdlEnumTypes.h:74:// initialization
```

## 68.3 What the MSDL converter does

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/msdl > msdl -about
2 What msdlConverter does:
3
4     This multi-pass converter performs various passes depending on the output generated,
5     which should be specified a '-lilypond', '-braille', '-musicxml' or '-guido' option.
6
7     Other passes are performed according to the options, such as
8     displaying views of the internal data or printing a summary of the score.
9
10    The activity log and warning/error messages go to standard error.
11
12    The output format is selected via options.

```

### 68.3.1 LilyPond generation

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/msdl > msdl -lilypond -about
2 What msdlConverter does:
3
4     This multi-pass converter basically performs 3 passes when generating LilyPond output
5     output:
6
7     Pass 1:  converts the MSDL input into a first MSR
8     Pass 2:  converts the first MSR into a second MSR;
9     Pass 3:  converts the second MSR into a
10             LilyPond Score Representation (LPSR);
11     Pass 4:  converts the LPSR to LilyPond 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.

```

### 68.3.2 Braille generation

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/msdl > msdl -braille -about
2 What msdlConverter does:
3
4     This multi-pass converter basically performs 4 passes when generating braille output
5     output:
6
7     Pass 1:  converts the MSDL input into a first MSR
8     Pass 2:  converts the first MSR into a second MSR;
9     Pass 3a: converts the second MSR into a
10             Braille Score Representation (BSR)
11             containing one Braille page per MusicXML page;
12     Pass 3b: converts the BSR into another BSR
13             with as many Braille pages as needed
14             to fit the line and page lengths;
15     Pass 4:  converts the BSR to Braille text
16             and writes it to standard output.)
17
18     In this preliminary version, pass 2b merely clones the BSR it receives.
19
20     Other passes are performed according to the options, such as
21     displaying views of the internal data or printing a summary of the score.
22
23     The activity log and warning/error messages go to standard error.

```

### 68.3.3 MusicXML generation

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/msdl > msdl -musicxml -about
2 What msdlConverter does:
3
4     This multi-pass converter basically performs 4 passes when generating MusicXML output
5     output:
6
7         Pass 1:  converts the MSDL input into a first MSR
8         Pass 2:  converts the first MSR into a second MSR;
9         Pass 3:  converts the second MSR into an MusicXML tree;
10        Pass 4:  converts the MusicXML tree to MusicXML code
11                and writes it to standard output.
12
13    Other passes are performed according to the options, such as
14    displaying views of the internal data or printing a summary of the score.
15
16    The activity log and warning/error messages go to standard error.

```

### 68.3.4 Guido generation

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/msdl > msdl -guido -about
2 What msdlConverter does:
3
4     This multi-pass converter basically performs 4 passes when generating Guido output
5     output:
6
7         Pass 1:  converts the MSDL input into a first MSR
8         Pass 2:  converts the first MSR into a second MSR;
9         Pass 3:  converts the second MSR into an MusicXML tree;
10        Pass 4:  converts the MusicXML tree to Guido code
11                and writes it to standard output.
12
13    Other passes are performed according to the options, such as
14    displaying views of the internal data or printing a summary of the score.
15
16    The activity log and warning/error messages go to standard error.
17 jacquesmenu@macmini: ~/musicformats-git-dev/msdl >

```

## 68.4 A first example

HelloWorld.msd is a minimal example:

```

1 %{
2     The unavoidable HelloWorld score
3     %}
4
5
6 % the language used for the pitches
7 % -----
8
9 pitches english           % default is english
10
11
12 % is there an anacrusis?
13 % -----
14
15 anacrusis                % measure numbers start at 0
16
17
18 % the structure

```



```

19 % -----
20
21 % score helloWorld = "Hello World in MSPL" {
22
23     music {
24         |1 c2.. d''8
25         |||                                % final bar
26     }
27
28 % } % helloWorld

```

## 68.5 First example output from the MSDL converter

Compiling HelloWorld.msd1 to LilyPond, we get the output below.

### 68.5.1 LilyPond output

```

1 \version "2.22.0"
2
3 % Comment or adapt next line as needed (default is 20)
4 #(set-global-staff-size 20 )
5
6 % Pick your choice from the next two lines as needed
7 %myBreak = { \break }
8 myBreak = {}
9
10 % Pick your choice from the next two lines as needed
11 %myPageBreak = { \pageBreak }
12 myPageBreak = {}
13
14 \header {
15     title                = ""
16     encodingDate         = "Sunday 2021-05-30 @ 12:11:50 CEST"
17     software              = "MSDL converter 1.0"
18 }
19
20 \paper {
21 }
22
23 \layout {
24     \context {
25         \Score
26         autoBeaming = ##f % to display tuplets brackets
27     }
28     \context {
29         \Voice
30     }
31 }
32
33 Part_Part_One_Staff_One_Voice_One = \absolute {
34     \language "nederlands"
35     c2.. d''8 }
36
37 \book {
38     \score {
39         <<
40
41         \new Staff = "Part_Part_One_Staff_One"
42         \with {
43             }
44         <<

```

```

45     \context Voice = "Part_Part_One_Staff_One_Voice_One" <<
46     \Part_Part_One_Staff_One_Voice_One
47     >>
48 >>
49
50 >>
51
52 \layout {
53   \context {
54     \Score
55     autoBeaming = ##f % to display tuplets brackets
56   }
57   \context {
58     \Voice
59   }
60 }
61
62 \midi {
63   \tempo 4 = 90
64 }
65 }
66
67 }

```

### 68.5.2 Braille output

With:

```
1 msdl -braille HelloWorld.msd1 -use-encoding-in-file-name -braille-output-kind utf8d
```

we get in file HelloWorld.msd1\_UTF8Debug.brf Braille 6-dots cells, which can be displayed in a suitable editor as:

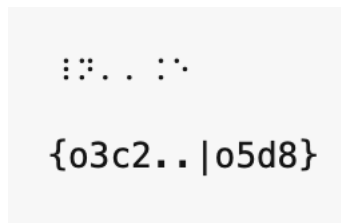


Figure 68.1: Braille for HelloWorld.xml with interpretation

The interpretation shows a textual view of the contents of the previous line. **o\*** indicates the octave number.

### 68.5.3 MusicXML output

Compiling HelloWorld.msd1 to MusicXML, we get:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
3   "http://www.musicxml.org/dtds/partwise.dtd">
4 <score-partwise version="3.1">
5   <!-- ===== Created by msdl 0.02 on Sunday 2021-05-30 @ 12:15:44 CEST from HelloWorld.
6   msdl ===== -->
7   <work>
8     <work-number/>
9     <work-title/>

```

```

9      </work>
10     <movement-number/>
11     <movement-title/>
12     <identification>
13         <encoding>
14             <software>msdl 0.02, https://github.com/jacques-menu/musicformats</software>
15             <encoding-date>2021-05-30</encoding-date>
16         </encoding>
17         <miscellaneous>
18             <miscellaneous-field name="description"/>
19         </miscellaneous>
20     </identification>
21     <part-std::list>
22         <score-part id="Part_One">
23             <part-name/>
24             <score-instrument id="Part_OneI1">
25                 <instrument-name/>
26             </score-instrument>
27         </score-part>
28     </part-std::list>
29     <part id="Part_One">
30         <measure number="1">
31             <attributes>
32                 <divisions>2</divisions>
33             </attributes>
34             <note>
35                 <pitch>
36                     <step>C</step>
37                     <octave>3</octave>
38                 </pitch>
39                 <duration>7</duration>
40                 <voice>1</voice>
41                 <type>half</type>
42                 <dot/>
43                 <dot/>
44                 <staff>1</staff>
45             </note>
46             <note>
47                 <pitch>
48                     <step>D</step>
49                     <octave>5</octave>
50                 </pitch>
51                 <duration>1</duration>
52                 <voice>1</voice>
53                 <type>eighth</type>
54                 <staff>1</staff>
55             </note>
56         </measure>
57     </part>
58 </score-partwise>

```

#### 68.5.4 Guido output

Compiling HelloWorld.msd1 to Guido, we get:

```

1  {[ \staff<1> \set<autoHideTiedAccidentals="on"> \title<"> \barFormat<style= "system",
   range="1"> \bar<hidden="true"> \beamsOff c0/2.. \beamsOff d2/8 ]
2  }

```

## 68.6 A more realistic example

Thanks to Jean Abou-Samra for providing UnPetitAir.msd1:

```

1  %{
2    An explicit and implicit voices piano score
3  %}
4
5
6  % l'identification
7  % -----
8
9  titre      "Un petit air"
10 compositeur "Jean Abou Samra"
11
12
13 % la langue pour les hauteurs de notes
14 % -----
15
16 hauteurs francais          % par défaut: english
17
18
19 % la partition
20 % -----
21
22 musique unPetitAir =
23 {
24   |1  clef treble
25       key c
26       time 9/8
27       r4. a,4-> <e g bf>8~ <e g bf>4.~
28
29   |2  <e g bf>4. r2.
30
31   % Maintenant, je reviens en arrière pour la voix supérieure.
32   |2  fs''16 gs'' fs''8 cs'' ds'' e'' b' d'' a' e'
33
34   % La voix inférieure s'éteint.
35   |3  c''8 gs' d' c' fs' a' b' gs' b
36   |4  a'8 e' a g as gs' d'( a ds)
37   |5  e8( b g d' a' e'' b'' c'' b''
38   |6  e''4.) % Rien à la fin.
39
40   % Je décide d'ajouter une tenue de la basse.
41   |5  e2.~ e4.
42
43   % J'ajoute encore une voix. Au passage, je change la métrique.
44   |6  time 6/8
45   |6  r8 e'( f') e' c'' d''
46
47   % Et encore un changement de métrique.
48   |7  time 4/4
49   |7  e''1~
50
51   % Je finis la phrase.
52   |7  e''4 e' d''8 c'' b' a'
53   |8  b'1
54
55   % Je retourne sur mes pas pour introduire l'ostinato.
56   |7  r8 e8 f e f e c' a
57   |8  r8 e8 f e c' a e f
58   |9  r8 ds e ds e ds b fs
59
60   % etc.
61 }
```

Jean also provided the output created by hand with LilyPond, see figure 68.2 [Un Petit Air, par Jean Abou-Samra], page 385:

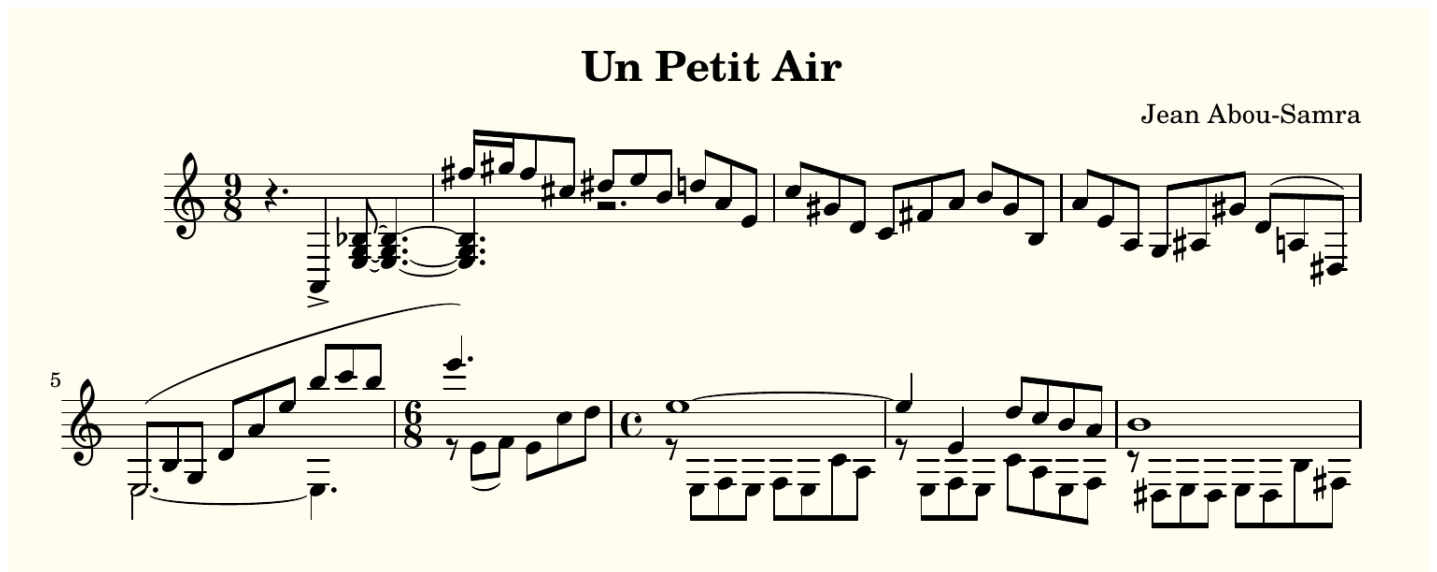


Figure 68.2: Un Petit Air, par Jean Abou-Samra

## 68.7 Multi-language support

### 68.7.1 Multi-language messages handling

### 68.7.2 Multi-language keywords handling

## 68.8 Lexical analysis

## 68.9 Music Scores Descriptions Representation (MSDR)

## 68.10 Syntax and semantic analysis

The language-dependent keywords leads to a recursive descent parser, since `flex`-generated scanners need 'fixed' keyword in the language description.

### 68.10.1 Error recovery

The MSDDL converter uses a variant of the *stopper sets* method that was present in the early Pascal and Pascal-S converters. The latter passed a set of tokens not to be overtaken to the procedures in charge of accepting the various statements in the language. Strangely enough, this was not done for declarations.

We use a stack of tokens sets that grows and shrinks in parallel with the accepting functions, to know more contextual informations when deciding whether to consume a token or not. The corresponding term is *it shift* when building the analysis tables in LR technology.

---

# Part XV

## Debugging

# Chapter 69

## Debugging

Debugging MusicFormats can be quite time-consuming. The trace options available have been designed to provide fine-grained tracing information to help locate issues.

Functions `catchSignals ()` at the beginning of the `main ()` function can be commented out in order to run a tool under a debugger.

File `src/wae/waeEnableAbortToDebugErrors.h` contains:

