MusicFormats maintainance guide

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

v0.9.61 – February 28, 2022

Jacques Menu

```
void msrSegment::browseData (basevisitor* v)
{
  // ... ... ...

  for (
    list<S_msrMeasure>::const_iterator i = fSegmentMeasuresList.begin ();
    i != fSegmentMeasuresList.end ();
    ++i
  ) {
    // browse the element
    msrBrowser<msrMeasure> browser (v);
    browser.browse (*(*i));
  } // for

  // ... ... ...
}
```

```
void msr2msrTranslator::visitStart (S_msrClef& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrClef" <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  fCurrentVoiceClone->
    appendClefToVoice (elt);
}
```

Abstract

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.

Contents

I Overview of MusicFormats **15**

1 Introduction . . . 16
1.1 Acknowledgements . . . 16
1.2 Prerequisites . . . 16
1.3 Chronology . . . 17
1.4 The GitHub repository . . . 17
2 Documentation . . . 19
2.1 LaTeX macros . . . 20
2.2 About this document . . . 22
2.3 The MusicFormats architecture . . . 22
2.4 User guide . . . 22
2.5 API guide . . . 22
2.6 Maintainance guide . . . 22
3 Building MusicFormats . . . 23
3.1 Cloning the repository . . . 23
3.2 One-shot partial `make` runs . . . 23
3.3 `cmakeMain]cmake` configuration . . . 23
3.4 Library `Makefile` . . . 24
3.5 CLI amples `Makefile` . . . 24
3.6 Building MusicFormats in practise . . . 24
4 Code base structure . . . 26
4.1 The `libmusicxml` folder . . . 26
4.2 The `doc` folder . . . 27
4.3 The `schemas` folder . . . 27
4.4 The `src` folder . . . 28
4.5 The `validation` folder . . . 31
5 Components . . . 32
5.1 Terminology . . . 32
5.2 Formats . . . 34
5.3 Representations . . . 34
5.4 Passes . . . 35
5.5 Generators . . . 36
5.6 Converters . . . 37
5.7 Running a service . . . 37
6 Command line samples . . . 41
7 Warning and errors (WAE) . . . 42
8 The trace facility . . . 43
8.1 Activating the trace . . . 43
8.2 Trace categories . . . 44
8.3 Using traces in practise . . . 44
8.4 Debugging traces handling . . . 44

9 Multi-lingual support . . . 45
10 Textual input and output . . . 46
10.1 Indented output streams . . . 46
10.2 Creating indented output streams . . . 47
10.3 Indenting the output . . . 48
10.4 Printing descriptions . . . 49
11 Binary data output . . . 51
12 CPU measurements . . . 52

II The two-phase visitors pattern 54
13 The two-phase visitors pattern . . . 55
13.1 Basic mechanism . . . 55
13.2 Browser template classes . . . 56
13.3 A first example: counting notes in MusicMXL data . . . 57
13.4 A more complex example . . . 57
13.5 Data browsing order . . . 59
13.6 Selectively inhibiting data browsing . . . 59
13.7 Adapting visitors to data browsing order with booleans . . . 61
13.8 Adapting visitors to data browsing order with stacks . . . 63
13.9 Avoiding the visiting pattern by cascading . . . 64

III Programming style and conventions 65
14 Programming style and conventions . . . 66
14.1 Files naming conventions . . . 66
14.2 Adding C++ files . . . 68
14.3 Renaming C++ files . . . 69
14.4 Source code layout . . . 69
14.5 Defensive programming . . . 70
14.6 Sanity checks . . . 71
14.7 `JMI` comments . . . 71
14.8 Exported symbols . . . 72
14.9 Smart pointers . . . 72
14.10 Files contents layout . . . 73
14.11 `#define DEBUG*` code sections . . . 74
14.12 Identifiers naming conventions . . . 75
14.13 Exceptions and warnings/errors reporting . . . 76
14.14 Exporting symbols for Windows DLLs . . . 78
14.15 Dynamic type checking . . . 79
14.16 Input line numbers . . . 79
14.17 Static declarations . . . 80
14.18 Avoiding MusicFormats multiple initializations . . . 80
14.19 Enumeration types . . . 81
14.20 yes/no enumerations types . . . 82
14.21 Boolean values anyway . . . 83
14.22 Iterating over numeration types . . . 83
14.23 Rational numbers . . . 85
14.24 Default values . . . 86
14.25 `create*` methods . . . 86
14.26 `get*()`, `set*()` and `fetch*()` methods . . . 87
14.27 `initialize*()` and `finalize*()` methods . . . 88
14.28 `*asString()` and `*fromString()` functions . . . 88
14.29 `translate*()` methods and `convert*()` functions . . . 90
14.30 `context` arguments . . . 91
14.31 Sorting and `compare*()` methods . . . 91

IV MusicFormats components **93**
15 musicformats components (MFC) . . . 94
15.1 Versions numbers . . . 94
15.2 Versions descriptions . . . 95
15.3 Versions histories . . . 95
15.4 Components descriptions . . . 96
15.5 Multi-components . . . 97
15.6 Versions history creation . . . 99
15.6.1 Representations and passes components creation . . . 99
15.6.2 Generators and converters components creation . . . 100
15.6.3 MusicFormats library component creation . . . 101
15.6.4 Version and history options handling . . . 103
15.7 Accessing versions in regular handlers . . . 105
15.8 Getting current version numbers . . . 105
15.8.1 Current version numbers in options . . . 106
15.8.2 Current version numbers in formats . . . 108
15.8.3 Current version numbers in passes . . . 109

V Options and help **111**
16 Options and help (OAH) . . . 112
16.1 OAH basics . . . 112
16.2 Features . . . 113
16.3 Atoms expecting a value . . . 113
16.3.1 The `oahAtomStoringAValue` class . . . 114
16.3.2 The `oahBooleanAtom` special case . . . 115
16.3.3 Checking whether an option has been selected . . . 116
16.3.4 The `oahAtomStoringAValue` subclasses . . . 116
16.4 Pure help atoms . . . 117
16.4.1 Pure help atoms without a value . . . 117
16.4.2 Pure help atoms expecting a value . . . 118
16.5 Options implicitly storing a value . . . 119
16.6 Options and help handling . . . 120
16.7 Basic OAH types . . . 120
16.8 Prefixes handling . . . 121
16.9 `argc/argv` versus `oahOptionsVector` . . . 121
16.10 Applying options . . . 122
16.11 A OAH atoms collection . . . 122
16.12 An option and help example . . . 123
16.13 Options and help introspection . . . 124
16.14 Insider versus regular handlers . . . 125
16.15 Deciphering the options and arguments . . . 126
16.15.1 Options and arguments multi-pass analysis . . . 126
16.15.2 Pure help runs . . . 127
16.15.3 Applying options . . . 128
16.15.4 Early handling of some options . . . 130
16.16 Adding new options . . . 131
16.16.1 Representations' vs. passes' options . . . 132
16.16.2 Using an existing OAH atom class . . . 132
16.16.3 Creating a new OAH atom class without a value . . . 134
16.16.4 Creating a new OAH atom class expecting a value . . . 136
16.17 Extra options . . . 139
16.18 `man` pages generation . . . 140
16.19 Specific global OAH groups . . . 140
16.20 Visiting OAH groups . . . 141

VI Representations **142**
17 Representations general principles . . . 143
17.1 Trees vs graphs . . . 143
17.2 Denormalization . . . 143
17.3 Newborn clones . . . 143
17.4 Deep clones . . . 145
17.5 Inheritance . . . 145
17.5.1 Single inheritance . . . 145
17.5.2 Single inheritance for smart pointers . . . 146
17.5.3 Multiple inheritance for visitors . . . 147
17.5.4 Multiple inheritance in other classes . . . 148
17.5.5 Reversibility . . . 149
18 Displaying formats . . . 150
18.1 Display categories . . . 150
18.2 Displaying in practise . . . 150
19 Music Scores Representation (MSR) . . . 151
19.1 MSR basic types . . . 152
19.2 Data matching across formats . . . 153
19.3 Lengths . . . 154
19.4 Positions in measures and moments . . . 154
19.5 Rests and skips . . . 154
19.6 Solo notes and rests . . . 154
19.7 Linear versus time-oriented representation . . . 155
19.8 Spanners . . . 156
19.9 Uplinks and sidelinks . . . 156
19.10 Sounding and displayed durations . . . 156
19.11 Printing descriptions . . . 157
19.12 Pitches . . . 158
19.13 Octaves . . . 160
19.14 Durations . . . 160
19.15 Alterations . . . 161
19.16 Accidentals . . . 161
19.17 Durations . . . 161
19.18 Tempos . . . 162
19.18.1 Tempos notes . . . 162
19.18.2 Tempos tuplets . . . 163
19.18.3 Tempos description . . . 164

19.19 Clefs . . . 165
19.20 Keys . . . 166
19.21 Time signatures . . . 166
19.22 MSR classes inheritance . . . 167
19.23 Books . . . 170
19.24 Scores . . . 170
19.25 Part groups . . . 171
19.26 Parts . . . 171
19.27 Staves . . . 171
19.28 Voice elements . . . 172
19.29 Voices . . . 173
19.30 Measures . . . 174
19.31 Repeats patterns and replicas . . . 175
19.32 Beat repeats . . . 176
19.33 Measure repeats . . . 177
19.34 Full measure rests . . . 178
19.35 Barlines . . . 179
19.36 Repeats . . . 179
19.37 Segments . . . 180
19.38 Notes and rests . . . 181
19.39 Articulations . . . 182
19.40 Ornaments . . . 182
19.41 Ties . . . 182
19.42 Dynamics . . . 182
19.43 Beams . . . 182
19.44 Slurs . . . 182
19.45 Grace notes groups . . . 182
19.46 Chords . . . 182
19.47 Tuplets . . . 182
19.48 Harmonies and figured bass similarities . . . 183
19.49 Harmonies . . . 185
19.50 Figured bass elements . . . 185
19.51 Lyrics . . . 185
19.52 MIDI . . . 186
20 MSR time-oriented represention . . . 187
20.1 Note events . . . 187
20.2 Simultaneous notes chunks . . . 188
20.3 Measures slices . . . 188
20.4 Measures slices sequences . . . 188
20.5 Building the measures slices . . . 189
20.5.1 Part measures slices . . . 189
20.5.2 Staff measures slices . . . 190
20.6 Solo notes and rests . . . 190
20.7 A measures slices example . . . 191
21 Path to voice . . . 192
22 LilyPond Scores Representation (LPSR) . . . 193
22.1 LPSR basic types . . . 193
22.2 Adapting LilyPond code generation to the target version number . . . 194
23 Braille Scores Representation (BSR) . . . 195
23.1 BSR basic types . . . 195
23.2 Representing cells . . . 195

24 MusicXML Scores Representation (MXSR) . . . 198
24.1 MusicXML elements and attributes . . . 198
24.2 The `xmlelement` and `xmlattribute` types . . . 199
24.3 Enumeration types for `xmlelement`'s `fType` . . . 200
24.4 Classes for the `xmlelements` . . . 201
24.5 `xmlelement` trees . . . 202
24.5.1 Creating `xmlelement` trees from textual data . . . 203
24.5.2 Printing `xmlelement` trees . . . 203
24.6 The `SXMLFile` type . . . 203

VII Passes **205**
25 The passes . . . 206
25.1 Translating MusicXML data to an MXSR format . . . 206
25.1.1 MusicXML coverage . . . 206
25.2 Translating an MXSR to an MSR . . . 206
25.3 Translating an MSR to an MXSR . . . 206
25.4 Translating an MSR to another MSR . . . 206
25.5 Translating an MSR to an LPSR . . . 206
25.6 Translating an LPSR to LilyPond code . . . 207
25.7 Translating an MSR to an BSR . . . 207
25.8 Translating a BSR to another BSR . . . 207
25.9 Translating an MXSRMain]MXSR to Guido . . . 207
26 LilyPond code generation . . . 208
26.1 Basic principle . . . 208
26.2 Generating Scheme functions in the LilyPond output . . . 209
27 Braille generation . . . 210
27.1 Basic principle . . . 210
27.2 Output files name and contents options . . . 211
27.3 Braille generators . . . 211
27.4 Writing braille cells . . . 212
28 MusicXMLMain]MusicXML generation . . . 214
28.1 Basic principle . . . 214
28.2 Creating an `xmlelement` . . . 214
28.3 Creating an `xmlelement` tree . . . 215
28.4 Browsing the visited MSR score . . . 216
28.5 Ancillary functions to create MXSRMain]MXSR data . . . 217
29 Guido code generation . . . 218
29.1 Basic principle . . . 218

VIII Generators **219**
30 The generators . . . 220
30.1 `MusicAndHarmonies` . . . 220
30.2 `Mikrokosmos3Wandering` . . . 220
30.3 `LilyPondIssue34` . . . 221

IX Converters **222**
31 The converters . . . 223
31.1 `xml2ly` . . . 223
31.2 `xml2brl` . . . 223
31.3 `xml2xml` . . . 224
31.4 `xml2gmn` . . . 224
31.5 `msdlconverter` . . . 224

X Interfaces **225**
32 Library interfaces . . . 226
33 Representations interfaces . . . 227
33.1 MSR interfaces . . . 227
33.2 LPSR interfaces . . . 227
33.3 MSDL interfaces . . . 227
34 Passes interfaces . . . 228
34.1 Translating MusicXML data to an MXSR . . . 229
34.2 Translating an MXSR to an MSR . . . 229
34.3 Translating an MSR to an MXSR . . . 229
34.4 Translating an MSR to another MSR . . . 229
34.5 Translating an MSR to an LPSR . . . 229
34.6 Translating an LPSR to LilyPond code . . . 229
34.7 Translating an MSR to an BSR . . . 229
34.8 Translating a BSR to another BSR . . . 229
34.9 Translating an MXSR to Guido . . . 229
35 Converters interfaces . . . 230

XI Selected topics **232**
36 Initializations . . . 233
36.1 Options and help initializations . . . 233
36.2 Representations initializations . . . 234
36.2.1 MSR initialization . . . 234
36.2.2 LPSR initialization . . . 235
36.2.3 BSR initialization . . . 235
36.3 Passes initializations . . . 235
36.4 Converters initializations . . . 235
37 The OAH atoms collection . . . 238
37.1 OAH macro atoms . . . 238
37.2 A OAH macro atom example . . . 240
37.3 LilyPond octave entry . . . 241
38 Measures handling . . . 243
38.1 Voices contents . . . 243
38.2 Voice elements . . . 243
38.3 Measure elements . . . 244
38.4 Appending measure elements to a measure . . . 244
38.5 Appending measures to a segment . . . 246
38.6 Appending measures to a voice . . . 248
38.7 Translating from MXSR to MSR . . . 250
38.8 Translating from MXSR to MSR . . . 251
38.9 Translating from MSR to MSR . . . 252
38.10 Translating from MSR to LPSR . . . 252
38.11 Translating from LPSR to LilyPond . . . 252
39 Positions in measures . . . 253
39.1 Determining positions in measures . . . 253
40 Finalizations . . . 254
40.1 Clones vs non-clones finalization . . . 254
40.2 The finalization methods . . . 255
40.3 Finalizing parts . . . 258
40.4 Finalizing staves . . . 259
40.5 Finalizing voices . . . 259
40.6 Finalizing repeats . . . 261
40.7 Finalizing measures . . . 261
40.7.1 Finalizing regular measures . . . 262
40.7.2 Finalizing harmonies measures . . . 265
40.7.3 Finalizing figured bass measures . . . 266

40.8 Determining positions in measures . . . 268
41 Tempos handling . . . 269
41.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 269
41.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 269
41.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 269
41.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 269
41.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 269
42 Positions in measures . . . 270
42.1 Determining positions in measures . . . 270
43 Notes handling . . . 271
43.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 271
43.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 271
43.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 271
43.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 271
43.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 271
44 Segments handling . . . 272
44.1 Segments creation . . . 272
44.1.1 Creating a new last segment for a voice . . . 273
44.1.2 Creating a new last segment for a voice from its first measure . . . 274
44.2 Appending measures to a segment . . . 275
44.3 Translating from MXSR to MSR . . . 276
44.4 Translating from MXSR to MSR . . . 276
44.5 Translating from MSR to MSR . . . 276
44.6 Translating from MSR to LPSR . . . 276
44.7 Translating from LPSR to LilyPond . . . 276
45 Beat repeats handling . . . 277
45.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 277
45.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 277
45.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 277
45.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 277
45.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 277
46 Measure repeats handling . . . 278
46.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 278
46.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 278
46.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 278
46.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 278
46.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 278
47 Full measure rests handling . . . 279
47.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 279
47.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 279
47.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 279
47.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 279
47.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 279
48 Repeats handling . . . 280
48.1 Translating repeats from MXSR to MSR . . . 280
48.2 Translating repeats from MXSR to MSR . . . 285
48.3 Translating repeats from MSR to MSR . . . 285
48.4 Translating repeats from MSR to LPSR . . . 285
48.5 Translating repeats from LPSR to LilyPond . . . 285
49 Voices handling . . . 286
49.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 286
49.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 286
49.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 286
49.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 286
49.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 286

50 Staves handling . . . 287
50.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 287
50.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 287
50.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 287
50.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 287
50.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 287
51 Parts handling . . . 288
51.1 Parts browsing . . . 288
51.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 289
51.3 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 289
51.4 Translating from MSR to MSR (src/passes/msr2msr/) . . . 289
51.5 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 289
51.6 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 289
52 Scores handling . . . 290
52.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 290
52.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 290
52.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 290
52.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 290
52.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 290
53 Books handling. . . . 291
53.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 291
53.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 291
53.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 291
53.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 291
53.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 291
54 Ornaments handling . . . 292
54.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 292
54.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 292
54.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 292
54.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 292
54.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 292
55 Ties handling . . . 293
55.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 293
55.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 293
55.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 293
55.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 293
55.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 293
56 Dynamics handling . . . 294
56.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 294
56.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 294
56.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 294
56.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 294
56.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 294
57 Beams handling . . . 295
57.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 295
57.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 295
57.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 295
57.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 295
57.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 295
58 Slurs handling . . . 296
58.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 296
58.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 296
58.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 296
58.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 296
58.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 296