```
1  /*
2   MusicFormats Library
3   Copyright (C) Jacques Menu 2016-2022
4
5   This Source Code Form is subject to the terms of the Mozilla Public
6   License, v. 2.0. If a copy of the MPL was not distributed with this
7   file, You can obtain one at http://mozilla.org/MPL/2.0/.
8
9   https://github.com/jacques-menu/musicformats
10  */
11
12  #ifndef __waeEnableAbortToDebugErrors__
13  #define __waeEnableAbortToDebugErrors__
14
15
16  // comment the following definition if abort on internal errors is desired
17  // CAUTION: DON'T USE THIS IN PRODUCTION CODE,
18  // since that could kill a session on a \Web\ server, for example
19
20  #ifndef ABORT_TO_DEBUG_ERRORS
21  #define ABORT_TO_DEBUG_ERRORS
22  #endif
23
24
25  #endif
```

### 69.1 Useful options

Here are the most basing options used when debugging:

- option `-trace-passes`, `-tpasses` this is the first option to use, to locate in which pass the problem arises
- option `-input-line-numbers`, `-iln` this option produces the music elements input-line numbers in the output files
- the `-display*` options

## 69.2 Removing the results of a build

The contents of `distrib/` after a build is:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > ll build/
2 total 80
3  0 drwxr-xr-x  10 jacquesmenu  staff    320 May  2 18:18:50 2022 ./
4  0 drwxr-xr-x  39 jacquesmenu  staff   1248 May  2 18:18:51 2022 ../
5 16 -rw-r--r--@   1 jacquesmenu  staff   6148 May  2 15:45:10 2022 .DS_Store
6  8 -rw-r--r--@   1 jacquesmenu  staff   1815 Jun 29 09:10:50 2021 Building.md
7 32 -rw-r--r--   1 jacquesmenu  staff  14849 May  2 18:18:50 2022 CMakeLists.txt
8  8 -rw-r--r--   1 jacquesmenu  staff    291 Jun 29 09:10:50 2021 MakePkg.bat
9 16 -rw-r--r--   1 jacquesmenu  staff   7463 May  2 18:18:50 2022 Makefile
10 0 drwxr-xr-x@  27 jacquesmenu  staff    864 May  2 18:21:17 2022 bin/
11 0 drwxr-xr-x@   7 jacquesmenu  staff    224 May  2 18:21:11 2022 lib/
12 0 drwxr-xr-x  12 jacquesmenu  staff    384 May  2 18:19:25 2022 libdir/
```

The built files are in `distrib/bin`, `distrib/lib` and `distrib/libdir`. There is no `clean` target in `Makefile`. They can be removed in a single step with this alias:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > type rmbuild
2 rmbuild is aliased to 'cd ${MUSIC_FORMATS_DEV}/build ; rm -r bin lib libdir; ls -sal'
```

## 69.3 Reverting to a previous MusicFormats version

The GitHub MusicFormats repository keeps a number of recent releases, such as:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > git branch
2 * master
3   v0.9.60
4   v0.9.61
5   v0.9.62
```

Then `master` branch contains the development version. To switch back to another version, one should check it out:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > git checkout v0.9.61
2 Switched to branch 'v0.9.61'
3 Your branch is up to date with 'master/v0.9.61'.
4 jacquesmenu@macmini: ~/musicformats-git-dev > git branch
5   master
6   v0.9.60
7 * v0.9.61
8   v0.9.62
```

Now building this version with `make`, we get:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/build > xml2ly -version
2 Command line version of musicxml2lilypond converter v0.9.61 (March 3, 2022)
```



## Chapter 70

# Locating a bug with Git's bisection

A bug appeared in v0.9.63, in which the `-display-lpsr-short`, `-dlpsrshort` option causes `xml2ly` to crash. The symptom varies with the operating system, pointing to a probable memory corruption.

Reverting to MusicFormats previous versions show that:

- v0.9.60 behaves alright;
- v0.9.61 exhibits the bug:

```

1 jacquesmenu@macmini: ~/Desktop > lldb -- xml2ly fullbarrests/
   FullBarRestWithoutBarLine.xml -display-lpsr-short -aofn
2 (lldb) target create "xml2ly"
3 Current executable set to 'xml2ly' (x86_64).
4 (lldb) settings set -- target.run-args "fullbarrests/FullBarRestWithoutBarLine.xml"
   "-display-lpsr-short" "-aofn"
5 (lldb) r
6 Process 28676 launched: '/Users/jacquesmenu/musicformats-git-dev/build/bin/xml2ly' (
   x86_64)
7     The measure with ordinal number 3 is now registered with a duration of 1/1 in
   part Part_POne (partID "P1", partName "Soprano"),
   fPartMeasuresWholeNotesDurationsVector.size () = 2
8     The measure with ordinal number 4 is now registered with a duration of 1/1 in
   part Part_POne (partID "P1", partName "Soprano"),
   fPartMeasuresWholeNotesDurationsVector.size () = 2
9 %-----
10 Pass (ptional): displaying the LPSR as text, short version
11 %-----
12
13 ... ..
14
15 [PartGroup "PartGroup_1 ('0', fPartGroupName "Implicit")" (1 part), line 0
16   fPartGroupName
17     : "Implicit"
18 Process 28676 stopped
19 * thread #1, queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS (code=1,
   address=0x1)
20   frame #0: 0x00000001007b879a xml2ly`MusicFormats::msrPartGroup::print(this=0
   x0000600003e08100, os=0x000000010ca04650) const at msrPartGroups.cpp:1185:13
21   1182         i         = iBegin;
22   1183
23   1184     for ( ; ; ) {
24 -> 1185         os << (*i);
25   1186         if (++i == iEnd) break;
26   1187         os << std::endl;
27   1188     } // for
28 Target 0: (xml2ly) stopped.
29 (lldb)

```

This bug has thus been introduced between v0.9.60 and v0.9.61. There have been several `git push` occurrences leading from v0.9.60 to v0.9.61.

## 70.1 Locating a bug at random in the Git log

git provides various ways to display the commits history of the repository through `git log` options, for example:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > git log --pretty=format:"%h - %ad : %s"
2 ea338fd - Tue May 3 10:09:04 2022 +0200 : Complement to the Makefile
3 083db8a - Tue May 3 07:43:09 2022 +0200 : Before reverting to v0.9.60
4 12b6d93 - Mon May 2 09:41:37 2022 +0200 : Prior to bisecting
5 03d98be - Tue Apr 26 11:15:23 2022 +0200 : Finalized sone tracing options
6 3dd7b72 - Tue Apr 26 10:10:38 2022 +0200 : Finalized sone tracing options
7 7f7507c - Thu Apr 14 17:01:14 2022 +0200 : Finalized MFSL symtax and semantics, fixed a
   couple of issues
8 06109d3 - Thu Apr 14 17:00:50 2022 +0200 : Finalized MFSL symtax and semantics, fixed a
   couple of issues
9 62aa64c - Thu Apr 7 07:18:44 2022 +0200 : v0.9.62
10 671ffa4 - Thu Apr 7 06:26:34 2022 +0200 : Pre v0.9.62
11 bf9eb63 - Wed Apr 6 23:42:44 2022 +0200 : Pre v0.9.62
12 db4397c - Wed Apr 6 22:14:43 2022 +0200 : Pre v0.9.62
13 9a80b24 - Mon Apr 4 13:06:12 2022 +0200 : Added MFSL (MusicFormats Script Language)
14 2ef1150 - Mon Apr 4 12:07:07 2022 +0200 : Added MFSL (MusicFormats Script Language)
15 3f56d52 - Tue Mar 29 16:34:23 2022 +0200 : Added MFSL (MusicFormats Script Language)
16 fc1ea21 - Tue Mar 29 08:58:34 2022 +0200 : Added MFSL (MusicFormats Script Language)
17 737b996 - Mon Mar 28 23:42:03 2022 +0200 : Added MFSL (MusicFormats Script Language)
18 8c91155 - Sat Mar 26 08:35:55 2022 +0100 : Added MFSL (MusicFormats Script Language)
19 fc68a93 - Fri Mar 18 15:11:19 2022 +0100 : Added MFSL (MusicFormats Script Language)
20 01430a9 - Fri Mar 18 15:11:12 2022 +0100 : Added MFSL (MusicFormats Script Language)
21 4082813 - Thu Mar 17 18:50:02 2022 +0100 : Added MFSL (MusicFormats Script Language)
22 2696628 - Sun Mar 13 00:48:05 2022 +0100 : Added MFSL (MusicFormats Script Language)
23 a828231 - Thu Mar 10 14:28:11 2022 +0100 : Added MFSL (MusicFormats Script Language)
24 bf04937 - Wed Mar 9 12:53:17 2022 +0100 : Added MFSL (MusicFormats Script Language)
25 a855ee4 - Tue Mar 8 16:39:28 2022 +0100 : Added MFSL (MusicFormats Script Language)
26 b636816 - Mon Mar 7 14:49:54 2022 +0100 : Added MFSL (MusicFormats Script Language)
27 ecd5eaa - Sun Mar 6 00:11:05 2022 +0100 : Added 'keep-msr-voice, kmv' option
28 ec1c8ef - Sat Mar 5 08:48:39 2022 +0100 : Added 'ignore-msr-voice, imv' option
29 8246467 - Thu Mar 3 16:11:37 2022 +0100 : v0.9.61
30 603e19c - Thu Mar 3 13:43:48 2022 +0100 : Switched from C++11 to C++17 for <filesystem>
31 77d3d29 - Thu Mar 3 07:56:00 2022 +0100 : Switched from C++11 to C++17 for <filesystem>
32 a880063 - Thu Mar 3 07:44:08 2022 +0100 : Switched from C++11 to C++17 for <filesystem>
33 38b584f - Wed Mar 2 12:44:22 2022 +0100 : Switched from C++11 to C++17 for <filesystem>
34 662454a - Tue Mar 1 17:14:47 2022 +0100 : Pre-v0.9.61
35 2cb4d5f - Mon Feb 28 11:53:04 2022 +0100 : Renamed some documentation folders and files
36 c7839a8 - Mon Feb 28 09:56:46 2022 +0100 : Renamed some documentation folders and files
37 0e85f99 - Mon Feb 28 09:06:16 2022 +0100 : Renames some documentation folders and files
38 c5a43d9 - Fri Feb 25 17:48:29 2022 +0100 : Finalized files/musicxmlfiles/Makefile
39 21e3898 - Thu Feb 24 21:54:28 2022 +0100 : Added '-replicate-msr-measure' option
40 9738598 - Thu Feb 24 21:53:00 2022 +0100 : Added '-replicate-msr-measure' option
41 ae751c3 - Mon Feb 21 09:58:35 2022 +0100 : Added various options
42 f2d2f57 - Sat Feb 19 08:09:02 2022 +0100 : Workflow to publish Mac OS release
43 ac5ad6b - Sat Feb 19 08:00:57 2022 +0100 : Initializa npm package
44 29de34d - Fri Feb 18 11:00:42 2022 +0100 : v0.9.60
45 7c067d6 - Fri Feb 18 10:56:17 2022 +0100 : v0.9.60
46 5e3ba90 - Fri Feb 18 09:57:50 2022 +0100 : Pre v0.9.60
47 dfef7be - Fri Feb 18 09:56:07 2022 +0100 : Pre v0.9.60
48 c31dde3 - Wed Feb 16 11:50:42 2022 +0100 : Updates to the make and cmake configuration
49 fd6fef0 - Wed Feb 16 09:45:44 2022 +0100 : Complements to the installation doc
50 b7ad2af - Tue Feb 15 17:40:53 2022 +0100 : Distrib test 17
51 50a904c - Tue Feb 15 17:37:53 2022 +0100 : Distrib test 16
52 cf65bd3 - Tue Feb 15 08:41:14 2022 +0100 : Distrib test 15
53 9cda15e - Tue Feb 15 08:38:57 2022 +0100 : Distrib test 14
54 74a2b7f - Tue Feb 15 08:30:21 2022 +0100 : Distrib test 13
```

```

55 ee011e9 - Mon Feb 14 15:06:59 2022 +0100 : Distrib test 13
56 4d6f9cb - Mon Feb 14 08:54:23 2022 +0100 : Distrib test 12
57 7692b7b - Mon Feb 14 08:52:50 2022 +0100 : Distrib test 12
58 d9e943d - Sat Feb 12 09:30:45 2022 +0100 : Distrib test 11
59 3dde810 - Sat Feb 12 09:29:51 2022 +0100 : Distrib test 11
60 f07b02a - Fri Feb 11 17:55:34 2022 +0100 : Distrib test 10
61 5113824 - Fri Feb 11 16:47:17 2022 +0100 : Distrib test 9
62 7f0fa8e - Fri Feb 11 16:45:21 2022 +0100 : Distrib test 8
63 93f72f4 - Fri Feb 11 16:32:33 2022 +0100 : Distrib test 6
64 121aa64 - Fri Feb 11 14:35:38 2022 +0100 : Distrib test 5
65 547556f - Fri Feb 11 11:30:17 2022 +0100 : Creation of MusicFormats repository

```

One can pick one of the commits, revert to it and check whether the bug is present in it.

## 70.2 Locating a bug in the commits with Git's bisection

Locating the particular push that introduced the bug can be facilitated by git's *bisect* facility. Here is how it works:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect start
2
3 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect good v0.9.60
4
5 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect bad v0.9.61
6 Bisecting: 7 revisions left to test after this (roughly 3 steps)
7 [0e85f994ab00ea2dd94ddcb1895cbae5a32f072a] Renames some documentation folders and files
8
9 jacquesmenu@macmini: ~/musicformats-git-dev > git branch
10 * (no branch, bisect started on v0.9.61)
11 master
12 v0.9.60
13 v0.9.61
14 v0.9.62

```

The bisection proposes commit 0e85f994ab00ea2dd94ddcb1895cbae5a32f072a as a middle point between v0.9.60 and v0.9.61. So let us check it out:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git checkout 0
   e85f994ab00ea2dd94ddcb1895cbae5a32f072a
2 HEAD is now at 0e85f99 Renames some documentation folders and files
3
4 jacquesmenu@macmini: ~/musicformats-git-dev > git branch
5 * (no branch, bisect started on v0.9.61)
6 master
7 v0.9.60
8 v0.9.61
9 v0.9.62

```

Then, building this intermediate development version leads to:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/build > make
2 make macos
3 cd libdir && cmake .. -G Xcode -Wno-dev
4 -- VERSION: v0.9.61
5
6 -- Configuring version v0
7 .v9
8 .v61
9
10 ... ..
11

```

```

12 /Users/jacquesmenu/musicformats-git-dev/src/components/mfcLibraryComponent.cpp:12:10:
    fatal error:
13   ' ../../../../MusicFormatsVersionNumber.h' file not found
14 #include " ../../../../MusicFormatsVersionNumber.h"
15   ~~~~~
16 1 error generated.
17
18 ... ..
19
20 ** BUILD FAILED **

```

Well, this dev version had been pushed to have new files and/or contents saved on the MusicFormats repository ...and we should try other commits around it.

A first possibility is to use `git bisect skip`, that moves to :

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect skip
2 Bisecting: 7 revisions left to test after this (roughly 3 steps)
3 [c7839a87549660963a8b1ef0898d5cbcce8305aa] Renamed some documentation folders and files

```

Checking commit `c7839a87549660963a8b1ef0898d5cbcce8305aa` out, we get:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git checkout
   c7839a87549660963a8b1ef0898d5cbcce8305aa
2 HEAD is now at c7839a8 Renamed some documentation folders and files

```

Building that leads to the same error as above. Let us skip one again:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect skip
2 Bisecting: 7 revisions left to test after this (roughly 3 steps)
3 [a880063c134a7ba49b31f5fb52b47f682058f64a] Switched from C++11 to C++17 for <filesystem>

```

It turns out the commit `a880063c134a7ba49b31f5fb52b47f682058f64a` does not build either. Let us skip to the next commit:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect skip
2 Bisecting: 7 revisions left to test after this (roughly 3 steps)
3 [2cb4d5f2133d34a19fea8f47d9ed2ccfb24d0fad] Renamed some documentation folders and files
4
5 jacquesmenu@macmini: ~/musicformats-git-dev > git checkout 2
   cb4d5f2133d34a19fea8f47d9ed2ccfb24d0fad
6 HEAD is now at 2cb4d5f Renamed some documentation folders and files

```

Here the code base builds alright, and bug does not show up, so we should continue skipping.

## 70.3 Locating the bug in the code base

The bug we're after is found to have been introduced at this point:

- commit `2cb4d5f`, does not exhibit the bug;
- the next one, commit `662454a`, does.