59 Grace notes groups handling . . . 297
59.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 297
59.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 297
59.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 297
59.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 297
59.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 297
60 Chords handling . . . 298
60.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 298
60.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 298
60.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 298
60.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 298
60.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 298
61 Tuplets handling . . . 299
61.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 299
61.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 299
61.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 299
61.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 299
61.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 299
62 Harmonies handling . . . 300
62.1 Harmonies staves and voices . . . 301
62.2 Harmonies staves creation . . . 301
62.3 Harmonies creation . . . 302
62.4 Translating harmonies from MXSR to MSR . . . 303
62.5 Translating harmonies from MXSR to MSR . . . 304
62.5.1 First S_harmony visit . . . 304
62.5.2 Second S_harmony visit . . . 305
62.5.3 Attaching msrHarmony instances to notes . . . 306
62.5.4 Populating msrHarmony instances . . . 307
62.5.5 First S_harmony visit . . . 308
62.5.6 Inserting msrHarmony instances in the part harmonies voice . . . 309
62.6 Translating harmonies from MSR to MSR . . . 310
62.7 Translating harmonies from MSR to LPSR . . . 312
62.8 Translating harmonies from LPSR to LilyPond . . . 314
63 Figured bass elements handling . . . 316
63.1 Figured bass staves and voices . . . 317
63.2 Figured bass staves creation . . . 317
63.3 Figured bass elements creation . . . 318
63.4 Translating figured bass elements from MXSR to MSR . . . 319
63.4.1 First S_figured_bass visit . . . 320
63.4.2 Second S_figured_bass visit . . . 321
63.4.3 Attaching msrFiguredBassElement instances to notes . . . 322
63.4.4 Populating msrFiguredBassElement instances . . . 323
63.4.5 Inserting S_msrFiguredBassElement instances in the part figured bass voice . . . 325
63.5 Translating figured bass elements from MSR to MSR . . . 325
63.6 Translating figured bass elements from MSR to LPSR . . . 327
63.7 Translating figured bass elements from LPSR to LilyPond . . . 328
64 Lyrics handling . . . 330
64.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 330
64.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 330
64.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 330
64.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 330
64.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 330

65 Full measure rests handling . . . 331
65.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 331
65.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 331
65.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 331
65.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 331
65.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 331
66 MIDI handling . . . 332
66.1 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 332
66.2 Translating from MXSR to MSR (src/passes/mxsr2msr/) . . . 332
66.3 Translating from MSR to MSR (src/passes/msr2msr/) . . . 332
66.4 Translating from MSR to LPSR (src/passes/msr2lpsr/) . . . 332
66.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/) . . . 332

XII Music Scores Description Language (MSDL) 333

67 MSDL . . . 334
67.1 Main features of MSDL . . . 334
67.2 MSDL basic types . . . 334
67.3 What the MSDL converter does . . . 335
67.3.1 LilyPondMain]LilyPond generation . . . 335
67.3.2 BrailleMain]Braille generation . . . 335
67.3.3 MusicXMLMain]MusicXML generation . . . 336
67.3.4 GuidoMain]Guido generation . . . 336
67.4 A first example . . . 336
67.5 First example output from the MSDL converter . . . 337
67.5.1 LilyPondMain]LilyPond output . . . 337
67.5.2 BrailleMain]Braille output . . . 338
67.5.3 MusicXMLMain]MusicXML output . . . 338
67.5.4 GuidoMain]Guido output . . . 339
67.6 A more realistic example . . . 340
67.7 Multi-language support . . . 341
67.7.1 Multi-language messages handling . . . 341
67.7.2 Multi-language keywords handling . . . 341
67.8 Lexical analysis . . . 341
67.9 Music Scores Descriptions Representation (MSDR) . . . 341
67.10 Syntax and semantic analysis . . . 341
67.10.1 Error recovery . . . 341

XIII Debugging 342

68 Debugging . . . 343
68.1 Useful options . . . 343

XIV Indexes 344

List of Figures

1.1 The MusicFormats architecture 18

19.1 The solo rests problem 155
19.2 Humdrum-Scot keys 166
19.3 The MSR classes hierarchy 168
19.4 Three segments in a voice 181

30.1 Zoltán Kodály's Mikrokosmos III Wandering 220
30.2 The LilyPondIssue34 score 221

67.1 Braille for HelloWorld.xml with interpretation 338
67.2 Un Petit Air, par Jean Abou-Samra 341

Listings

13.1 libmusicxml/samples/countnotes.cppMain]countnotes.cppFiles]libmusicxml/samples!countnotes.cpp 57
13.2 Visiting <scaling/>Main]<scaling />MusicXML]scaling </> 58
13.3 msrDoubleTremolo::browseData (basevisitor* v) 59

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/files/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/files/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 1.1, [Architecture], page 18, 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 coude 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 1.1, [Architecture], page 18, Dom's work is essentially represented by the MusicXML, MXSRand 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 1.1, [Architecture], page 18, 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 lenghs, 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 prefered 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 Guidoand 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.

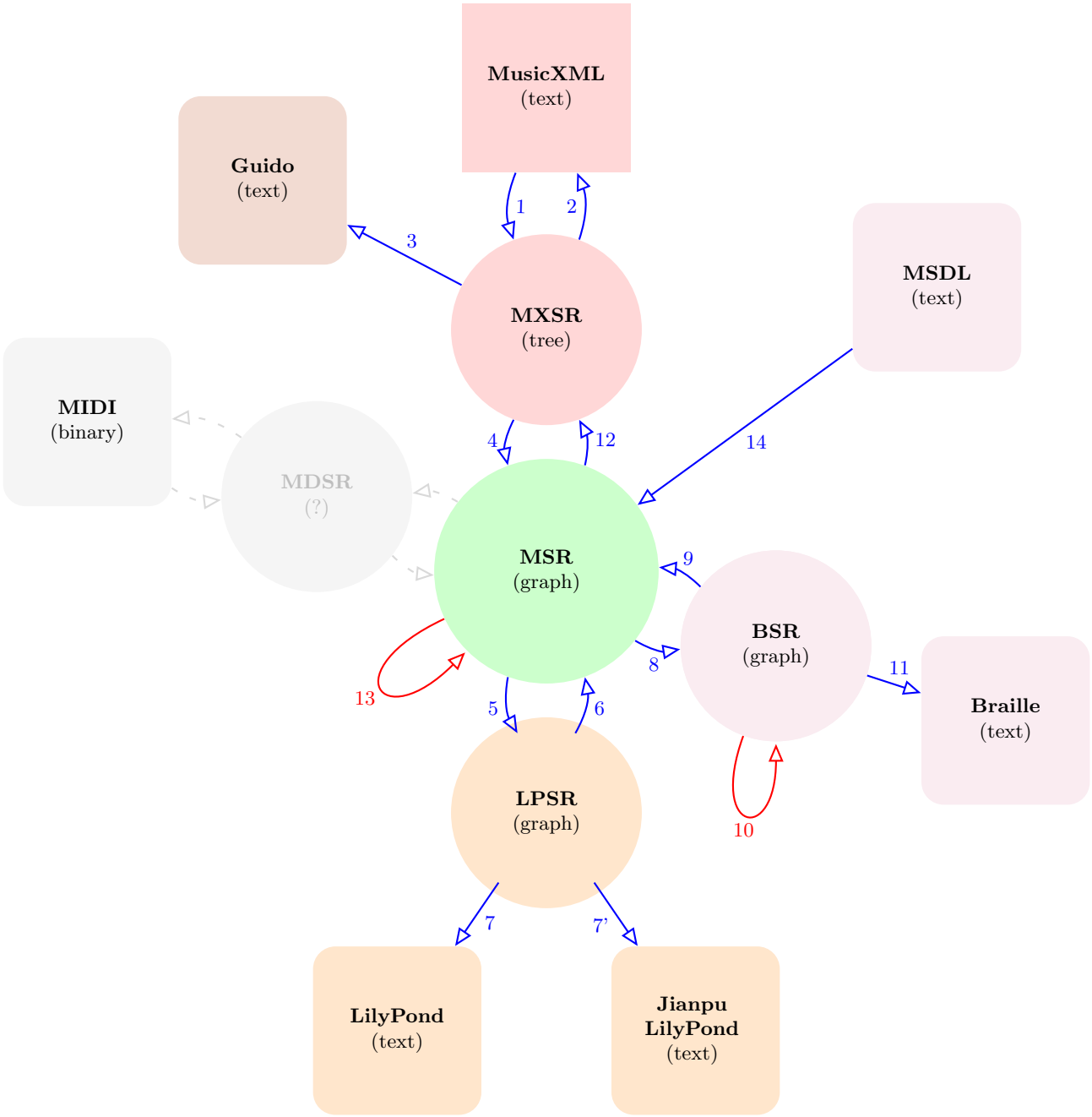
This document contains technical information about the internal working of the code added to MusicFormats by this author as their contribution to this great piece of software.

1.4 The GitHub repository

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

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

Figure 1.1: The MusicFormats architecture



Chapter 2

Documentation

The MusicFormats documentation is written in LaTeX, the pictures being created with the TikZ/PGF package, see https://www.bu.edu/math/files/2013/08/tikzpgfmanual.pdf.

The documentation/ directory contains:

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

The CommonLaTeXFiles/ directory contains LaTeX settings used by the various documentation files and the code for pictures:

```
jacquesmenu@macmini: ~/musicformats-git-dev/documentation > ls -sal CommonLaTeXFiles/*.tex
 48 -rw-r--r--@ 1 jacquesmenu  staff  21749 Feb  3 08:00 CommonLaTeXFiles/
    MusicFormatsArchitecturePicture.tex
 40 -rw-r--r--@ 1 jacquesmenu  staff  17187 Feb 28 08:25 CommonLaTeXFiles/
    MusicFormatsClassesHierarchyPicture.tex
104 -rw-r--r--@ 1 jacquesmenu  staff  49379 Feb 28 07:40 CommonLaTeXFiles/
    MusicFormatsLaTeXSettings.tex
 40 -rw-r--r--@ 1 jacquesmenu  staff  17113 Feb 28 08:27 CommonLaTeXFiles/
    MusicFormatsScoreViewPicture.tex
 40 -rw-r--r--@ 1 jacquesmenu  staff  17177 Aug  9  2021 CommonLaTeXFiles/
    MsrClassesHierarchyPicture.tex
```

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

Directory presentation/ 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).