The changes brought by commit can be shown with:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git log --patch -1 662454a > patch_662454a.
   txt
2
3 jacquesmenu@macmini: ~/musicformats-git-dev > ls -sal patch_662454a.txt
4 256 -rw-r--r--@ 1 jacquesmenu staff 80782 May  4 10:59 patch_662454a.txt

```

The bug shows up with `files/musicxmlfiles/fullbarrests/FullBarRestWithoutBarLine.xml`, but not with others such as `files/musicxmlfiles/multistaff/SATBExample.xml`.

Analysing the patch description in `patch_662454a.txt`, we find that nothing in the differences between those two successive patches can explain the crash. The problem thus lies elsewhere...

---

## Part XVI

## Indexes

# Files index

## Symbols

<a href="#">.cpp</a>	<a href="#">70</a>	<a href="#">lib/</a>	<a href="#">262</a>
<a href="#">.h</a>	<a href="#">49</a>	<a href="#">libmusicxml</a>	<a href="#">28</a>
<a href="#">&lt;std::regex&gt;</a>	<a href="#">25</a>	<a href="#">libmusicxml/build/bin</a>	<a href="#">28</a>
<a href="#">*.cpp</a>	<a href="#">46</a>	<a href="#">libmusicxml/src</a>	<a href="#">222</a>
<a href="#">*EnumTypes*</a>	<a href="#">92</a>	<a href="#">libmusicxml2Presentation/</a>	<a href="#">21</a>
<a href="#">*Interface.*</a>	<a href="#">40</a>	<a href="#">lilypond</a>	<a href="#">36, 61</a>
<a href="#">.github/workflows/</a>	<a href="#">257</a>	<a href="#">lilypondGeneration</a>	<a href="#">35</a>
<a href="#">/Applications/</a>	<a href="#">262</a>	<a href="#">lpsr</a>	<a href="#">35, 61</a>
<a href="#">1.0.0</a>	<a href="#">36</a>	<a href="#">lpsr2lilypond</a>	<a href="#">35</a>
<a href="#">CommonLaTeXFiles/</a>	<a href="#">20</a>	<a href="#">mfsl/</a>	<a href="#">25</a>
<a href="#">Downloads/</a>	<a href="#">260</a>	<a href="#">mfutilities</a>	<a href="#">34</a>
<a href="#">Interface</a>	<a href="#">37</a>	<a href="#">msdl</a>	<a href="#">35, 61</a>
<a href="#">IntroductionToMusicxml/</a>	<a href="#">21</a>	<a href="#">msdl2braille</a>	<a href="#">33</a>
<a href="#">LaTeXCommonSettings.tex</a>	<a href="#">31</a>	<a href="#">msdl2guido</a>	<a href="#">33</a>
<a href="#">LilyPondIssue34</a>	<a href="#">33, 245</a>	<a href="#">msdl2lilypond</a>	<a href="#">33</a>
<a href="#">Makefile</a>	<a href="#">26, 28, 36</a>	<a href="#">msdl2musicxml</a>	<a href="#">33</a>
<a href="#">Makefile_ORIGINAL</a>	<a href="#">29</a>	<a href="#">msdlconverter</a>	<a href="#">33</a>
<a href="#">Mikrokosmos3Wandering</a>	<a href="#">33, 121</a>	<a href="#">msdr</a>	<a href="#">35</a>
<a href="#">MusicFormatsVersionDate.txt</a>	<a href="#">21, 265</a>	<a href="#">msr</a>	<a href="#">35, 61</a>
<a href="#">MusicFormatsVersionNumber.txt</a>	<a href="#">21, 22, 265</a>	<a href="#">msr2braille</a>	<a href="#">33</a>
<a href="#">oah</a>	<a href="#">61</a>	<a href="#">msr2bsr</a>	<a href="#">35</a>
<a href="#">bin/</a>	<a href="#">262</a>	<a href="#">msr2guido</a>	<a href="#">33</a>
<a href="#">braille</a>	<a href="#">61</a>	<a href="#">msr2lilypond</a>	<a href="#">33</a>
<a href="#">brailleGeneration</a>	<a href="#">35</a>	<a href="#">msr2lpsr</a>	<a href="#">35</a>
<a href="#">bsr</a>	<a href="#">35, 61</a>	<a href="#">msr2msr</a>	<a href="#">35</a>
<a href="#">bsr2braille</a>	<a href="#">35</a>	<a href="#">msr2musicxml</a>	<a href="#">33</a>
<a href="#">bsr2bsr</a>	<a href="#">35</a>	<a href="#">msr2mxsr</a>	<a href="#">35</a>
<a href="#">build</a>	<a href="#">28</a>	<a href="#">msr2mxsrOah</a>	<a href="#">153</a>
<a href="#">build-ubuntu-version.yml</a>	<a href="#">257</a>	<a href="#">msrGeneration</a>	<a href="#">35</a>
<a href="#">build/bin</a>	<a href="#">26</a>	<a href="#">msrapi</a>	<a href="#">35</a>
<a href="#">clean</a>	<a href="#">36</a>	<a href="#">multiGeneration</a>	<a href="#">35</a>
<a href="#">clisamples</a>	<a href="#">32</a>	<a href="#">musicformats-*-distrib/</a>	<a href="#">262</a>
<a href="#">common</a>	<a href="#">31</a>	<a href="#">musicformats-windows-distrib/</a>	<a href="#">262</a>
<a href="#">components</a>	<a href="#">33</a>	<a href="#">musicformatsversion.txt</a>	<a href="#">36</a>
<a href="#">converters</a>	<a href="#">33</a>	<a href="#">musicxml2braille</a>	<a href="#">33, 116</a>
<a href="#">countnotes</a>	<a href="#">28</a>	<a href="#">musicxml2guido</a>	<a href="#">33</a>
<a href="#">doc</a>	<a href="#">28</a>	<a href="#">musicxml2lilypond</a>	<a href="#">33</a>
<a href="#">documentation/</a>	<a href="#">20</a>	<a href="#">musicxml2musicxml</a>	<a href="#">33</a>
<a href="#">elements.bash</a>	<a href="#">32</a>	<a href="#">mxsr</a>	<a href="#">35, 61, 242</a>
<a href="#">elements.h</a>	<a href="#">30</a>	<a href="#">mxsr2guido</a>	<a href="#">35</a>
<a href="#">elements.txt</a>	<a href="#">30</a>	<a href="#">mxsr2msr</a>	<a href="#">35</a>
<a href="#">files</a>	<a href="#">28, 36</a>	<a href="#">mxsr2msrOah</a>	<a href="#">153</a>
<a href="#">formats</a>	<a href="#">35</a>	<a href="#">mxsr2musicxml</a>	<a href="#">35</a>
<a href="#">formatsgeneration</a>	<a href="#">35</a>	<a href="#">mxsrGeneration</a>	<a href="#">35</a>
<a href="#">generators</a>	<a href="#">33</a>	<a href="#">oah</a>	<a href="#">34</a>
<a href="#">graphics/</a>	<a href="#">21</a>	<a href="#">packages</a>	<a href="#">28</a>
<a href="#">guidoGeneration</a>	<a href="#">35</a>	<a href="#">passes</a>	<a href="#">33, 35</a>
<a href="#">javascript</a>	<a href="#">28</a>	<a href="#">patch_662454a.txt</a>	<a href="#">404</a>
		<a href="#">presentation</a>	<a href="#">32</a>

[samples](#) .....[28](#)  
[schemas](#) .....[28](#)  
[scripts](#) .....[32](#)  
[src](#) .....[28](#), [32](#)  
[std::ostream](#) .....[236](#)  
[validation](#) .....[28](#)  
[wae](#) .....[35](#)  
[win32](#) .....[28](#)  
[xml2brl](#) .....[61](#), [130](#)  
[xml2guido](#) .....[28](#)  
[xml2lbr](#) .....[130](#)  
[xml2ly](#) .....[48](#), [61](#), [130](#)  
[xml2xml](#) .....[130](#)  
[musicxml2ly](#) .....[245](#)

**B**

[build](#)  
     [CMakeLists.txt](#) .....[64](#)  
     [CMakesList.txt](#) .....[26](#)  
     [lib](#) .....[22](#)  
     [Makefile](#) .....[26](#)

**C**

[clisamples](#) .....[140](#)  
     [displayMusicformatsHistory.cpp](#) .....[120](#)  
     [displayMusicformatsVersion.cpp](#) .....[120](#)  
     [libMultipleInitsTest.cpp](#) .....[140](#)  
     [LilyPondIssue34.cpp](#) .....[42](#), [140](#)  
     [Makefile](#) .....[26](#)  
     [mfsl.cpp](#) .....[375](#)  
     [Mikrokosmos3Wandering.cpp](#) [17](#), [41](#), [140](#), [144](#)  
     [msdl.cpp](#) .....[140](#), [144](#)  
     [xml2Any.cpp](#) .....[140](#)  
     [xml2ly.cpp](#) .....[44](#)  
[tt \\*Component.h/.cpp](#) .....[115](#)

**D**

[distrib](#)  
     .....[263](#), [399](#)  
     [bin](#) .....[399](#)  
     [lib](#) .....[399](#)  
     [libdir](#) .....[399](#)  
[documentation](#)  
     [CommonLaTeXFiles](#) .....[22](#)  
     [CommonLaTeXFiles/MSRClassesHierarchyPicture.tex](#)  
         [21](#)  
     [CommonLaTeXFiles/MSRScoreRepresentation.tex](#)  
         [21](#)  
     [MusicFormatsAPIGuide/MusicFormatsAPIGuide.pdf](#)  
         [24](#)  
     [MusicFormatsMaintenanceGuide/MusicFormatsMaintenanceGuide.pdf](#)  
         [24](#)  
     [MusicFormatsUserGuide/MusicFormatsUserGuide.pdf](#)  
         [24](#)

**F**

[files](#)  
     [musicxmlfiles/fullbarrests/FullBarRestWithoutBarLine.pdf](#)  
         [404](#)  
     [musicxmlfiles/multistaff/SATBExample.xml](#)  
         [404](#)

**I**

[include/](#) .....[18](#)  
[InsiderHandler.cpp](#) .....[49](#)

**L**

[libmusicxml/samples](#)  
     .....[74](#)  
     [countnotes.cpp](#) .....[100](#)  
     [elements/factory.cpp](#) .....[225](#)  
[libmusicxml/src/files](#)  
     [xmlfile.h](#) .....[226](#)  
     [xmlreader.h/.cpp](#) .....[227](#)

**M**

[main \(\)](#) .....[32](#), [46](#), [138](#), [252](#)

**S**

[scripts](#)  
     [MakeMusicFormatsDistributions.bash](#) ...[262](#),  
         [263](#)  
     [ShowMusicFormatsVersion.bash](#) .....[265](#)  
     [ZshDefinitionsForMusicFormats.zsh](#) .....[18](#)  
[src](#)  
     .....[64](#)  
     [clisamples](#) .....[46](#)  
     [formats](#) .....[39](#)  
     [MusicFormatsVersionDate.h](#) .....[21](#)  
     [MusicFormatsVersionNumber.h](#) .....[21](#)  
     [passes](#) .....[40](#)  
     [representations](#) .....[39](#)  
[src/](#) .....[18](#)  
[src/clisamples](#) .....[140](#)  
     [displayMusicformatsHistory.cpp](#) .....[120](#)  
     [displayMusicformatsVersion.cpp](#) .....[120](#)  
     [libMultipleInitsTest.cpp](#) .....[140](#)  
     [LilyPondIssue34.cpp](#) .....[42](#), [140](#)  
     [Makefile](#) .....[26](#)  
     [mfsl.cpp](#) .....[375](#)  
     [Mikrokosmos3Wandering.cpp](#) [17](#), [41](#), [140](#), [144](#)  
     [msdl.cpp](#) .....[140](#), [144](#)  
     [xml2Any.cpp](#) .....[140](#)  
     [xml2ly.cpp](#) .....[44](#)  
[src/components](#)  
     .....[110](#)  
     [mfcComponents.h](#) .....[38](#)  
     [mfIndentedTextOutput.h](#) .....[54](#)  
[src/formats/bsr](#)  
     [bsr.cpp](#) .....[76](#)  
     [bsrEnumTypes.h](#) .....[219](#)  
     [bsrEnumTypes.h/.cpp](#) .....[219](#)  
[src/formats/lpsr](#)  
     [lpsr/lpsrPapers.h](#) .....[80](#)  
     [lpsr2lilypondTranslator.cpp](#) .....[134](#)  
     [lpsrElements.h/.cpp](#) .....[61](#)  
     [lpsrEnumTypes.h/.cpp](#) .....[217](#)  
     [lpsrOah.cpp](#) .....[160](#)  
     [lpsrScores.cpp](#) .....[124](#)  
     [lpsrScores.h/.cpp](#) .....[44](#)  
[src/formats/msdl](#)  
     [msdlEnumTypes.h/.cpp](#) .....[389](#)  
     [msdlScanner.cpp](#) .....[169](#)



msdlScanner.h .....	168	mfslDriver.h .....	378
msdlTokens.cpp .....	82	mfslInterpreterInterface.h .....	385
src/formats/msr .....	198	mfslParser.cpp .....	375, 377
mfslNotes.h/.cpp .....	86	mfslParser.h .....	375
msr.h/.cpp .....	273	mfslParser.output .....	375
msr2msrTranslator.h .....	169	mfslParser.yy .....	375--377
msrBarLines.h .....	63, 317	mfslScanner.cpp .....	375
msrBarlines.h .....	63	mfslScanner.ll .....	375--377, 380, 385
msrBasicTypes.cpp .....	65, 82	src/interpretersmfslinterpreter//mfsl	
msrBasicTypes.h ...	81, 157, 182, 185, 186, 198, 206	mfslDriver.h/.cpp .....	86
msrBasicTypes.h/.cpp .....	78, 87, 157, 175, 207, 273	src/mflibrary	
msrBeatRepeats.h/.cpp .....	200	mfMusicformatsErrors.h .....	139
msrChords.h/.cpp .....	293	mfOnOff.cpp .....	80
msrDurations.h .....	177	mfOnOff.h .....	80
msrElements.h .....	99	mfServiceRunData.h/.cpp .....	43, 272
msrFiguredBasses.h/.cpp .....	356	src/mfutilities .....	61
msrHistory.h/.cpp .....	115	mfcBasicTypes.h .....	155
msrIdentification.h .....	176	mfEnumAll.h .....	66, 81, 82
msrKeys.h/.cpp .....	85	mfIndentedTextOutput.h .....	52, 53
msrLigatures.h .....	180	mfStringsHandling.cpp .....	70
msrMeasureElement.h/.cpp .....	178	src/oah	
msrMeasureRepeat.h/.cpp .....	202	tracingOah.h/.cpp .....	61
msrMeasureRepeats.h/.cpp .....	89	enableHarmoniesExtraOahIfDesired.h .....	160
msrMeasures.h/.cpp .....	283, 293	generalOah.h/.cpp .....	161
msrMeasuresSlice.h .....	179	harmoniesExtraOah.h/.cpp .....	160, 161
msrMeasuresSlices.h/.cpp .....	211	mfIndentedTextOutput.cpp .....	70
msrMeausre.h/.cpp .....	198	musicxmlOah.h/.cpp .....	61
msrMultipleFullBarRests.h/.cpp .....	203	oahAtomCollection.h/.cpp .....	155
msrNotes.cpp .....	104, 349, 365	oahAtomsCollection .....	130
msrOah.cpp .....	129	oahAtomsCollection.cpp .....	133, 148, 158
msrOah.h .....	129	oahAtomsCollection.h .....	157
msrPartGroups.h .....	93	oahAtomsCollection.h.h/.cpp .....	131
msrPartGroupsElements.h .....	92	oahAtomsCollection.h/.cpp ..	123, 141, 143, 152, 277
msrParts.cpp .....	71	oahBasicTypes.cpp .....	140, 146
msrParts.h .....	70, 93, 341, 358	oahBasicTypes.h .....	140
msrParts.h/.cpp .....	293--295, 326	oahBasicTypes.h/.cpp ..	132, 136, 139, 140, 146
msrPathToVoice.h.h/.cpp .....	216	oahEarlyOptions.cpp .....	70, 149
msrRepeats.h/.cpp .....	203	oahEarlyOptions.h .....	148
msrSegment.cpp .....	85	oahElement.cpp .....	151
msrSegments.cpp .....	312	oahElements.cpp .....	152, 154
msrSegments.h/.cpp .....	199, 285, 293	oahElements.h .....	146
msrSpanners.h .....	181	oahElements.h/.cpp .....	140, 150
msrStaves.h/.cpp .....	293--295	oahEnableTracingIfDesired.h .....	48
msrTempos.h .....	186	oahOah.cpp .....	156
msrTempos.h/.cpp .....	84, 177	oahOah.h/.cpp .....	153
msrTies.h .....	85	tracingOah.h/.cpp .....	48
msrTimeSignature.cpp .....	168	src/utilities	
msrTimeSignature.h .....	167	mfBool.h/.cpp .....	80
msrTuplets.h/.cpp .....	293	mfcLibraryComponent.h/.cpp .....	118
msrVoiceElements.h/.cpp .....	196	mfIndentedTextOutput.cpp.h/.cpp .....	171
msrVoices.cpp .....	309--311	mfIndentedTextOutput.h/.cpp .....	51, 61
msrVoices.h .....	90, 165, 174, 175, 198	mfTiming.h/.cpp .....	58
msrVoices.h/.cpp .....	167, 287, 293--295	src/wae	
mxsr2msrTranslator.cpp .....	347, 364	waeEnableAbortToDebugErrors.h ..	47, 54, 398
src/formats/mxsr		waeExceptions .....	47
mxsr.h/.cpp .....	222, 227, 241	waeExceptions.h/.cpp .....	72
mxsrOah.h/.cpp .....	153	waeHandlers.cpp .....	69
src/interpreters/mfslinterpreter/mfsl		waeHandlers.h .....	68
location.hh .....	375	waeInterface.h.h/.cpp .....	72
mfslDriver.cpp .....	379		

# Types index

## Symbols

.....	181
,	224
:	225
Bool	80
Enum*	83
EnumTrueHarmonies	83
Mikrokosmos3WanderingInsiderHandler	121
SMARTP	84
SXMLFile	227
S_FiguredBass	360
S_figured_bass	361, 362
S_harmony	342, 344, 345
S_msrBarLine	165
S_msrFiguredBass	365, 368
S_msrHarmony	354
S_msrPartGroup	92, 93
S_msrVoice	166
S_part_name	31
TXMLFile	227
.....	157, 224, 273
_msrMeasure	203
bsr2brailleTranslator	234
bsrBrailleGenerator	235
bsrCellKind	219, 220
c	87
char*	378
cubase	130
lpsr2lilypondTranslator	102, 103, 232
lpsrComment	124
lpsrPitchesLanguageAtom	130
lpsrScore	44, 124, 217, 233
mfEnumAll	66
mfMultiGenerationOutputKindAtom	138
mfOnOffKind	80
mfOptionsAndArguments	144
mfServiceRunData	42
mfTimingItemsList	58
mfcComponenKind	112
mfcComponent	112
mfcConverterComponent	113
mfcGeneratorComponent	113
mfcLibraryComponent	113
mfcMultiComponent	113, 114, 120, 155
mfcMultiComponentEntropyKind	114
mfcMultiComponentUsedFromTheCLIKind	114
mfcPassComponent	113
mfcRepresentationComponent	113
mfcVersionDescr	111
mfcVersionNumber	110
mfcVersionsHistory	111
mfsldriver	378
msr*Element	192
msr2mxsrTranslator	106, 240
msrAccordionRegistration	194
msrBarCheck	192
msrBarLine	192, 317, 319
msrBarNumberCheck	192
msrBeatRepeat	200
msrBeatRepeatPattern	201
msrBeatRepeatReplicas	201
msrBook	195, 216
msrChord	95, 164, 192, 198, 340, 357
msrChordInKind	78
msrChords	192
msrClef	190, 194
msrClefKind	190
msrCoda	194
msrDalSegno	194
msrDamp	194
msrDampAll	194
msrDottedDuration	186
msrDoubleTremolo	95
msrDurationKind	177, 185, 186
msrElement	192
msrEyeGlasses	194
msrFiguredBass	208, 356, 359, 362--364, 369
msrGraceNotesGroup	131, 192
msrHarmony	208, 344--349, 354
msrHarmonyKind	81
msrHarpPedalsTuning	194
msrHiddenMeasureAndBarLine	192
msrHumdrumScotKeyItem	190
msrIdentification	176
msrKey	85, 190, 194
msrLength	157, 158
msrLigature	181
msrLigatureKind	180
msrLineBreak	194
msrMeasure	95, 102, 174, 289, 290
msrMeasureElement	95, 177, 178
msrMeasureElements	198, 283
msrMeasureRepeat	202
msrMeasureRepeatPattern	202
msrMeasureRepeatReplicas	202
msrMeasuresSlice	179, 212
msrMeasuresSlicesSequence	179, 212
msrMeausre	198

## Types index

msrMoment	178	msrVoiceElement	192, 196--198, 204, 282
msrMultipleFullBarRests	102, 203	msrVoiceKind	198
msrNote	95, 108, 165, 177, 180, 205, 206, 208, 210, 340, 343, 346, 357, 360, 363	msrVoiceStaffChange	194
msrNoteEvent	180	msrmeasure	199
msrNoteEventKind	180, 211	mxmlelement	222
msrNoteKind	206	mxsr2msrSkeletonBuilder	31, 40
msrOctaveEntryVariable	280, 281	mxsr2msrTranslator	230
msrOctaveShift	194	oah*Element	130
msrPageBreak	194	oahAtom	129--131
msrPart	92, 93, 192, 209, 295, 340, 341, 357, 358	oahAtomExpectingAValue	132
msrPartGroup	91, 93, 192, 195	oahAtomImplicitlyStoringAValue	137
msrPartGroupElement	91--93, 192	oahAtomStoringAValue	130, 132, 133, 135
msrPathToVoice	216	oahBooleanAtom	130, 133, 153
msrPedal	194	oahColorRGBAtom	133
msrPrintLayout	194	oahCombinedBooleansAtom	130
msrRehearsalMark	194	oahContactAtom	130
msrRepeat	198, 199, 203	oahElement	129, 131
msrRepeatCommonPart	203, 204	oahElementHelpOnlyKind	146
msrRepeatEnding	203, 204	oahElementUse	138
msrRepeatEndingKind	204	oahFindStringMatch	150
msrScaling	100	oahFindableElement	130, 131, 150
msrScordatura	194	oahFloatAtom	130
msrScore	102, 126, 195, 216, 329	oahGroup	129, 130
msrSegment	192, 198, 282, 309	oahHandler	120, 129, 130
msrSegno	194	oahHistoryAtom	155
msrSlur	174	oahIntegerAtom	130
msrSpannerTypeKind	180	oahLengthAtom	130, 157
msrStaff	192, 341, 358	oahLengthUnitKindAtom	133
msrStaffDetails	194	oahMacroAtom	277
msrStaffLevelElement	194	oahMultiplexBooleansAtom	130
msrStanza	210	oahOahGroup	153
msrSyllable	192, 194, 210	oahOptionsVector	132, 137
msrSyllableKind	210	oahPrefix	130
msrTempo	84, 177, 189, 194	oahPureHelpAtomExpectingAValue	136
msrTempoKBeatUnitsKind	186	oahPureHelpAtomWithoutAValue	136
msrTempoNote	187	oahRationalAtom	130
msrTempoTuplet	187	oahSubGroup	129, 130
msrTime	194	oahThreeBooleansAtom	130
msrTimeSignature	70, 191, 192	oahVersionAtom	123, 155
msrTimeSignatureSymbolKind	191	s	65, 69, 75, 77, 78, 84, 317
msrTransposition	194	v	80
msrTuplet	95, 107, 131, 340, 357	xml2lyInsiderHandler	274
msrTupletElement	95, 192, 194	xmlattribute	223
msrTupletFactor	207	xmlelement	223, 224
msrTupletInKind	207	yy::location	375
msrVoice	167, 197, 210, 282, 295, 341, 358		
msrVoiceCreateInitialLastSegmentKind	198	<b>S</b>	
		SXMLFile	227

# Methods and fields index

## Symbols

<code>*DeepClone ()</code> .....	167
<code>*IsNeeded</code> .....	233
<code>*NewbornClone ()</code> .....	166
<code>fBooleanVariable</code> .....	133
<code>fContainingFindableElementInfo</code> .....	151
<code>fInsiderHandler</code> .....	143
<code>fPartGroupElementsList</code> .....	93
<code>fReverseNamesDisplayOrder</code> .....	153
<code>fSetByAnOption</code> .....	133
<code>fVersionsList</code> .....	111
<code>msr2msrTranslator</code> .....	350, 366
<code>mxsr2msrTranslator</code> .....	345, 346, 362, 363
<code>RegularHandler</code>	
<code>createOahRegularGroup ()</code> .....	154
<code>acceptIn ()</code> .....	69
<code>acceptOut ()</code> .....	69
<code>applyAtomWithDefaultValue ()</code> .....	139
<code>applyAtomWithValue ()</code> .....	139
<code>applyElement ()</code> .....	139
<code>asString() ()</code> .....	88
<code>asStringShort() ()</code> .....	88
<code>asString ()</code> .....	54, 55, 69, 181, 182
<code>browseData ()</code> .....	69, 102
<code>browsetData ()</code> .....	98, 102
<code>checkGroupOptionsConsistency ()</code> .....	152
<code>compare* ()</code> .....	90
<code>countnotes</code>	
<code>visitStart</code> .....	100
<code>create* ()</code> .....	69, 71, 84
<code>createGlobalMxsr2msrOahGroup ()</code> .....	49
<code>finalize*() ()</code> .....	87
<code>finalize* ()</code> .....	293
<code>generateCodeForBrailleCell ()</code> .....	236
<code>get*() ()</code> .....	85
<code>get* ()</code> .....	69
<code>getSetByAnOption ()</code> .....	134
<code>getValue ()</code> .....	375
<code>initialize*() ()</code> .....	86
<code>initializeHandlerMultiComponent ()</code> ..	120, 121
<code>lpsr2lilypondTranslator</code>	
<code>generateLilypondVersion ()</code> .....	134
<code>lpsrOahGroup</code>	
<code>initializeLpsrPaperOptions ()</code> .....	160
<code>lpsrScore</code>	
<code>lpsrScore ()</code> .....	124
<code>mfIndentedStreamBuf</code>	
<code>sync ()</code> .....	51
<code>mfMultiGenerationOutputKindAtom</code>	
<code>mfMultiGenerationOutputKindAtom ()</code> .....	138
<code>mfcComponent</code>	
<code>printOwnHistory ()</code> .....	113
<code>mfcMultiComponent</code>	
<code>printHistory ()</code> .....	114
<code>print ()</code> .....	114
<code>mfsldriver</code>	
<code>parseInput_Pass1 ()</code> .....	379
<code>msr2bsrTranslator</code>	
<code>finalizeCurrentMeasureClone ()</code> .....	293
<code>msr2lpsrTranslator</code>	
<code>visitStart (S_msrMeasure&amp; elt)</code> .....	287
<code>msr2msrTranslator</code>	
<code>visitStart (S_msrMeasure&amp; elt)</code> .....	287
<code>msr2mxsrTranslator</code>	
<code>translateMsrToMxsr ()</code> .....	240
<code>msrChord</code>	
<code>finalizeChord ()</code> .....	293
<code>msrClef</code>	
<code>getClefStaffNumber ()</code> .....	240
<code>msrHarmony</code>	
<code>create ()</code> .....	345
<code>msrMeasureElement</code>	
<code>fMeasureElementMeasurePosition</code> .....	178
<code>msrMeasureRepeat</code>	
<code>displayMeasureRepeat ()</code> .....	89
<code>msrMeasuresSlicesSequence</code>	
<code>identifySoloNotesAndRests ()</code> .....	215
<code>msrMeasure</code>	
<code>appendElementToMeasure ()</code> .....	283
<code>finalizeFiguredBassMeasure ()</code> .....	293
<code>finalizeHarmonyMeasure ()</code> .....	293
<code>finalizeMeasureClone ()</code> .....	293
<code>finalizeMeasure ()</code> .....	292, 293, 299
<code>finalizeRegularMeasure ()</code> .....	293
<code>msrMultipleFullBarRests</code>	
<code>appendMeasureCloneToMultipleFullBarRests ()</code>	287
<code>msrNote</code>	
<code>appendHarmonyToNote ()</code> .....	347, 348
<code>msrPart</code>	
<code>appendStaffDetailsToPart ()</code> .....	108
<code>browseData ()</code> .....	326
<code>collectPartMeasuresSlices ()</code> .....	213
<code>createAMeasureAndAppendItToPart ()</code> .....	290
<code>createPartFiguredBassVoice ()</code> .....	359
<code>finalizeLastAppendedMeasureInPart ()</code> ..	295
<code>finalizePartAndAllItsMeasures ()</code> .....	295
<code>finalizePartClone ()</code> .....	293

- finalizePart () ..... 293, 296
- finalizeRepeatEndInPart () ..... 294
- msrSegment
  - appendMeasureToSegment () ..... 285, 287
  - assertSegmentElementsListIsNotEmpty () 312
  - createAMeasureAndAppendItToSegment () 199, 287
  - createSegmentDeepClone () ..... 309
  - createSegmentNewbornClone () ..... 309
  - create () ..... 309
  - finalizeAllTheMeasuresOfSegment () .... 293
- msrStaff
  - collectStaffMeasuresSlices () .... 214, 215
  - finalizeLastAppendedMeasureInStaff () . 295
  - finalizeRepeatEndInStaff () ..... 294
  - finalizeStaff () ..... 293, 296, 297
- msrTempo
  - createTempoPerMinute () ..... 76
  - fTempoBeatUnit ..... 189
  - fTempoEquivalentBeatUnit ..... 189
  - fTempoKind ..... 189
  - fTempoNotesRelationshipKind ..... 189
  - fTempoNotesRelationshipLeftElements ... 189
  - fTempoNotesRelationshipRightElements .. 189
  - fTempoParenthesizedKind ..... 189
  - fTempoPerMinute ..... 189
  - fTempoPlacementKind ..... 189
  - fTempoWordsList ..... 189
  - kTempoNotesRelationshipEquals ..... 189
- msrTemp
  - msrTempoBeatUnitsKindAsString () ..... 76
- msrTie
  - create () ..... 76
- msrTuplets
  - finalizeTuplet () ..... 293
- msrVoice
  - addGraceNotesGroupBeforeAheadOfVoiceIfNeeded () 311
  - appendBarLineToVoice () ..... 311
  - appendCodaToVoice () ..... 311
  - appendDampAllToVoice () ..... 311
  - appendDampToVoice () ..... 311
  - appendEyeGlassesToVoice () ..... 311
  - appendMeasureCloneToVoiceClone () .... 287
  - appendMultipleFullBarRestsToVoice () .. 287
  - appendNoteToVoice () ..... 349, 365
  - appendPedalToVoice () ..... 311
  - appendPendingMeasureRepeatToVoice () .. 287
  - appendSegnoToVoice () ..... 311
  - appendStaffDetailsToVoice () ..... 311
  - createAMeasureAndAppendItToVoice () .. 287, 311
  - createMeasureRepeatFromItsFirstMeasures () 287, 310, 315
  - createNewLastSegmentForVoice () ..... 310
  - createNewLastSegmentFromItsFirstMeasureForVoice ( 287, 311
  - fVoiceLastSegment ..... 198
  - finalizeLastAppendedMeasureInVoice () . 295
  - finalizeRepeatEndInVoice () ..... 294
  - finalizeVoiceAndAllItsMeasures () ..... 295
  - finalizeVoice () ..... 293, 297
  - handleHooklessRepeatEndingEndInVoice () 311
  - handleMultipleFullBarRestsStartInVoiceClone () 311
  - handleVoiceLevelContainingRepeatEndWithoutStart () 311
  - handleVoiceLevelRepeatEndWithStart () . 311
  - handleVoiceLevelRepeatEndWithoutStart () 311
  - handleVoiceLevelRepeatEndingStartWithExplicitStart 287, 311
  - handleVoiceLevelRepeatEndingStartWithoutExplicitSta 287, 311
  - handleVoiceLevelRepeatStart () ... 287, 311
  - initializeVoice () ..... 309
- mxsr2msrSkeletonBuilder
  - fThereAreHarmoniesToBeAttachedToCurrentNote 342
- mxsr2msrTranslator
  - finalizeCurrentChord () ..... 293
  - finalizeTupletAndPopItFromTupletsStack () 293
  - handlePendingFiguredBasses () ..... 364
  - populateNote () ..... 346, 363
  - visitStart ( S\_bass& elt) ..... 31
  - visitStart ( S\_notations& elt) ..... 31
  - visitStart ( S\_other\_notation& elt) .... 31
  - visitStart ( S\_part\_name& elt) ..... 31
- oahAtomStoringAValue
  - fSetByAnOption ..... 133
- oahEarlyOptions
  - applyEarlyOptionIfRelevant () ..... 149
- oahElement
  - fetchNames () ..... 154
- oahHandler