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:

- file `MusicFormatsVersionNumber.h` and file `MusicFormatsVersionDate.h` files are used by the C++ code base;
- file `MusicFormatsVersionNumber.txt` and file `MusicFormatsVersionDate.txt` are used by the LaTeX source files

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

```
jacquesmenu@macmini: ~/musicformats-git-dev > build/scripts/SetMusicFormatsVersionNumber.
    bash 0.9.61
==> Writing MusicFormats version number 0.9.61 to MusicFormatsVersionNumber.txt

8 -rw-r--r--@ 1 jacquesmenu  staff  7 Feb 28 10:16:48 2022 MusicFormatsVersionNumber.txt
0.9.61

==> Writing MusicFormats version number 0.9.61 to MusicFormatsVersionNumber.h

8 -rw-r--r--@ 1 jacquesmenu  staff  45 Feb 28 10:16:48 2022 MusicFormatsVersionNumber.h
#define MUSICFORMATS_VERSION_NUMBER "0.9.61"
```

and:

```
jacquesmenu@macmini: ~/musicformats-git-dev > build/scripts/SetMusicFormatsVersionDate.
    bash "February 28, 2022"
==> Writing MusicFormats version date February 28, 2022 to MusicFormatsVersionDate.txt

8 -rw-r--r--@ 1 jacquesmenu  staff  18 Feb 28 10:17:36 2022 MusicFormatsVersionDate.txt
February 28, 2022

==> Writing MusicFormats version date February 28, 2022 to MusicFormatsVersionDate.h

8 -rw-r--r--@ 1 jacquesmenu  staff  54 Feb 28 10:17:36 2022 MusicFormatsVersionDate.h
#define MUSICFORMATS_VERSION_DATE "February 28, 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 LaTeX 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/MusicFormatsLaTeXSettings.tex`.

For example:

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

```
\newcommand{\musicXmlMarkup}[1]{%
{\tt <#1/>}\index[Main]{{\tt $<$#1 /$>$}}\index[MusicXML]{{\tt #1 $<$/$>$}}%
}
\newcommand{\musicXmlAttribute}[1]{%
{\tt "#1"}\index[Main]{{\tt $<$#1 /$>$}}\index[MusicXML]{{\tt #1 ""}}%
}
```

```
\newcommand{\Main}[1]{%
#1\index[Main]{#1}%
}
\newcommand{\MainName}[1]{%
\index[Main]{#1}%
}

\newcommand{\code}[1]{%
{\tt #1}\index[Main]{{\tt #1}}%
}
```

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

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

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

```
\newcommand{\fileName}[1]{%
{\tt #1}\index[Main]{{\tt #1}}\index[Files]{{\tt #1}}%
}
\newcommand{\fileNameBoth}[1]{%
{\textcolor{brown}{\tt *#1.h/.cpp}}\index[Main]{#1.h/.cpp@{{tt *#1.h/.cpp}}}\index[Files
    ]{#1.h/.cpp@{{tt *#1.h/.cpp}}}%
}
```

```
\newcommand{\msrToMsr}[1]{%
{\textcolor{brown}{\tt src/passes/msr2msr/#1}}%
}
\newcommand{\msrToMsrBoth}[1]{%
{\textcolor{brown}{\tt src/passes/msr2msr/#1.h/.cpp}}%
}
```

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

```
\newcommand{\methodName}[1]{%
{\tt #1~()}\index[Main]{{\tt #1}~()}\index[MethodsAndFields]{{\tt #1}~()}%
}
\newcommand{\starMethodName}[1]{%
{\tt *#1~()}\index[Main]{#1~()@{\tt *#1}}\index[MethodsAndFields]{*#1~()@{\tt *#1~()}}%
}
```

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

```
\newcommand{\file}[1]{%
file {\tt #1}\index[Main]{{\tt #1}}\index[Files]{{\tt #1}}%
}
\newcommand{\File}[1]{%
File {\tt #1}\index[Main]{{\tt #1}}\index[Files]{{\tt #1}}%
}

\newcommand{\fileName}[1]{%
{\tt #1}\index[Main]{{\tt #1}}\index[Files]{{\tt #1}}%
}
\newcommand{\fileNameBoth}[1]{%
{\textcolor{brown}{\tt *#1.h/.cpp}}\index[Main]{#1.h/.cpp@{{tt *#1.h/.cpp}}}\index[Files
    ]{#1.h/.cpp@{{tt *#1.h/.cpp}}}%
}
```

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

```
\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 is 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 MSDL 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++11 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++11 language is needed because MusicFormats uses <`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:

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

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

3.3 `cmake` configuration

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

3.4 Library Makefile

This Makefile is build/Makefile.

3.5 CLI amples Makefile

This Makefile is src/clisamples/Makefile.

3.6 Building MusicFormats in practise

Once in the local repository clone, just execute:

```
cd build
make
```

The resulting executables are in build/bin:

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

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

```
jacquesmenu@macmini: ~/musicformats-git-dev > ll build/lib
total 1888712
      0 drwxr-xr-x  10 jacquesmenu  staff        320 Sep 27 00:04:46 2021 ./
      0 drwxr-xr-x  11 jacquesmenu  staff        352 Aug  1 18:32:54 2021 ../
 104904 -rwxr-xr-x   1 jacquesmenu  staff   53707712 Sep 27 00:04:46 2021 libmusicxml2
    .3.2.0.dylib*
      0 lrwxr-xr-x   1 jacquesmenu  staff         24 Sep 27 00:04:45 2021 libmusicxml2.3.
    dylib@ -> libmusicxml2.3.2.0.dylib
1055040 -rw-r--r--   1 jacquesmenu  staff  532838416 Sep 27 00:04:41 2021 libmusicxml2.a
```

```
591776 -rw-r--r--   1 jacquesmenu  staff  302989312 Sep 21 09:05:55 2021 libmusicxml2.a.A93i4n
 57056 -rw-r--r--   1 jacquesmenu  staff   29212672 Sep 21 09:01:27 2021 libmusicxml2.a.KHrJT0
 39968 -rw-r--r--   1 jacquesmenu  staff   20463616 Sep 21 09:11:20 2021 libmusicxml2.a.gZfmqe
 39968 -rw-r--r--   1 jacquesmenu  staff   20463616 Sep 21 09:09:22 2021 libmusicxml2.a.tndUAV
    0 lrwxr-xr-x   1 jacquesmenu  staff         20 Sep 27 00:04:45 2021 libmusicxml2.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 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 supplide 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 generated 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, 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}` environment variable or its equivalent.

4.2 The doc folder

This folder contains MusicFormatsLaTeXSettings.tex, included by the various LaTeX documents whose code is in the respective folders, together with the PDF files:

```
jacquesmenu@macmini: ~/musicformats-git-dev/documentation > ll
total 56
 0 drwxr-xr-x    9 jacquesmenu  staff    288 Jan  4 17:23:41 2022 ./
 0 drwxr-xr-x   37 jacquesmenu  staff   1184 Dec 29 08:09:43 2021 ../
48 -rw-r--r--@   1 jacquesmenu  staff  22532 Jan  4 17:23:00 2022 .DS_Store
 8 -rw-r--r--    1 jacquesmenu  staff     42 Apr 22 15:48:42 2021 .gitignore
 0 drwxr-xr-x   38 jacquesmenu  staff   1216 Dec 26 08:49:24 2021 IntroductionToMusicXML/
 0 drwxr-xr-x  119 jacquesmenu  staff   3808 Jan  4 17:19:38 2022
   MusicFormatsMaintainanceGuide/
 0 drwxr-xr-x   49 jacquesmenu  staff   1568 Jan  4 17:04:10 2022 MusicFormatsUserGuide/
 0 drwxr-xr-x    9 jacquesmenu  staff    288 Jan  4 17:23:41 2022 CommonLaTeXFiles/
 0 drwxr-xr-x    6 jacquesmenu  staff    192 Apr 22 15:48:43 2021 presentation/
```

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

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/documentation/presentation > ll
total 504
  0 drwxr-xr-x  6 jacquesmenu  staff     192 Apr 22 15:48:43 2021 ./
  0 drwxr-xr-x  9 jacquesmenu  staff     288 Jan  4 17:23:41 2022 ../
  0 drwxr-xr-x  6 jacquesmenu  staff     192 Apr 22 15:48:43 2021 imgs/
 88 -rw-r--r--  1 jacquesmenu  staff   41534 Apr 22 15:48:43 2021 libmusicxml2.odg
392 -rw-r--r--  1 jacquesmenu  staff  200524 Apr 22 15:48:43 2021 libmusicxml2.pdf
 24 -rw-r--r--@ 1 jacquesmenu  staff   11003 Oct 15 18:48:11 2021 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, MEIand BMMLforat, and elements.bash scripts that compile the definitions into the C++ code files containing the classes:

```
jacquesmenu@macmini: ~/musicformats-git-dev/schemas > ll
total 2576
   0 drwxr-xr-x   9 jacquesmenu  staff      288 May 21 18:30:08 2021 ./
   0 drwxr-xr-x  22 jacquesmenu  staff      704 May 25 17:19:16 2021 ../
  16 -rw-r--r--   1 jacquesmenu  staff     6148 May 21 18:30:08 2021 .DS_Store
   0 drwxr-xr-x   4 jacquesmenu  staff      128 Apr 22 15:49:14 2021 BMML/
   0 drwxr-xr-x   5 jacquesmenu  staff      160 May 21 18:30:08 2021 MEI/
```

```
   8 -rw-r--r--   1 jacquesmenu  staff     2502 Apr 22 15:49:15 2021 Makefile
   0 drwxr-xr-x   6 jacquesmenu  staff      192 May 21 18:30:08 2021 MusicXML/
2552 -rw-r--r--   1 jacquesmenu  staff  1305905 Apr 22 15:49:13 2021 mei-CMN.rng
   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:

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > ll components
total 168
 0 drwxr-xr-x   7 jacquesmenu  staff    224 Oct 22 08:53:06 2021 ./
 0 drwxr-xr-x  19 jacquesmenu  staff    608 Oct 22 05:29:29 2021 ../
 8 -rw-r--r--@  1 jacquesmenu  staff   1106 Oct 22 09:16:21 2021 mfcComponents.h
96 -rw-r--r--@  1 jacquesmenu  staff  46691 Nov  8 12:59:57 2021 mfcBasicTypes.cpp
40 -rw-r--r--@  1 jacquesmenu  staff  20121 Nov  8 12:59:43 2021 mfcBasicTypes.h
16 -rw-r--r--@  1 jacquesmenu  staff   4950 Nov  8 12:59:08 2021 mfcLibraryComponent.
   cpp
 8 -rw-r--r--@  1 jacquesmenu  staff    605 Oct 22 10:36:30 2021 mfcLibraryComponent.
   h
```

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

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

- oah : object-oriented Options And Help support

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > ll oah
total 1456
  0 drwxr-xr-x  34 jacquesmenu  staff    1088 Nov 16 08:12:11 2021 ./
  0 drwxr-xr-x  20 jacquesmenu  staff     640 Nov 16 08:12:03 2021 ../
 48 -rw-r--r--@  1 jacquesmenu  staff   23743 Nov 16 08:16:55 2021 basicOah2manPage.
   cpp
 16 -rw-r--r--@  1 jacquesmenu  staff    5202 Nov 15 12:56:16 2021 basicOah2manPage.h
  8 -rw-r--r--@  1 jacquesmenu  staff     539 Jun  6 06:38:55 2021
   enableHarmoniesExtraOahIfDesired.h
  8 -rw-r--r--@  1 jacquesmenu  staff     526 Oct 11 11:56:29 2021
   enableTracingIfDesired.h
 72 -rw-r--r--@  1 jacquesmenu  staff   34280 Nov 16 08:16:55 2021 harmoniesExtraOah.
   cpp
 24 -rw-r--r--@  1 jacquesmenu  staff   11848 Nov 15 12:56:16 2021 harmoniesExtraOah.
   h
 16 -rw-r--r--@  1 jacquesmenu  staff    5154 Nov 16 08:16:55 2021
   harmoniesExtraOah2manPage.cpp
  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 oahOah.cpp
25  32 -rw-r--r--@  1 jacquesmenu  staff   13849 Nov 16 08:16:55 2021 oahOah.h
26   8 -rw-r--r--@  1 jacquesmenu  staff    1966 Nov 16 08:16:55 2021 oahOah2manPage.cpp
27   8 -rw-r--r--@  1 jacquesmenu  staff    1021 Nov 15 12:56:16 2021 oahOah2manPage.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 outputFileOah.cpp
33   8 -rw-r--r--@  1 jacquesmenu  staff    3593 Nov 15 12:56:16 2021 outputFileOah.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

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/validation > cat musicformatsversion.txt
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 comparisions 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 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 `xml2ly` `xml2xml` and MSDL converter.

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

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > look Interface
./formats/msr/msrInterface.cpp
./formats/msr/msrInterface.h
./formats/lpsr/lpsrInterface.cpp
./formats/lpsr/lpsrInterface.h
./formats/bsr/bsrInterface.h
./formats/bsr/bsrInterface.cpp
./passes/mxsr2musicxml/mxsr2musicxmlTranlatorInterface.h
./passes/mxsr2musicxml/mxsr2musicxmlTranlatorInterface.cpp
./passes/bsr2bsr/bsr2bsrFinalizerInterface.h
./passes/bsr2bsr/bsr2bsrFinalizerInterface.cpp
./passes/msr2mxsr/msr2mxsrInterface.cpp
./passes/msr2mxsr/msr2mxsrInterface.h
./passes/mxsr2msr/mxsr2msrSkeletonBuilderInterface.h
./passes/mxsr2msr/mxsr2msrTranslatorInterface.cpp
./passes/mxsr2msr/mxsr2msrTranslatorInterface.h
./passes/mxsr2msr/mxsr2msrSkeletonBuilderInterface.cpp
./passes/msr2msr/msr2msrInterface.h
./passes/msr2msr/msr2msrInterface.cpp
./passes/lpsr2lilypond/lpsr2lilypondInterface.h
./passes/lpsr2lilypond/lpsr2lilypondInterface.cpp
./passes/msr2lpsr/msr2lpsrInterface.cpp
./passes/msr2lpsr/msr2lpsrInterface.h
./passes/bsr2braille/bsr2brailleTranslatorInterface.h
./passes/bsr2braille/bsr2brailleTranslatorInterface.cpp
./passes/msr2bsr/msr2bsrInterface.h
./passes/msr2bsr/msr2bsrInterface.cpp
./passes/musicxml2mxsr/musicxml2mxsrInterface.h
./passes/musicxml2mxsr/musicxml2mxsrInterface.cpp
./passes/mxsr2guido/mxsr2guidoTranlatorInterface.h
./passes/mxsr2guido/mxsr2guidoTranlatorInterface.cpp
./converters/msr2guido/msr2guidoInterface.h
./converters/msr2guido/msr2guidoInterface.cpp
./converters/msr2braille/msr2brailleInterface.h
./converters/msr2braille/msr2brailleInterface.cpp
./converters/msdl2braille/msdl2brailleInterface.h
./converters/msdl2braille/msdl2brailleInterface.cpp
./converters/msdl2guido/msdl2guidoInterface.cpp
./converters/msdl2guido/msdl2guidoInterface.h
./converters/msdl2musicxml/msdl2musicxmlInterface.h
./converters/msdl2musicxml/msdl2musicxmlInterface.cpp
./converters/msdl2lilypond/msdl2lilypondInterface.h
./converters/msdl2lilypond/msdl2lilypondInterface.cpp
./converters/musicxml2braille/musicxml2brailleInterface.cpp
./converters/musicxml2braille/musicxml2brailleInterface.h
./converters/msr2lilypond/msr2lilypondInterface.cpp
./converters/msr2lilypond/msr2lilypondInterface.h
./converters/msr2musicxml/msr2musicxmlInterface.cpp
./converters/msr2musicxml/msr2musicxmlInterface.h
./converters/musicxml2musicxml/musicxml2musicxmlInterface.h
./converters/musicxml2musicxml/musicxml2musicxmlInterface.cpp
./converters/musicxml2lilypond/musicxml2lilypondInterface.h
./converters/musicxml2lilypond/musicxml2lilypondInterface.cpp
./converters/musicxml2guido/musicxml2guidoInterface.cpp
./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:

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/bsr/bsr.*
8 -rw-r--r--@ 1 jacquesmenu  staff   700 Jun  6 06:35:19 2021 formats/bsr/bsr.cpp
8 -rw-r--r--@ 1 jacquesmenu  staff  1206 Jun 18 10:04:45 2021 formats/bsr/bsr.h

jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/lpsr/lpsr.*
8 -rw-r--r--@ 1 jacquesmenu  staff   703 Jun  6 06:35:19 2021 formats/lpsr/lpsr.cpp
8 -rw-r--r--@ 1 jacquesmenu  staff  1004 Jun  6 06:35:19 2021 formats/lpsr/lpsr.h

jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/msdl/msdl.*
8 -rw-r--r--@ 1 jacquesmenu  staff  736 Jun  6 06:35:19 2021 formats/msdl/msdl.cpp
8 -rw-r--r--@ 1 jacquesmenu  staff  643 Jun  6 06:35:19 2021 formats/msdl/msdl.h

jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/msdr/msdr.*
8 -rw-r--r--@ 1 jacquesmenu  staff  709 Jun  6 06:35:19 2021 formats/msdr/msdr.cpp
8 -rw-r--r--@ 1 jacquesmenu  staff  531 Jun  6 06:35:19 2021 formats/msdr/msdr.h

jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/msr/msr.*
8 -rw-r--r--@ 1 jacquesmenu  staff   700 Jun  6 06:35:19 2021 formats/msr/msr.cpp
8 -rw-r--r--@ 1 jacquesmenu  staff  2410 Jun 20 09:58:38 2021 formats/msr/msr.h

jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/mxsr/mxsr.*
8 -rw-r--r--@ 1 jacquesmenu  staff  3292 Jun  6 06:35:19 2021 formats/mxsr/mxsr.cpp
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:

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

5.4 Passes

The passs are in src/passes:

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

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > look Interface
./representations/msr/msrInterface.cpp
./representations/msr/msrInterface.h
./representations/lpsr/lpsrInterface.cpp
./representations/lpsr/lpsrInterface.h
./representations/bsr/bsrInterface.h
./representations/bsr/bsrInterface.cpp
./passes/mxsr2musicxml/mxsr2musicxmlTranlatorInterface.h
./passes/mxsr2musicxml/mxsr2musicxmlTranlatorInterface.cpp
./passes/bsr2bsr/bsr2bsrFinalizerInterface.h
./passes/bsr2bsr/bsr2bsrFinalizerInterface.cpp
./passes/msr2mxsr/msr2mxsrInterface.cpp
./passes/msr2mxsr/msr2mxsrInterface.h
./passes/mxsr2msr/mxsr2msrSkeletonBuilderInterface.h
./passes/mxsr2msr/mxsr2msrTranslatorInterface.cpp
./passes/mxsr2msr/mxsr2msrTranslatorInterface.h
./passes/mxsr2msr/mxsr2msrSkeletonBuilderInterface.cpp
./passes/msr2msr/msr2msrInterface.h
./passes/msr2msr/msr2msrInterface.cpp
./passes/lpsr2lilypond/lpsr2lilypondInterface.h
./passes/lpsr2lilypond/lpsr2lilypondInterface.cpp
./passes/msr2lpsr/msr2lpsrInterface.cpp
./passes/msr2lpsr/msr2lpsrInterface.h
./passes/bsr2braille/bsr2brailleTranslatorInterface.h
./passes/bsr2braille/bsr2brailleTranslatorInterface.cpp
```

```
./passes/msr2bsr/msr2bsrInterface.h
./passes/msr2bsr/msr2bsrInterface.cpp
./passes/musicxml2mxsr/musicxml2mxsrInterface.h
./passes/musicxml2mxsr/musicxml2mxsrInterface.cpp
./passes/mxsr2guido/mxsr2guidoTranlatorInterface.h
./passes/mxsr2guido/mxsr2guidoTranlatorInterface.cpp
./converters/msr2guido/msr2guidoInterface.h
./converters/msr2guido/msr2guidoInterface.cpp
./converters/msr2braille/msr2brailleInterface.h
./converters/msr2braille/msr2brailleInterface.cpp
./converters/msdl2braille/msdl2brailleInterface.h
./converters/msdl2braille/msdl2brailleInterface.cpp
./converters/msdl2guido/msdl2guidoInterface.cpp
./converters/msdl2guido/msdl2guidoInterface.h
./converters/msdl2musicxml/msdl2musicxmlInterface.h
./converters/msdl2musicxml/msdl2musicxmlInterface.cpp
./converters/msdl2lilypond/msdl2lilypondInterface.h
./converters/msdl2lilypond/msdl2lilypondInterface.cpp
./converters/musicxml2braille/musicxml2brailleInterface.cpp
./converters/musicxml2braille/musicxml2brailleInterface.h
./converters/msr2lilypond/msr2lilypondInterface.cpp
./converters/msr2lilypond/msr2lilypondInterface.h
./converters/msr2musicxml/msr2musicxmlInterface.cpp
./converters/msr2musicxml/msr2musicxmlInterface.h
./converters/musicxml2musicxml/musicxml2musicxmlInterface.h
./converters/musicxml2musicxml/musicxml2musicxmlInterface.cpp
./converters/musicxml2lilypond/musicxml2lilypondInterface.h
./converters/musicxml2lilypond/musicxml2lilypondInterface.cpp
./converters/musicxml2guido/musicxml2guidoInterface.cpp
./converters/musicxml2guido/musicxml2guidoInterface.h
```

5.5 Generators

A generator is a multi-pass command line tool that creates an ouput 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 the bartok in various forms, depending on the options. It has been used to check th e MSR API's:

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

  For example:

```
jacquesmenu@macmini: ~ > Mikrokosmos3Wandering -lilypond -a
What LilyPondIssue34 does:

    This multi-pass generator creates a textual representation
    of the LilyPondIssue34 score.
    It basically performs 4 passes when generating LilyPond output output:

        Pass 1:  generate a first MSR for the LilyPondIssue34 score
        Pass 2:  converts the first MSR a second MSR;
```

```
        Pass 3:  converts the second MSR into a
                 LilyPond Score Representation (LPSR);
        Pass 4:  converts the LPSR to LilyPond code
                 and writes it to standard output.

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

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

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

```
jacquesmenu@macmini: ~ > LilyPondIssue34 -musicxml -a
What LilyPondIssue34 does:

    This multi-pass generator creates a textual representation
    of the LilyPondIssue34 score.
    It basically performs 4 passes when generating MusicXML output output:

        Pass 1:  generate a first MSR for the LilyPondIssue34 score
        Pass 2:  converts the first MSR a second MSR, to apply options;
        Pass 3:  converts the second MSR into an MusicXML tree;
        Pass 4:  converts the MusicXML tree to MusicXML code
                 and writes it to standard output.

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

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

5.6 Converters

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

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

```
class EXP mfServiceRunData : public smartable
{
  public:

    // creation
    // ------------------------------------------------------

    static SMARTP<mfServiceRunData> create (const string& serviceName);

    static SMARTP<mfServiceRunData> create (
                            const string& serviceName,
                            int           argc,
                            char*         argv[]);

    static SMARTP<mfServiceRunData> create (
                            const string&           serviceName,
                            mfOptionsAndArguments& optionsAndArguments);

  public:

    // constructors/destructor
    // ------------------------------------------------------

                            mfServiceRunData (const string& serviceName);

                            mfServiceRunData (
                              const string& serviceName,
                              int           argc,
                              char*         argv[]);

                            mfServiceRunData (
                              const string&           serviceName,
                              mfOptionsAndArguments& optionsAndArguments);

    virtual                 ~mfServiceRunData ();

  // .. .. ..

  private:

    // private fields
    // ------------------------------------------------------

    // service name
    string                  fServiceName;

    // conversion date
    string                  fRunDateFull;
    string                  fRunDateYYYYMMDD;

    // conversion command
    string                  fCommandAsSupplied;

    string                  fCommandWithLongOptionsNames;
    string                  fCommandWithShortOptionsNames;

    // options and arguments
    mfOptionsAndArguments
                            fOptionsAndArguments;

    // command line
    string                  fCommandLineAsSupplied;

```

```
    // input source
    string                  fInputSourceName;
};
```

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:

```
int main (int argc, char* argv[])
{
  // setup signals catching
  // ------------------------------------------------------

// JMI  catchSignals ();

  // the service name
  // ------------------------------------------------------

  string serviceName = argv [0];

  // create the global output and log indented streams
  // ------------------------------------------------------

  createTheGlobalIndentedOstreams (cout, cerr);

  // create the global run data
  // ------------------------------------------------------

  gGlobalServiceRunData =
    mfServiceRunData::create (serviceName);

  // ... ... ...
}
```

Then the various run data can be accessed easily:

```
  string
    inputSourceName =
      gGlobalServiceRunData->getInputSourceName ();
```

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

```
lpsrScore::lpsrScore (
  int                 inputLineNumber,
  S_msrScore          theMsrScore,
  S_mfcMultiComponent multiComponent)
    : lpsrElement (inputLineNumber)
{
  // ...

  fMultiComponent = multiComponent;

  // should the initial comments about the service and the options used
  // be generated?
  if (gGlobalLpsr2lilypondOahGroup->getXml2lyInfos ()) {
    // create the 'generated by' comment
    {
      stringstream s;

      s <<
        "Generated by " <<
        gGlobalOahOahGroup->getOahOahGroupServiceName () <<
        " " <<
        fMultiComponent->
          currentVersionNumberAndDateAsString () <<
        endl <<
```

```
25
26          "% on " <<
27          gGlobalServiceRunData->getRunDateFull () <<
28          endl <<
29
30          "% from ";
31
32        string inputSourceName =
33          gGlobalServiceRunData->getInputSourceName ();
34
35        if (inputSourceName == "-") {
36          s << "standard input";
37        }
38        else {
39          s << "\"" << inputSourceName << "\"";
40        }
41
42        fInputSourceNameComment =
43          lpsrComment::create (
44            inputLineNumber,
45            s.str (),
46            lpsrComment::kGapAfterwardsYes);
47      }
48    }
49
50    // ...
51  }
```

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:

```
jacquesmenu@macmini: ~/musicformats-git-dev > ll samples/
total 320
 0 drwxr-xr-x  16 jacquesmenu  staff    512 Jun 29 09:59:07 2021 ./
 0 drwxr-xr-x  28 jacquesmenu  staff    896 Jul  1 05:37:35 2021 ../
16 -rw-r--r--   1 jacquesmenu  staff   6148 May 26 08:20:55 2021 .DS_Store
 8 -rw-r--r--   1 jacquesmenu  staff    116 Apr 22 15:49:06 2021 .gitignore
40 -rw-r--r--@  1 jacquesmenu  staff  18344 Jun 29 11:05:18 2021 LilyPondIssue34.cpp
 8 -rw-r--r--@  1 jacquesmenu  staff   2101 Jun 29 10:00:56 2021 Makefile
40 -rw-r--r--@  1 jacquesmenu  staff  18362 Jun 29 11:05:10 2021 Mikrokosmos3Wandering.cpp
24 -rw-r--r--@  1 jacquesmenu  staff  10017 May 31 11:12:12 2021 MusicAndHarmonies.cpp
 8 -rw-r--r--@  1 jacquesmenu  staff   3117 May 31 11:17:27 2021 libMultipleInitsTest.cpp
48 -rw-r--r--@  1 jacquesmenu  staff  21459 Jun 29 11:05:02 2021 msdlconverter.cpp
 8 -rw-r--r--@  1 jacquesmenu  staff    898 May 31 11:15:59 2021 musicformatsversion.cpp
24 -rw-r--r--@  1 jacquesmenu  staff   8642 Jun 28 07:42:57 2021 xml2Any.cpp
24 -rw-r--r--@  1 jacquesmenu  staff  10085 Jul  1 06:22:13 2021 xml2brl.cpp
24 -rw-r--r--@  1 jacquesmenu  staff  10519 Jul  1 06:22:50 2021 xml2gmn.cpp
24 -rw-r--r--@  1 jacquesmenu  staff  10320 Jul  1 00:09:51 2021 xml2ly.cpp
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 maintainance tool to check that the MusicFormats library is not initialized more that 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 it 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.

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