  - handleOptionNameCommon () ..... 138
  - applyOptionsFromElementUsesList () .... 146
  - checkMissingPendingArgvAtomExpectingAValueValue () 140
  - fHandlerArgumentsVector ..... 139
  - fHandlerMultiComponent ..... 120, 121
  - fPendingArgvAtomExpectingAValue ..... 139
  - findStringInFindableElement () ..... 151
  - handleKnownArgvAtom () ..... 139
  - handleOptionNameCommon () ..... 138
  - handleOptionsAndArgumentsFromArgcArgv () 138
- oahHistoryAtom
  - printHistory () ..... 156
- oahIntegerAtom
  - setIntegerVariable () ..... 133
- oahLengthAtom
  - applyAtomWithValue () ..... 158
  - printAtomWithVariableOptionsValues () . 159
- oahMacroAtom
  - applyElement () ..... 278
  - fMacroAtomsList ..... 278
- oahOahGroup
  - initializeOahBasicHelpOptions () . 153, 156
  - printOahOahValues () ..... 153

oahPrefix		
findStringInFindableElement ()	.....	152
oahVersionAtom		
applyElement ()	.....	123, 148
printVersion ()	.....	123
printHistory ()	.....	113, 124, 155, 156
printSlices ()	.....	108
printVersion ()	.....	113, 123, 124, 155, 156
print ()	.....	54, 55, 69, 181
set*() ()	.....	85
set*Variable ()	.....	133
set* ()	.....	69
translate*() ()	.....	88
visit* ()	.....	98, 170
visitEnd ()	.....	98, 170
visitStart ()	.....	31, 98, 100, 170
xml2lyInsiderHandler		
createTheXml2lyOptionGroups ()	.....	274
xml2lyRegularHandler		
createInformationsRegularGroup ()	.....	156
xml2xmlInsiderHandler		
xml2xmlInsiderHandler ()	.....	120
xmlelement		
fType	.....	224
yy::parser		
make_OPTION ()	.....	378
()	.....	156, 166

# Constants, functions and variables index

## Symbols

AllFirst .....	82
AllLast .....	82
K_STANZA_NUMBER_UNKNOWN .....	76
TrueHarmoniesFirst .....	83
TrueHarmoniesLast .....	83
gGlobalOStreamIndenter .....	76
gGlobalOahOahGroup .....	153
gGlobalServiceRunData .....	272
gGlobalTimingItemsList .....	76
gIndenter .....	52
i .....	70
kComponentUsedFromTheCLIYes .....	114
kPlacementAbove .....	189
kSlurTypeRegularStart .....	77
k_bass .....	31
k_notations .....	31
k_other_notation .....	31
k_part_name .....	31
pPrivateThisMethodHasBeenRun .....	76
*AsString() () .....	87
catchSignals () .....	398
convert*() () .....	89
convertArgcArgvToOptionsAndArguments () ..	140, 144
create*OahGroup () .....	272
create*PassComponent () .....	115

create*RepresentationComponent () .....	115
createGlobalHarmoniesExtraOahGroup () .....	160
createLibraryComponent () .....	118, 120
createMsrRepresentationComponent () .....	115
createMusicxml2brailleConverterComponent () ..	116
createMxmlScorePartWiseElement () .....	241
create () .....	91
fromString () .....	87
getopt* () .....	129
initialize* () .....	273
initializeBSR () .....	273
initializeLPSR () .....	273
initializeMSRBasicTypes () .....	273
initializeMSR () .....	273
initializeMsrGenerationAPI () .....	76
main () .....	398
mfAssert () .....	65
musicxmlfile2lilypond () .....	129
printMxsr () .....	227
translateLpsrToLilypondWithHandler () .....	52
translateMsrToLpsrScore () .....	59
yylex () .....	378

## A

argc/argv .....	138--140
-----------------	----------



# Options

## Symbols

--hist, --history .....	113	-display-options-values .....	159
--v, --version .....	113	-display-options-values, -dov .....	130
-auto-output-file-name, -aofn .....	232, 242	-files .....	235
-auto-utf8, -au8d .....	279	-find .....	131
-cpu .....	58	-history, -hist .....	114, 115, 120, 122
-display* .....	173, 398	-ignore-musicxml-figured-bass, -ofigbass .....	360
-display-lpsr, -dlpsr .....	356	-ignore-musicxml-harmonies, -oharms .....	343
-display-lpsr-full, -dlpsrfull .....	356	-input-line-numbers, -iln .....	398
-display-lpsr-short, -dlpsrshort .....	356, 400	-insider .....	48, 130, 148
-display-msr-1, -dmsr1 .....	340, 356	-insider, -ins .....	152
-display-msr-1-details, -dmsr1d .....	340	-lilypond-generation-infos, -lpgi .....	124
-display-msr-1-full, -dmsr1full .....	356	-lilypond-version, -lpv .....	134
-display-msr-1-short, -dmsr1s .....	340	-output-file-name, -o .....	232, 242
-display-msr-1-short, -dmsr1short .....	356	-quiet, -q .....	148
-display-msr-2, -dmsr2 .....	356	-reverse-names-display-order, -rndo .....	153, 154
-display-msr-2-details, -dmsr2d .....	340	-trace-components, -tcomps .....	148
-display-msr-2-full, -dmsr2full .....	356	-trace-figured-bass, -tfigbass .....	356
-display-msr-2-short, -dmsr2short .....	356	-trace-harmonies, -tharms .....	340
-display-msr-2-short, -msr2s .....	340	-trace-oah, -toah .....	148
-display-msr-2msr2, -dmsr2 .....	340	-trace-oah-details, -toahd .....	148
-display-msr-skeleton, -dmsrskel .....	340, 356	-trace-passes, -tpasses .....	398
		-version, -v .....	114, 115, 120, 122, 148, 156



# MusicXML index

## Symbols

!DOCTYPE </> .....	222, 228
?xml </> .....	222, 228
backup </> .....	153
barLine </> .....	317
barline </> .....	318
defaults </> .....	105
direction </> .....	176
direction-type </> .....	177
divisions </> .....	175
forward </> .....	153

metronome </> .....	176, 177
millimeter </> .....	100
notations </> .....	30
part </> .....	342, 360
print </> .....	105
repeat </> .....	78
scaling </> .....	100
system-layout </> .....	105
tenth </> .....	100
words </> .....	176, 177

# Main index

## Symbols

.	74, 110	-display-msr-1, -dmsr1	340, 356
.cpp	70	-display-msr-1-details, -dmsr1d	340
.h	49	-display-msr-1-full, -dmsr1full	356
	144	-display-msr-1-short, -dmsr1s	340
	181, 403	-display-msr-1-short, -dmsr1short	356
TEX root	20	-display-msr-2, -dmsr2	356
<!DOCTYPE />	222, 228	-display-msr-2-details, -dmsr2d	340
<<	78	-display-msr-2-full, -dmsr2full	356
<?xml />	222, 228	-display-msr-2-short, -dmsr2short	356
<backup />	153	-display-msr-2-short, -msr2s	340
<barLine />	317	-display-msr-2msr2, -dmsr2	340
<barline />	318	-display-msr-skeleton, -dmsrskel	340, 356
<defaults />	105	-display-options-values	159
<direction />	176	-display-options-values, -dov	130
<direction-type />	177	-files	235
<divisions />	175	-find	131
<forward />	153	-history, -hist	114, 115, 120, 122
<metronome />	176, 177	-ignore-musicxml-figured-bass, -ofigbass	360
<millimeter />	100	-ignore-musicxml-harmonies, -oharms	343
<notations />	30	-input-line-numbers, -iln	398
<part />	342, 360	-insider	48, 130, 148
<print />	105	-insider, -ins	152
<repeat />	78	-lilypond-generation-infos, -lpgi	124
<scaling />	100	-lilypond-version, -lpv	134
<std::regex>	25	-m ... .. hyperpage	268
<system-layout />	105	-output-file-name, -o	232, 242
<tenth />	100	-quiet, -q	148
<words />	176, 177	-reverse-names-display-order, -rndo	153, 154
**/	18	-trace-components, -tcomps	148
*.cpp	46	-trace-figured-bass, -tfigbass	356
*.h	18	-trace-harmonies, -tharms	340
*Both*	23	-trace-oah, -toah	148
*EnumTypes*	92	-trace-oah-details, -toahd	148
*Interface.*	40	-trace-passes, -tpasses	398
*Name*	23	-version, -v	114, 115, 120, 122, 148, 156
*Repr	23	.github/workflows	257
++gIndenter	54	.h	91
,	224	.yml	257
--gIndenter	54	/Applications	262
--hist, --history	113	0e85f994ab00ea2dd94ddcb1895cbae5a32f072a	402
--v, --version	113	1.0.0	36
-auto-output-file-name, -aofn	232, 242	2.22.0	134
-auto-utf8, -au8d	279	2cb4d5f	403
-cpu	58	662454a	403
-display*	173, 398	:	225
-display-lpsr, -dlpsr	356	AllFirst	82
-display-lpsr-full, -dlpsrfull	356	AllLast	82
-display-lpsr-short, -dlpsrshort	356, 400	Bool	80
		C++-calc	375

CommonLaTeXFiles .....	20	chmod .....	263
Downloads .....	260	clean .....	36, 399
EXP .....	67, 74	clisamples .....	32
Enum* .....	83	cmake .....	25, 63
EnumTrueHarmonies .....	83	common .....	31
Interface .....	37	components .....	33
IntroductionToMusicxml .....	21	converters .....	33
K_STANZA_NUMBER_UNKNOWN .....	76	countnotes .....	28
LaTeXCommonSettings.tex .....	31	cubase .....	130
LilyPondIssue34 .....	33, 245	doc .....	28
Makefile .....	26, 28, 29, 36, 376, 399	documentation .....	20
Makefile_ORIGINAL .....	29	dynamic_cast .....	75
Mikrokosmos3Wandering .....	33, 121	elements.bash .....	32
Mikrokosmos3WanderingInsiderHandler .....	121	elements.h .....	30
MusicAndHarmonies.cpp .....	244	elements.txt .....	30
MusicFormatsVersionDate.txt .....	21, 265	f .....	85
MusicFormatsVersionNumber.txt .....	21, 22, 265	fBooleanVariable .....	133
NonConst .....	86	fContainingFindableElementInfo .....	151
OPTION .....	377	fHandlerArgumentsVector .....	139
Rational .....	178	fHandlerMultiComponent .....	120, 121
SMARTP .....	84	fInsiderHandler .....	143
SXMLFile .....	227	fMacroAtomsList .....	278
S_FiguredBass .....	360	fMeasureElementMeasurePosition .....	178
S_accidental .....	29	fPartGroupElementsList .....	93
S_figured_bass .....	361, 362	fPendingArgvAtomExpectingAValue .....	139
S_harmony .....	342, 344, 345	fReverseNamesDisplayOrder .....	153
S_msrBarLine .....	165	fSetByAnOption .....	133
S_msrFiguredBass .....	365, 368	fTempoBeatUnit .....	189
S_msrHarmony .....	354	fTempoEquivalentBeatUnit .....	189
S_msrPartGroup .....	92, 93	fTempoKind .....	189
S_msrVoice .....	166	fTempoNotesRelationshipKind .....	189
S_part_name .....	31	fTempoNotesRelationshipLeftElements .....	189
SetMusicFormatsVersionNumber.bash .....	115	fTempoNotesRelationshipRightElements .....	189
TRACING_IS_ENABLED .....	48	fTempoParenthesizedKind .....	189
TXMLFile .....	227	fTempoPerMinute .....	189
TrueHarmoniesFirst .....	83	fTempoPlacementKind .....	189
TrueHarmoniesLast .....	83	fTempoWordsList .....	189
.....	157, 224, 273	fThereAreHarmoniesToBeAttachedToCurrentNote	
\${HOME}/Downloads .....	267	342	
oah .....	61	fType .....	224
__declspec(dllexport) .....	74	fVersionsList .....	111
_msrMeasure .....	203	fVoiceLastSegment .....	198
a880063c134a7ba49b31f5fb52b47f682058f64a .....	403	false .....	133
addAll .....	265	fetch .....	85
api.prefix .....	383	files .....	28, 36
bin .....	262, 263	find .....	18
bool .....	80, 375	formats .....	35
braille .....	61	formatsgeneration .....	35
brailleGeneration .....	35	gGlobalOStreamIndenter .....	76
bsr .....	35, 61	gGlobalOahOahGroup .....	153
bsr2braille .....	35	gGlobalServiceRunData .....	272
bsr2brailleTranslator .....	234	gGlobalTimingItemsList .....	76
bsr2bsr .....	35	gIndenter .....	52, 54
bsrBrailleGenerator .....	235	generators .....	33
bsrCellKind .....	219, 220	get .....	85
build .....	28	git bisect skip .....	403
build-ubuntu-version.yml .....	257	git log .....	401
build/bin .....	26	git push .....	257, 265, 401
c .....	87	graphics .....	21
calc++ .....	378	guidoGeneration .....	35
char* .....	378	i .....	70

javascript	28	msr2bsr	35
kComponentUsedFromTheCLIYes	114	msr2guido	33
kPlacementAbove	189	msr2lilypond	33
kSlurTypeRegularStart	77	msr2lpsr	35
kTempoNotesRelationshipEquals	189	msr2msr	35
k_accidental	29	msr2msrTranslator	350, 366
k_bass	31	msr2musicxml	33
k_notations	30, 31	msr2mxsr	35
k_other_notation	31	msr2mxsrOah	153
k_part_name	31	msr2mxsrTranslator	106, 240
lambda	96	msrAccordionRegistration	194
lib	262	msrBarCheck	192
libmusicxml	28	msrBarLine	192, 317, 319
libmusicxml/build/bin	28	msrBarNumberCheck	192
libmusicxml/src	222	msrBeatRepeat	200
libmusicxml2Presentation	21	msrBeatRepeatPattern	201
lilypond	36, 61	msrBeatRepeatReplicas	201
lilypondGeneration	35	msrBook	195, 216
lpsr	35, 61	msrChord	95, 164, 192, 198, 340, 357
lpsr2lilypond	35	msrChordInKind	78
lpsr2lilypondTranslator	102, 103, 232	msrChords	192
lpsrComment	124	msrClef	190, 194
lpsrPitchesLanguageAtom	130	msrClefKind	190
lpsrScore	44, 124, 217, 233	msrCoda	194
make	25, 28, 36, 64, 399	msrDalSegno	194
make_	377	msrDamp	194
make_...	382	msrDampAll	194
master	399	msrDottedDuration	186
mf	61	msrDoubleTremolo	95
mfEnumAll	66	msrDurationKind	177, 185, 186
mfMultiGenerationOutputKindAtom	138	msrElement	192
mfOnOffKind	80	msrEyeGlasses	194
mfOptionsAndArguments	144, 145	msrFiguredBass	208, 356, 359, 362--364, 369
mfServiceRunData	42	msrGeneration	35
mfTimingItemsList	58	msrGraceNotesGroup	131, 192
mfcComponentKind	112	msrHarmony	208, 344--349, 354
mfcComponent	112	msrHarmonyKind	81
mfcConverterComponent	113	msrHarpPedalsTuning	194
mfcGeneratorComponent	113	msrHiddenMeasureAndBarLine	192
mfcLibraryComponent	113	msrHumdrumScotKeyItem	190
mfcMultiComponent	113, 114, 120, 155	msrIdentification	176
mfcMultiComponentEntropicityKind	114	msrKey	85, 190, 194
mfcMultiComponentUsedFromTheCLIKind	114	msrLength	157, 158
mfcPassComponent	113	msrLigature	181
mfcRepresentationComponent	113	msrLigatureKind	180
mfcVersionDescr	111	msrLineBreak	194
mfcVersionNumber	110	msrMeasure	95, 102, 174, 289, 290
mfcVersionsHistory	111	msrMeasureElement	95, 177, 178
mfsl	25	msrMeasureElements	198, 283
mfslDriver	378	msrMeasureRepeat	202
mfutilities	34	msrMeasureRepeatPattern	202
msdl	35, 61	msrMeasureRepeatReplicas	202
msdl2braille	33	msrMeasuresSlice	179, 212
msdl2guido	33	msrMeasuresSlicesSequence	179, 212
msdl2lilypond	33	msrMeausre	198
msdl2musicxml	33	msrMoment	178
msdlconverter	33	msrMultipleFullBarRests	102, 203
msdr	35	msrNote	95, 108, 165, 177, 180, 205, 206, 208, 210, 340, 343, 346, 357, 360, 363
msr	35, 61	msrNoteEvent	180
msr*Element	192	msrNoteEventKind	180, 211
msr2braille	33		

msrNoteKind .....	206	mxsr2msrOah .....	153
msrOctaveEntryVariable .....	280, 281	mxsr2msrSkeletonBuilder .....	31, 40
msrOctaveShift .....	194	mxsr2msrTranslator ...	230, 345, 346, 362, 363
msrPageBreak .....	194	mxsr2musicxml .....	35
msrPart 92, 93, 192, 209, 295, 340, 341, 357, 358		mxsrGeneration .....	35
msrPartGroup .....	91, 93, 192, 195	oah .....	34
msrPartGroupElement .....	91--93, 192	oah*Element .....	130
msrPathToVoice .....	216	oahAtom .....	129--131
msrPedal .....	194	oahAtomExpectingAValue .....	132
msrPrintLayout .....	194	oahAtomImplicitlyStoringAValue .....	137
msrRehearsalMark .....	194	oahAtomStoringAValue .....	130, 132, 133, 135
msrRepeat .....	198, 199, 203	oahBooleanAtom .....	130, 133, 153
msrRepeatCommonPart .....	203, 204	oahColorRGBAtom .....	133
msrRepeatEnding .....	203, 204	oahCombinedBooleansAtom .....	130
msrRepeatEndingKind .....	204	oahContactAtom .....	130
msrScaling .....	100	oahElement .....	129, 131
msrScordatura .....	194	oahElementHelpOnlyKind .....	146
msrScore .....	102, 126, 195, 216, 329	oahElementUse .....	138
msrSegment .....	192, 198, 282, 309	oahFindStringMatch .....	150
msrSegno .....	194	oahFindableElement .....	130, 131, 150
msrSlur .....	174	oahFloatAtom .....	130
msrSpannerTypeKind .....	180	oahGroup .....	129, 130
msrStaff .....	192, 341, 358	oahHandler .....	120, 129, 130
msrStaffDetails .....	194	oahHistoryAtom .....	155
msrStaffLevelElement .....	194	oahIntegerAtom .....	130
msrStanza .....	210	oahLengthAtom .....	130, 157
msrSyllable .....	192, 194, 210	oahLengthUnitKindAtom .....	133
msrSyllableKind .....	210	oahMacroAtom .....	277
msrTempo .....	84, 177, 189, 194	oahMultiplexBooleansAtom .....	130
msrTempoKBeatUnitsKind .....	186	oahOahGroup .....	153
msrTempoNote .....	187	oahOptionsVector .....	132, 137
msrTempoTuplet .....	187	oahPrefix .....	130
msrTime .....	194	oahPureHelpAtomExpectingAValue .....	136
msrTimeSignature .....	70, 191, 192	oahPureHelpAtomWithoutAValue .....	136
msrTimeSignatureSymbolKind .....	191	oahRationalAtom .....	130
msrTransposition .....	194	oahSubGroup .....	129, 130
msrTuplet .....	95, 107, 131, 340, 357	oahThreeBooleansAtom .....	130
msrTupletElement .....	95, 192, 194	oahVersionAtom .....	123, 155
msrTupletFactor .....	207	option hyperpage .....	377
msrTupletInKind .....	207	pPrivateThisMethodHasBeenRun .....	76
msrVoice ... 167, 197, 210, 282, 295, 341, 358		packages .....	28
msrVoiceCreateInitialLastSegmentKind .....	198	passes .....	33, 35
msrVoiceElement .....	192, 196--198, 204, 282	patch_662454a.txt .....	404
msrVoiceKind .....	198	prefix .....	380
msrVoiceStaffChange .....	194	presentation .....	32
msrapi .....	35	rmcache .....	64
msrmeasure .....	199	s .....	65, 69, 75, 77, 78, 84, 317
multiGeneration .....	35	samples .....	28
musicformats-*-distrib .....	262, 265, 266	schemas .....	28
musicformats-ubuntu-distrib .....	260	scripts .....	32
musicformats-windows-distrib .....	262	set .....	85
musicformatsversion.txt .....	36	src .....	28, 32
musicxml2braille .....	33, 116	star* .....	23
musicxml2guido .....	33	std::ostream .....	236
musicxml2lilypond .....	33	v .....	80
musicxml2musicxml .....	33	v* .....	48
mxmlelement .....	222	v... ..	18
mxsr .....	35, 61, 242	v0.9.61 .....	115
mxsr2guido .....	35	v0.9.63 .....	265
mxsr2msr .....	35	v0.9.65 .....	257
		vX.Y.Z .....	265

validation .....	28	checkMissingPendingArgvAtomExpectingAValueValue ()	140
visitStart .....	100	collectPartMeasuresSlices ()	213
visitStart (S_msrMeasure& elt) .....	287	collectStaffMeasuresSlices ()	214, 215
wae .....	35	compare* ()	90
win32 .....	28	convert*() ()	89
xattr .....	263	convertArgcArgvToOptionsAndArguments ()	140, 144
xml2brl .....	61, 130	countnotes	
xml2guido .....	28	visitStart .....	100
xml2lbr .....	130	create*OahGroup ()	272
xml2ly .....	48, 61, 130	create*PassComponent ()	115
xml2lyInsiderHandler .....	274	create*RepresentationComponent ()	115
xml2xml .....	130	create* ()	69, 71, 84
xmlattribute .....	223	createAMeasureAndAppendItToPart ()	290
xmlelement .....	223, 224	createAMeasureAndAppendItToSegment ()	199, 287
yy::location .....	375	createAMeasureAndAppendItToVoice ()	287, 311
yy::parser::make_ .....	377	createGlobalHarmoniesExtraOahGroup ()	160
musicxml2ly .....	245	createGlobalMxsr2msrOahGroup ()	49
*AsString() ()	87	createInformationsRegularGroup ()	156
//#define DEBUG_EARLY_OPTIONS		createLibraryComponent ()	118, 120
i .....	70	createMeasureRepeatFromItsFirstMeasures ()	287, 310, 315
//#define DEBUG_INDENTER		createMsrRepresentationComponent ()	115
i .....	70	createMusicxml2brailleConverterComponent ()	116
//#define DEBUG_SPLITTING		createMxmlScorePartWiseElement ()	241
i .....	70	createNewLastSegmentForVoice ()	310
handleOptionNameCommon ()	138	createNewLastSegmentFromItsFirstMeasureForVoice ()	287, 311
acceptIn ()	69	createPartFiguredBassVoice ()	359
acceptOut ()	69	createSegmentDeepClone ()	309
addGraceNotesGroupBeforeAheadOfVoiceIfNeeded ()	311	createSegmentNewbornClone ()	309
appendBarLineToVoice ()	311	createTempoPerMinute ()	76
appendCodaToVoice ()	311	createTheXml2lyOptionGroups ()	274
appendDampAllToVoice ()	311	create ()	76, 91, 309, 345
appendDampToVoice ()	311	displayMeasureRepeat ()	89
appendElementToMeasure ()	283	fetchNames ()	154
appendEyeGlassesToVoice ()	311	finalize*() ()	87
appendHarmonyToNote ()	347, 348	finalize* ()	293
appendMeasureCloneToMultipleFullBarRests ()	287	finalizeAllTheMeasuresOfSegment ()	293
appendMeasureCloneToVoiceClone ()	287	finalizeChord ()	293
appendMeasureToSegment ()	285, 287	finalizeCurrentChord ()	293
appendMultipleFullBarRestsToVoice ()	287	finalizeCurrentMeasureClone ()	293
appendNoteToVoice ()	349, 365	finalizeFiguredBassMeasure ()	293
appendPedalToVoice ()	311	finalizeHarmonyMeasure ()	293
appendPendingMeasureRepeatToVoice ()	287	finalizeLastAppendedMeasureInPart ()	295
appendSegnoToVoice ()	311	finalizeLastAppendedMeasureInStaff ()	295
appendStaffDetailsToPart ()	108	finalizeLastAppendedMeasureInVoice ()	295
appendStaffDetailsToVoice ()	311	finalizeMeasureClone ()	293
applyAtomWithDefaultValue ()	139	finalizeMeasure ()	292, 293, 299
applyAtomWithValue ()	139, 158	finalizePartAndAllItsMeasures ()	295
applyEarlyOptionIfRelevant ()	149	finalizePartClone ()	293
applyElement ()	123, 139, 148, 278	finalizePart ()	293, 296
applyOptionsFromElementUsesList ()	146	finalizeRegularMeasure ()	293
asString() ()	88	finalizeRepeatEndInPart ()	294
asStringShort() ()	88	finalizeRepeatEndInStaff ()	294
asString ()	54, 55, 69, 181, 182	finalizeRepeatEndInVoice ()	294
assertSegmentElementsListIsNotEmpty ()	312	finalizeStaff ()	293, 296, 297
browseData ()	69, 102, 326	finalizeTupletAndPopItFromTupletsStack ()	293
browsedata ()	98, 102		
catchSignals ()	398		
checkGroupOptionsConsistency ()	152		



<code>finalizeTuplet ()</code> .....	293	<code>printHistory ()</code> .....	114
<code>finalizeVoiceAndAllItsMeasures ()</code> .....	295	<code>print ()</code> .....	114
<code>finalizeVoice ()</code> .....	293, 297	<code>mfslDriver</code>	
<code>findStringInFindableElement ()</code> .....	151, 152	<code>parseInput_Pass1 ()</code> .....	379
<code>fromString ()</code> .....	87	<code>msdlTokenKind</code>	
<code>generateCodeForBrailleCell ()</code> .....	236	<code>AllFirst</code> .....	82
<code>generateLilypondVersion ()</code> .....	134	<code>AllLast</code> .....	82
<code>get* () ()</code> .....	85	<code>msr2bsrTranslator</code>	
<code>get* ()</code> .....	69	<code>finalizeCurrentMeasureClone ()</code> .....	293
<code>getClefStaffNumber ()</code> .....	240	<code>msr2lpsrTranslator</code>	
<code>getSetByAnOption ()</code> .....	134	<code>visitStart (S_msrMeasure&amp; elt)</code> .....	287
<code>getValue ()</code> .....	375	<code>msr2msrTranslator</code>	
<code>getopt* ()</code> .....	129	<code>visitStart (S_msrMeasure&amp; elt)</code> .....	287
<code>handleHooklessRepeatEndingEndInVoice ()</code> ..	311	<code>msr2mxsrTranslator</code>	
<code>handleKnownArgvAtom ()</code> .....	139	<code>translateMsrToMxsr ()</code> .....	240
<code>handleMultipleFullBarRestsStartInVoiceClone ()</code>	311	<code>msrChord</code>	
<code>handleOptionNameCommon ()</code> .....	138	<code>finalizeChord ()</code> .....	293
<code>handleOptionsAndArgumentsFromArgcArgv ()</code> ..	138	<code>msrClef</code>	
<code>handlePendingFiguredBasses ()</code> .....	364	<code>getClefStaffNumber ()</code> .....	240
<code>handleVoiceLevelContainingRepeatEndWithoutStart ()</code>	311	<code>msrHarmonyKind</code>	
<code>handleVoiceLevelRepeatEndWithStart ()</code> .....	311	<code>AllFirst</code> .....	82
<code>handleVoiceLevelRepeatEndWithoutStart ()</code> ..	311	<code>AllLast</code> .....	82
<code>handleVoiceLevelRepeatEndingStartWithExplicitStart ()</code>	287, 311	<code>TrueHarmoniesFirst</code> .....	83
<code>handleVoiceLevelRepeatEndingStartWithoutExplicitStart ()</code>	287, 311	<code>TrueHarmoniesLast</code> .....	83
<code>handleVoiceLevelRepeatStart ()</code> .....	287, 311	<code>msrHarmony</code>	
<code>identifySoloNotesAndRests ()</code> .....	215	<code>create ()</code> .....	345
<code>initialize* () ()</code> .....	86	<code>msrMeasureElement</code>	
<code>initialize* ()</code> .....	273	<code>fMeasureElementMeasurePosition</code> .....	178
<code>initializeBSR ()</code> .....	273	<code>msrMeasureRepeat</code>	
<code>initializeHandlerMultiComponent ()</code> ..	120, 121	<code>displayMeasureRepeat ()</code> .....	89
<code>initializeLPSR ()</code> .....	273	<code>msrMeasuresSlicesSequence</code>	
<code>initializeLpsrPaperOptions ()</code> .....	160	<code>identifySoloNotesAndRests ()</code> .....	215
<code>initializeMSRBasicTypes ()</code> .....	273	<code>msrMeasure</code>	
<code>initializeMSR ()</code> .....	273	<code>appendElementToMeasure ()</code> .....	283
<code>initializeMsrGenerationAPI ()</code> .....	76	<code>finalizeFiguredBassMeasure ()</code> .....	293
<code>initializeOahBasicHelpOptions ()</code> .....	153, 156	<code>finalizeHarmonyMeasure ()</code> .....	293
<code>initializeVoice ()</code> .....	309	<code>finalizeMeasureClone ()</code> .....	293
<code>lpsr2lilypondTranslator</code>		<code>finalizeMeasure ()</code> .....	292, 293, 299
<code>generateLilypondVersion ()</code> .....	134	<code>finalizeRegularMeasure ()</code> .....	293
<code>lpsrOahGroup</code>		<code>msrMultipleFullBarRests</code>	
<code>initializeLpsrPaperOptions ()</code> .....	160	<code>appendMeasureCloneToMultipleFullBarRests ()</code>	287
<code>lpsrScore</code>		<code>msrNote</code>	
<code>lpsrScore ()</code> .....	124	<code>appendHarmonyToNote ()</code> .....	347, 348
<code>lpsrScore ()</code> .....	124	<code>msrPart</code>	
<code>main ()</code> .....	398	<code>appendStaffDetailsToPart ()</code> .....	108
<code>make_OPTION ()</code> .....	378	<code>browseData ()</code> .....	326
<code>mfAssert ()</code> .....	65	<code>collectPartMeasuresSlices ()</code> .....	213
<code>mfIndentedStreamBuf</code>		<code>createAMeasureAndAppendItToPart ()</code> .....	290
<code>sync ()</code> .....	51	<code>createPartFiguredBassVoice ()</code> .....	359
<code>mfMultiGenerationOutputKindAtom</code>		<code>finalizeLastAppendedMeasureInPart ()</code> ..	295
<code>mfMultiGenerationOutputKindAtom ()</code> .....	138	<code>finalizePartAndAllItsMeasures ()</code> .....	295
<code>mfMultiGenerationOutputKindAtom ()</code> .....	138	<code>finalizePartClone ()</code> .....	293
<code>mfcComponent</code>		<code>finalizePart ()</code> .....	293, 296
<code>printOwnHistory ()</code> .....	113	<code>finalizeRepeatEndInPart ()</code> .....	294
<code>mfcMultiComponentUsedFromTheCLIKind</code>		<code>msrPlacementKind</code>	
<code>kComponentUsedFromTheCLIYes</code> .....	114	<code>kPlacementAbove</code> .....	189
<code>mfcMultiComponent</code>		<code>msrSegment</code>	
		<code>appendMeasureToSegment ()</code> .....	285, 287
		<code>assertSegmentElementsListIsEmpty ()</code>	312