```
//________________________________________________________________________
class EXP mfOahException: public mfException
{
  public:
    mfOahException (string const& exceptionDescription) throw ()
      : mfException (exceptionDescription)
    {}
};
typedef SMARTP<mfOahException> S_mfOahException;
```

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

```
    // convert the LPSR score to LilyPond code
    try {
      translateLpsrToLilypond (
        theLpsrScore,
        gGlobalMsrOahGroup,
        gGlobalLpsrOahGroup,
        passNumber,
        passDescription,
        lilypondStandardOutputStream);
    }
    catch (lpsr2lilypondException& e) {
      mfDisplayException (e, gOutputStream);
      return;
    }
    catch (std::exception& e) {
      mfDisplayException (e, gOutputStream);
      return;
    }
```

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

```
// comment the following definition if abort on internal errors is desired
// CAUTION: DON'T USE THIS IN PRODUCTION CODE,
// since that could kill a session on a \Web\ server, for example

#ifndef ABORT_TO_DEBUG_ERRORS
  #define ABORT_TO_DEBUG_ERRORS
#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/enableTracingIfDesired.h:

```
#ifndef ___enableTracingIfDesired___
#define ___enableTracingIfDesired___

#ifndef TRACING_IS_ENABLED
  // comment the following definition if no tracing is desired
  #define TRACING_IS_ENABLED
#endif

#endif
```

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

```
#include "enableTracingIfDesired.h"
#ifdef TRACING_IS_ENABLED
  #include "tracingOah.h"
#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:

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

```
            Write a trace of options and help handling to standard error.
      -toahd, -trace-oah-details
            Write a trace of options and help handling with more details to standard error
    .
    Score to voices           (-htstv, -help-trace-score-to-voices):
      -t<SHORT_NAME>, -trace-<LONG_NAME>
            Trace SHORT_NAME/LONG_NAME in books to voices.
      The 10 known SHORT_NAMEs are:
        book, scores, pgroups, pgroupsd, parts, staves, st, schanges,
        .
      The 10 known LONG_NAMEs are:
        -books, -scores, -part-groups, -part-groups-details,
        -parts, -staves, -staff-details, -staff-changes, -voices and
        -voices-details.
... ... ...
```

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:

```
void lpsr2lilypondTranslator::generateCodeForNoteRegularInMeasure (
  S_msrNote note)
{
  int inputLineNumber =
    note->getInputLineNumber ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceNotes ()) {
    stringstream s;

    s <<
      endl <<
      "% --> generating code for noteRegularInMeasure " <<
      note->asString () <<
      ", line " << inputLineNumber <<
      endl;

    gLogStream          << s.str ();
    fLilypondCodeStream << s.str ();
  }
#endif
```

8.4 Debugging traces handling

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

```
    gGlobalMsrOahGroup->getUseFilenameAsWorkTitle ()
```

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:

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

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

```
//--------------------------------------------------------------------------
int mfIndentedStreamBuf::sync ()
{
  // When we sync the stream with fOutputSteam:
  // 1) output the indentation then the buffer
  // 2) reset the buffer
  // 3) flush the actual output stream we are using.

  unsigned int strSize = str ().size ();

  // fetch the last non-space character in the buffer
  // caution: the '\n' is present as the last character!
  size_t found = str ().find_last_not_of (' ', strSize - 2);

  // this can be uncommented to see low level informations
  // fOutputSteam << "% strSize: " << strSize << ", found: " << found << '\n';

  // output the indenter
  fOutputSteam << fOutputIndenter;

  // output the buffer
  if (found == strSize - 3) {
    // don't output the trailing spaces, but output the end of line
    fOutputSteam << str ().substr (0, found + 1) << '\n';
  }
  else {
    // output the whole buffer
    fOutputSteam << str ();
  }

  // reset the buffer
  str ("");

  // flush the output stream
  fOutputSteam.flush ();

```

```
  return 0;
}
```

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 `osstream`, among them:

```
    void                  print (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 an 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 wether it writes the LilyPond code to standard output of to a file.

The parameters to this function are:

```
EXP void translateLpsrToLilypondWithHandler (
  const S_lpsrScore theLpsrScore,
  S_msrOahGroup     msrOpts,
  S_lpsrOahGroup    lpsrOpts,
  const string&     passNumber,
  const string&     passDescription,
  S_oahHandler      handler,
  ostream&          out,
  ostream&          err)
```

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

```
#define gIndenter mfOutputIndenter::gGlobalOStreamIndenter
```

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

```
    // create an indented output stream for the LilyPond code
    // to be written to outputFileStream
    mfIndentedOstream
      lilypondStandardOutputStream (
        out,
        gIndenter);

    // convert the LPSR score to LilyPond code
    try {
      translateLpsrToLilypond (
        theLpsrScore,
        gGlobalMsrOahGroup,
        gGlobalLpsrOahGroup,
        passNumber,
        passDescription,
        lilypondStandardOutputStream);
    }
```

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

```
     ofstream
      outputFileStream (
        outputFileName.c_str (),
        ofstream::out);

    // create an indented output stream for the LilyPond code
    // to be written to outputFileStream
    mfIndentedOstream
      lilypondFileOutputStream (
        outputFileStream,
        gIndenter);

    // convert the LPSR score to LilyPond code
    try {
      translateLpsrToLilypond (
        theLpsrScore,
        gGlobalMsrOahGroup,
        gGlobalLpsrOahGroup,
        passNumber,
        passDescription,
        lilypondFileOutputStream);
    }
```

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:

```
// useful shortcut macros
#define gIndenter mfOutputIndenter::gGlobalOStreamIndenter
#define gTab      mfOutputIndenter::gGlobalOStreamIndenter.getSpacer ()
```

A typical sequence to produce indented output is:

```
void msrTranspose::print (ostream& os) const
{
  const int fieldWidth = 22;

  os <<
    "Transpose" <<
    ", line " << fInputLineNumber <<
    endl;

  ++gIndenter;

  os << left <<
    setw (fieldWidth) <<
    "transposeDiatonic" << " = " << fTransposeDiatonic <<
    endl <<
    setw (fieldWidth) <<
    "transposeChromatic" << " = " << fTransposeChromatic <<
    endl <<
    setw (fieldWidth) <<
    "transposeOctaveChange" << " = " << fTransposeOctaveChange <<
    endl <<
    setw (fieldWidth) <<
```

```
    "transposeDouble" << " = " << fTransposeDouble <<
    endl << endl;

  --gIndenter;
}
```

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:

```
    void                print (ostream& os) const override;

    string              asString () const override;
    string              asStringShort () const override;
```

There are also more specific methods such as:

```
    void                printShort (ostream& os) const override;

    void                printSummary (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:

```
ostream& operator<< (ostream& os, const S_msrElement& elt)
{
  elt->print (os);
  return os;
}
```

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

```
void msrElement::print (ostream& os) const
{
  os << asString () << endl;
}
```

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:

```
string msrTranspose::asString () const
{
  stringstream s;

  s <<
    "[Transpose" <<
    ", diatonic = " << fTransposeDiatonic <<
    ", chromatic = " << fTransposeChromatic <<
    ", transposeOctaveChange = " << fTransposeOctaveChange <<
    ", transposeDouble = " << fTransposeDouble <<
    ", line " << fInputLineNumber <<
    "]";
```

```
13
14   return s.str ();
15 }
```

A typical sequence to produce indented output is:

```
1  void msrTranspose::print (ostream& os) const
2  {
3    const int fieldWidth = 22;
4
5    os <<
6      "Transpose" <<
7      ", line " << fInputLineNumber <<
8      endl;
9
10   ++gIndenter;
11
12   os << left <<
13     setw (fieldWidth) <<
14     "transposeDiatonic" << " = " << fTransposeDiatonic <<
15     endl <<
16     setw (fieldWidth) <<
17     "transposeChromatic" << " = " << fTransposeChromatic <<
18     endl <<
19     setw (fieldWidth) <<
20     "transposeOctaveChange" << " = " << fTransposeOctaveChange <<
21     endl <<
22     setw (fieldWidth) <<
23     "transposeDouble" << " = " << fTransposeDouble <<
24     endl << 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 argc/argv             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:

```
class EXP mfTimingItemsList {
  // ... ... ...

  public:

    // global variable for general use
    // ------------------------------------------------------

    static mfTimingItemsList         gGlobalTimingItemsList;

  public:

    // public services
    // ------------------------------------------------------

    // add an item
    void                    appendTimingItem (
                              string                      activity,
                              string                      description,
                              mfTimingItem::timingItemKind kind,
                              clock_t                     startClock,
                              clock_t                     endClock);

  // ... ... ...
}
```

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

```
S_lpsrScore translateMsrToLpsr (
  S_msrScore          originalMsrScore,
  S_msrOahGroup       msrOpts,
  S_lpsrOahGroup      lpsrOpts,
  string              passNumber,
  string              passDescription,
  S_mfcMultiComponent multiComponent)
{
  if (gGlobalLpsr2lilypondOahGroup->getNoLilypondCode ()) {
    gLogStream <<
      "Option '-nolpc, -no-lilypond-code' is set, no LPSR is created" <<
      endl;

    return nullptr;
  }

  // sanity check
  mfAssert (
    __FILE__, __LINE__,
    originalMsrScore != nullptr,
    "originalMsrScore is null");

  // start the clock
  clock_t startClock = clock ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalOahEarlyOptions.getEarlyTracePasses ()) {
    string separator =
      "%-------------------------------------------------------------";

    gLogStream <<
      endl <<
      separator <<
      endl <<
      gTab <<
      passNumber << ": " << passDescription <<
      endl <<
      separator <<
      endl;
  }
#endif

  // create an msr2lpsrTranslator
  msr2lpsrTranslator
    translator (
      originalMsrScore);

  // build the LPSR score
  S_lpsrScore
    resultingLpsr =
      translator.translateMsrToLpsr (
        originalMsrScore,
        multiComponent);

  clock_t endClock = clock ();

  // register time spent
  mfTimingItemsList::gGlobalTimingItemsList.appendTimingItem (
    passNumber,
    passDescription,
    mfTimingItem::kMandatory,
    startClock,
    endClock);
```

Part II

The two-phase visitors pattern

Chapter 13

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.