```

createAMeasureAndAppendItToSegment () 199,
287
createSegmentDeepClone () .....309
createSegmentNewbornClone () .....309
create () .....309
finalizeAllTheMeasuresOfSegment () ....293
msrSlurTypeKind
    kSlurTypeRegularStart .....77
msrStaff
    collectStaffMeasuresSlices () .....214, 215
    finalizeLastAppendedMeasureInStaff () .295
    finalizeRepeatEndInStaff () .....294
    finalizeStaff () .....293, 296, 297
msrStanza
    K_STANZA_NUMBER_UNKNOWN .....76
msrTempoBeatUnitsKindAsString () .....76
msrTempo
    createTempoPerMinute () .....76
    fTempoBeatUnit .....189
    fTempoEquivalentBeatUnit .....189
    fTempoKind .....189
    fTempoNotesRelationshipKind .....189
    fTempoNotesRelationshipLeftElements ...189
    fTempoNotesRelationshipRightElements ..189
    fTempoParenthesizedKind .....189
    fTempoPerMinute .....189
    fTempoPlacementKind .....189
    fTempoWordsList .....189
    kTempoNotesRelationshipEquals .....189
msrTemp
    msrTempoBeatUnitsKindAsString () .....76
msrTie
    create () .....76
msrTuplets
    finalizeTuplet () .....293
msrVoice
    addGraceNotesGroupBeforeAheadOfVoiceIfNeeded ()
    311
    appendBarLineToVoice () .....311
    appendCodaToVoice () .....311
    appendDampAllToVoice () .....311
    appendDampToVoice () .....311
    appendEyeGlassesToVoice () .....311
    appendMeasureCloneToVoiceClone () .....287
    appendMultipleFullBarRestsToVoice () ..287
    appendNoteToVoice () .....349, 365
    appendPedalToVoice () .....311
    appendPendingMeasureRepeatToVoice () ..287
    appendSegnoToVoice () .....311
    appendStaffDetailsToVoice () .....311
    createAMeasureAndAppendItToVoice () ..287,
    311
    createMeasureRepeatFromItsFirstMeasures ()
    287, 310, 315
    createNewLastSegmentForVoice () .....310
    createNewLastSegmentFromItsFirstMeasureForVoice ()
    287, 311
    fVoiceLastSegment .....198
    finalizeLastAppendedMeasureInVoice () .295
    finalizeRepeatEndInVoice () .....294
    finalizeVoiceAndAllItsMeasures () .....295
    finalizeVoice () .....293, 297
    handleHooklessRepeatEndingEndInVoice ()
    311
    handleMultipleFullBarRestsStartInVoiceClone ()
    311
    handleVoiceLevelContainingRepeatEndWithoutStart ()
    311
    handleVoiceLevelRepeatEndWithStart () .311
    handleVoiceLevelRepeatEndWithoutStart ()
    311
    handleVoiceLevelRepeatEndingStartWithExplicitStart
    287, 311
    handleVoiceLevelRepeatEndingStartWithoutExplicitSta
    287, 311
    handleVoiceLevelRepeatStart () ... 287, 311
    initializeVoice () .....309
musicxmlfile2lilypond () .....129
mxsr2msrSkeletonBuilder
    fThereAreHarmoniesToBeAttachedToCurrentNote
    342
mxsr2msrTranslator
    finalizeCurrentChord () .....293
    finalizeTupletAndPopItFromTupletsStack ()
    293
    handlePendingFiguredBasses () .....364
    populateNote () .....346, 363
    visitStart ( S_bass& elt) () .....31
    visitStart ( S_notations& elt) () .....31
    visitStart ( S_other_notation& elt) () .31
    visitStart ( S_part_name& elt) () .....31
oahAtomStoringAValue
    fSetByAnOption .....133
oahEarlyOptions
    applyEarlyOptionIfRelevant () .....149
oahElement
    fetchNames () .....154
oahHandler
    handleOptionNameCommon () .....138
    applyOptionsFromElementUsesList () ....146
    checkMissingPendingArgvAtomExpectingAValueValue ()
    140
    fHandlerArgumentsVector .....139
    fHandlerMultiComponent .....120, 121
    fPendingArgvAtomExpectingAValue .....139
    findStringInFindableElement () .....151
    handleKnownArgvAtom () .....139
    handleOptionNameCommon () .....138
    handleOptionsAndArgumentsFromArgcArgv ()
    138
oahHistoryAtom
    printHistory () .....156
oahIntegerAtom
    setIntegerVariable () .....133
oahLengthAtom
    applyAtomWithValue () .....158
    printAtomWithVariableOptionsValues () .159
oahMacroAtom
    applyElement () .....278
    fMacroAtomsList .....278
oahOahGroup
    initializeOahBasicHelpOptions () .153, 156

```



printOahOahValues () ..... 153  
 oahPrefix  
   findStringInFindableElement () ..... 152  
 oahVersionAtom  
   applyElement () ..... 123, 148  
   printVersion () ..... 123  
 parseInput\_Pass1 () ..... 379  
 populateNote () ..... 346, 363  
 printAtomWithVariableOptionsValues () ..... 159  
 printHistory () ..... 113, 114, 124, 155, 156  
 printMxsr () ..... 227  
 printOahOahValues () ..... 153  
 printOwnHistory () ..... 113  
 printSlices () ..... 108  
 printVersion () ..... 113, 123, 124, 155, 156  
 print () ..... 54, 55, 69, 114, 181  
 set\*() () ..... 85  
 set\*Variable () ..... 133  
 set\* () ..... 69  
 setIntegerVariable () ..... 133  
 sync () ..... 51  
 translate\*() () ..... 88  
 translateLpsrToLilypondWithHandler () ..... 52  
 translateMsrToLpsrScore () ..... 59  
 translateMsrToMxsr () ..... 240  
 visit\* () ..... 98, 170  
 visitEnd () ..... 98, 170  
 visitStart ( S\_bass& elt) () ..... 31  
 visitStart ( S\_notations& elt) () ..... 31  
 visitStart ( S\_other\_notation& elt) () ..... 31  
 visitStart ( S\_part\_name& elt) () ..... 31  
 visitStart () ..... 31, 98, 100, 170  
 xml2lyInsiderHandler  
   createTheXml2lyOptionGroups () ..... 274  
 xml2lyRegularHandler  
   createInformationsRegularGroup () ..... 156  
 xml2xmlInsiderHandler  
   xml2xmlInsiderHandler () ..... 120  
 xml2xmlInsiderHandler () ..... 120  
 xmlelement  
   fType ..... 224  
 yy::parser  
   make\_OPTION () ..... 378  
 yylex () ..... 378  
   () ..... 156, 166  
   () ..... 156, 166

**A**

actions ..... 257  
 Allow Anyway ..... 264  
 API .. 17, 24, 41, 42, 52, 114, 129, 130, 139,  
   145, 244, 252  
 argc/argv ..... 138--140  
 auto ..... 25

**B**

basic blocs ..... 205  
 bisect ..... 402  
 BMML ..... 32  
 Braille ..... 17, 393  
 branch ..... 18, 25, 48, 265, 269

BSR ..... 17, 37, 172, 219  
 bsr.cpp ..... 76  
 bsrEnumTypes ..... 219  
 bsrEnumTypes.h ..... 219

**C**

C++17 ..... 25, 65, 77, 91  
 cache ..... 63, 64  
 cascade ..... 290, 297  
 cascaded ..... 108, 299  
 cascading ..... 108, 295, 296  
 Catalina ..... 263  
 circularity ..... 148  
 command line ... 24, 25, 32, 41, 42, 114, 129,  
   130, 132, 137, 139, 144, 161  
 commit and push again ..... 268  
 component ..... 110, 112  
 tt \*Component.h/.cpp ..... 115  
 converter ..... 37, 42  
 countnotes.cpp ..... 100

**D**

data driven ..... 98  
 \*DeepClone ..... 167  
 default ..... 18, 257  
 defensive programming ..... 65  
 denormalization ..... 164, 340, 354, 357, 369  
 description ..... 17  
 displayMusicformatsHistory.cpp ..... 120  
 displayMusicformatsVersion.cpp ..... 120  
 DLL ..... 74  
 Dominique Fober .. 16, 17, 21, 32, 67, 68, 98,  
   222  
 drawing ..... 317  
 drawn ..... 177, 309, 340, 357  
 driver ..... 375  
 DTD ..... 25  
 dynamics ..... 165

**E**

elements/factory.cpp ..... 225  
 enableHarmoniesExtraOahIfDesired.h ..... 160  
 encapsulates ..... 80  
 enumeration type . 65, 69, 75, 77, 78, 80, 81,  
   84, 87, 112, 114, 146, 157, 177, 180, 181,  
   185, 186, 190, 191, 198, 204, 206, 207,  
   210, 211, 219, 220, 224, 225, 273, 317

**F**

figured bass ..... 165  
 format ..... 39  
 frozen ..... 17, 18  
 functions ..... 32, 129  
 functor ..... 96

**G**

Gatekeeper ..... 263, 264  
 General button ..... 263  
 generator ..... 41  
 Git ..... 64  
 git ..... 401, 402

GitHub .....257, 260, 265, 399  
 GUI .....177, 263  
 Guido .....17, 389

## H

harmonies .....165  
 has not been set yet .....77  
 High Sierra .....263

## I

initialization .....77, 272--274  
 insider .....121, 130, 143, 144, 148, 152, 274  
 InsiderHandler.cpp .....49  
 introspection .....150  
 \*IsNeeded .....233

## L

libMultipleInitsTest.cpp .....140  
 libmusicxml2 .....16, 17, 21, 28, 29, 32,  
     63, 67, 68, 75, 83, 84, 91, 116, 222, 224,  
     226, 230  
 LilyPond .....17,  
     42, 52, 124, 134, 165, 178, 179, 195, 217,  
     230--233, 245, 253, 354, 369, 389, 396  
 LilyPondIssue34.cpp .....42, 140  
 Linux .....266  
 location.hh .....375  
 lpsr/lpsrPapers.h .....80  
 lpsr2lilypondTranslator.cpp .....134  
 lpsrOah.cpp .....160  
 lpsrScores.cpp .....124  
 LSPR .....37, 172, 217, 251, 274  
 lyrics .....165

## M

Mac OS™.....257, 260, 262, 263, 266  
 main () .....32, 46, 138, 252  
 Makefile .....26  
 master .....18, 25  
 master branch .....257, 258, 265, 268  
 measure position .....348  
 MEI .....32  
 mfBool.h/.cpp .....80  
 MFC .....124  
 mfcBasicTypes.h .....155  
 mfcComponents.h .....38  
 mfcLibraryComponent.h/.cpp .....118  
 mfEnumAll.h .....66, 81, 82  
 mfIndentedTextOutput.cpp .....70  
 mfIndentedTextOutput.cpp.h/.cpp .....171  
 mfIndentedTextOutput.h .....52--54  
 mfIndentedTextOutput.h/.cpp .....51, 61  
 mfMusicformatsErrors.h .....139  
 mfOnOff.cpp .....80  
 mfOnOff.h .....80  
 MFSL .....374, 375, 378, 380, 382--384  
 mfs1.cpp .....375  
 mfs1Driver.cpp .....379  
 mfs1Driver.h .....378  
 mfs1InterpreterInterface.h .....385  
 mfs1Parser.cpp .....375, 377

mfs1Parser.h .....375  
 mfs1Parser.output .....375  
 mfs1Parser.yy .....375--377  
 mfs1Scanner.cpp .....375  
 mfs1Scanner.ll .....375--377, 380, 385  
 mfStringsHandling.cpp .....70  
 mfTiming.h/.cpp .....58  
 Mikrokosmos3Wandering.cpp ...17, 41, 140, 144  
 MSDL .....28, 389  
 msdl.cpp .....140, 144  
 msdlScanner.cpp .....169  
 msdlScanner.h .....168  
 msdlTokens.cpp .....82  
 MSR .....21, 37, 108, 115, 141, 165, 167, 172,  
     175--177, 186, 198, 200, 230, 251, 274,  
     328, 356, 372, 389  
 msr2msrTranslator.h .....169  
 msrBarLines.h .....63, 317  
 msrBarlines.h .....63  
 msrBasicTypes.cpp .....65, 82  
 msrBasicTypes.h .81, 157, 182, 185, 186, 198,  
     206  
 msrDurations.h .....177  
 msrElements.h .....99  
 msrIdentification.h .....176  
 msrLigatures.h .....180  
 msrMeasuresSlice.h .....179  
 msrNotes.cpp .....104, 349, 365  
 msrOah.cpp .....129  
 msrOah.h .....129  
 msrPartGroups.h .....93  
 msrPartGroupsElements.h .....92  
 msrParts.cpp .....71  
 msrParts.h .....70, 93, 341, 358  
 msrSegment.cpp .....85  
 msrSegments.cpp .....312  
 msrSpanners.h .....181  
 msrTempos.h .....186  
 msrTies.h .....85  
 msrTimeSignature.cpp .....168  
 msrTimeSignature.h .....167  
 msrVoices.cpp .....309--311  
 msrVoices.h .....90, 165, 174, 175, 198  
 MuseScore .....179  
 Music eXtended Markup Language .....247  
 MusicFormats .....1, 16--18, 20--22,  
     24, 25, 28--30, 32, 33, 35, 37, 38, 42,  
     46--48, 50, 52--54, 63, 64, 66--68, 70,  
     72, 74--76, 78, 80, 83, 84, 88--92, 95,  
     96, 98, 107, 108, 110, 112, 113, 115, 116,  
     129, 139, 143, 146, 160, 161, 164, 165,  
     167, 172--175, 180, 181, 185, 203, 209,  
     230, 252, 257, 262, 263, 265--267, 272,  
     277, 293, 309, 328, 341, 358, 372, 389,  
     398--400, 403  
 MusicXML .....16, 17, 25, 28--32,  
     40, 46, 78, 100, 105, 116, 125, 172, 175,  
     176, 180, 190, 203, 208, 222, 224--227,  
     230, 238, 240, 242, 245, 247, 290, 317,  
     328, 340, 342, 357, 360, 372, 393  
 musicxmlfiles/fullbarrests/FullBarRestWithoutBarLine.x

- 404
- musicxmlfiles/multistaff/SATBExample.xml .404
- mutual dependency .....91
- MXSR .17, 25, 29, 31, 37, 116, 141, 172, 230, 238
- mxsr2msrTranslator.cpp .....347, 364
- N**
- \*NewbornClone .....166
- no error message whatsoever .....170
- O**
- OAH ..131, 151--153, 155, 156, 160, 161, 272, 274, 375
- oahAtomsCollection .....130
- oahAtomsCollection.cpp .....133, 148, 158
- oahAtomsCollection.h .....157
- oahBasicTypes.cpp .....140, 146
- oahBasicTypes.h .....140
- oahEarlyOptions.cpp .....70, 149
- oahEarlyOptions.h .....148
- oahElement.cpp .....151
- oahElements.cpp .....152, 154
- oahElements.h .....146
- oahEnableTracingIfDesired.h .....48
- oahOah.cpp .....156
- On/off .....80
- Open .....264
- operating system ...25, 64, 80, 257, 263, 400
- P**
- part group .....91
- partial order .....326
- pass .....40
- PNG .....21
- pre-declaration .....91
- programming .....17
- pure help .....136
- Q**
- quarantine .....263
- R**
- regular .....121, 130, 143, 144, 148, 152, 156
- RegularHandler
- createOahRegularGroup () .....154
- createOahRegularGroup () .....154
- release notes .....110, 114
- repository 257, 262, 263, 265--267, 399, 401, 403
- representation .....39
- run .....42
- S**
- script .....18
- Security & Privacy .....263
- semantic .....115
- service .....42, 144
- shebang .....374
- shortcut .....164
- smart pointer .....213, 224, 225, 227
- standard output ...52, 54, 129, 181, 232, 242
- stopper sets .....387, 396
- sub-class .....92, 93, 95
- System Preferences .....263
- T**
- TeXShop .....20
- three-state .....80
- type .....31, 78, 80, 92, 93, 132, 137, 378
- U**
- Ubuntu .....257
- V**
- version .....25, 48
- very fine-grained .....175
- W**
- waeEnableAbortToDebugErrors.h .....47, 54, 398
- waeExceptions .....47
- waeHandlers.cpp .....69
- waeHandlers.h .....68
- Web .....24, 28, 52, 140, 162, 252
- Windows .....74
- Windows™ .....67, 257, 266
- X**
- xml2Any.cpp .....140
- xml2ly.cpp .....44
- xmlfile.h .....226
- xmlreader.h/.cpp .....227
- Z**
- Zip .....257, 260