13.1 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:

```
class basevisitor
{
  public:
    virtual ~basevisitor() {}
};
```

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

```
void msrWords::acceptIn (basevisitor* v)
{
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "% ==> msrWords::acceptIn ()" <<
      endl;
  }

  if (visitor<S_msrWords>*
    p =
      dynamic_cast<visitor<S_msrWords>*> (v)) {
        S_msrWords elem = this;

        if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
          gLogStream <<
            "% ==> Launching msrWords::visitStart ()" <<
            endl;
        }
        p->visitStart (elem);
  }
}
```

13.2 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 seach for them in the code base. For example, in src/formats/msr/msrElements.h, there is:

```
//______________________________________________________________________________
template <typename T> class msrBrowser : public browser <T>
{
  public:

    msrBrowser (basevisitor* v) : fVisitor (v) {}

    virtual               ~msrBrowser () {}

  public:

    virtual void set (basevisitor* v) { fVisitor = v; }

    virtual void browse (T& t) {
      enter (t);

      t.browseData (fVisitor);

      leave (t);
    }

  protected:

    basevisitor*  fVisitor;

    virtual void enter (T& t) { t.acceptIn  (fVisitor); }
    virtual void leave (T& t) { t.acceptOut (fVisitor); }
};
```

13.3 A first example: counting notes in MusicMXL 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.

vVsitor method countnotes::visitStart only has to increment the notes count:

Listing 13.1: libmusicxml/samples/countnotes.cpp

```
class countnotes :
  public visitor<S_note>
{
  public:
    int fCount;

    countnotes() : fCount (0) {}

    virtual ~countnotes () {}

    void visitStart ( S_note& elt )   { fCount++; }
};
```

13.4 A more complex example

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

```
    <scaling>
      <millimeters>7</millimeters>
      <tenths>40</tenths>
    </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:

```
  mxsr2msrTranslator::visitStart ( S_scaling& elt)
    mxsr2msrTranslator::visitStart ( S_millimeters& elt )
    mxsr2msrTranslator::visitStart ( S_tenths& elt )
  mxsr2msrTranslator::visitEnd ( S_scaling& elt)
```

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

```
  mxsr2msrTranslator::visitStart ( S_scaling& elt)
    mxsr2msrTranslator::visitStart ( S_tenths& elt )
    mxsr2msrTranslator::visitStart ( S_millimeters& elt )
  mxsr2msrTranslator::visitEnd ( S_scaling& elt)
```

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

Listing 13.2: Visiting <scaling/>

```
void mxsr2msrTranslator::visitStart ( S_millimeters& elt )
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
    gLogStream <<
      "--> Start visiting S_millimeters" <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  fCurrentMillimeters = (float)(*elt);
}

void mxsr2msrTranslator::visitStart ( S_tenths& elt )
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
    gLogStream <<
      "--> Start visiting S_tenths" <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  fCurrentTenths = (float)(*elt);
}

void mxsr2msrTranslator::visitEnd ( S_scaling& elt)
{
  int inputLineNumber =
    elt->getInputLineNumber ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
    gLogStream <<
      "--> End visiting S_scaling" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  // create a scaling
  S_msrScaling
    scaling =
      msrScaling::create (
        inputLineNumber,
        fCurrentMillimeters,
        fCurrentTenths);

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceGeometry ()) {
    gLogStream <<
      "There are " << fCurrentTenths <<
      " tenths for " <<  fCurrentMillimeters <<
      endl;
  }
#endif

  // set the MSR score's scaling
  fMsrScore->
    setScaling (scaling);
}
```

13.5 Data browsing order

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

Listing 13.3: `msrDoubleTremolo::browseData (basevisitor* v)`

```
void msrDoubleTremolo::browseData (basevisitor* v)
{
  if (fDoubleTremoloFirstElement) {
    // browse the first element
    msrBrowser<msrElement> browser (v);
    browser.browse (*fDoubleTremoloFirstElement);
  }

  if (fDoubleTremoloSecondElement) {
    // browse the second element
    msrBrowser<msrElement> browser (v);
    browser.browse (*fDoubleTremoloSecondElement);
  }
}
```

Since this order is set in the `browsedata ()` 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.

13.6 Selectively inhibiting data browsing

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 `msrFullMeasureRests` contains class `msrMeasure` instances. class `msrScore` contains:

```
    // in <multiple-rest/>, the full measure rests are explicit,
    // whereas LilyPond only needs the number of full measure rests
    Bool                   fInhibitFullMeasureRestsBrowsing;

    void                   setInhibitFullMeasureRestsBrowsing ()
                               {
                                 fInhibitFullMeasureRestsBrowsing = true;
                               }

    Bool                   getInhibitFullMeasureRestsBrowsing () const
                               {
                                 return
                                   fInhibitFullMeasureRestsBrowsing;
                               };
```

Class `lpsr2lilypondTranslator` checks this setting:

```
void lpsr2lilypondTranslator::visitEnd (S_msrNote& elt)
{
  // ... ... ...

  if (fOnGoingFullMeasureRests) {
    switch (elt->getNoteKind ()) {
      case msrNoteKind::kNoteRestInMeasure:
        // don't handle full measure restss, that's done in visitEnd (
    S_msrFullMeasureRests&)
          if (elt->getNoteOccupiesAFullMeasure ()) {
            Bool inhibitFullMeasureRestsBrowsing =
```

```
              fVisitedLpsrScore->
                getMsrScore ()->
                  getInhibitFullMeasureRestsBrowsing ();

            if (inhibitFullMeasureRestsBrowsing) {
#ifdef TRACING_IS_ENABLED
              if (
                gGlobalTracingOahGroup->getTraceNotes ()
                  ||
                gGlobalTracingOahGroup->getTraceFullMeasureRests ()
              ) {
                gLogStream <<
                  "% ==> end visiting full measure rests is ignored" <<
                  endl;
              }
#endif

#ifdef TRACING_IS_ENABLED
              if (gGlobalTracingOahGroup->getTraceNotesDetails ()) {
                gLogStream <<
                  "% ==> returning from visitEnd (S_msrNote&)" <<
                  endl;
              }
#endif

              noteIsToBeIgnored = true;
            }
          }
        break;
  // ... ... ...
}
```

Another example is in the class `lpsr2lilypondTranslator` constructor:

```
lpsr2lilypondTranslator::lpsr2lilypondTranslator (
  S_lpsrScore     lpsrScore,
  S_msrOahGroup&  msrOpts,
  S_lpsrOahGroup& lpsrOpts,
  ostream&        lilypondCodeStream)
  : fLilypondCodeStream (
      lilypondCodeStream)
{
  fMsrOahGroup  = msrOpts;
  fLpsrOahGroup = lpsrOpts;

  // the LPSR score we're visiting
  fVisitedLpsrScore = lpsrScore;

  // inhibit the browsing of grace notes groups before,
  // since they are handled at the note level
  fVisitedLpsrScore->
    getMsrScore ()->
      setInhibitGraceNotesGroupsBeforeBrowsing ();

  // inhibit the browsing of grace notes groups after,
  // since they are handled at the note level
  fVisitedLpsrScore->
    getMsrScore ()->
      setInhibitGraceNotesGroupsAfterBrowsing ();
```

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

```
void msrNote::browseData (basevisitor* v)
{
  // browse the grace notes group before if any
  if (fNoteGraceNotesGroupBefore) {
```

```
    // fetch the score
    S_msrScore
      score =
        fetchNoteScoreUpLink ();

    if (score) {
      Bool
        inhibitGraceNotesGroupsBeforeBrowsing =
          score->
            getInhibitGraceNotesGroupsBeforeBrowsing ();

      if (inhibitGraceNotesGroupsBeforeBrowsing) {
#ifdef TRACING_IS_ENABLED
        if (
          gGlobalMsrOahGroup->getTraceMsrVisitors ()
            ||
          gGlobalTracingOahGroup->getTraceNotes ()
            ||
          gGlobalTracingOahGroup->getTraceGraceNotes ()
        ) {
          gLogStream <<
            "% ==> visiting grace notes groups before is inhibited" <<
            endl;
        }
#endif
      }
      else {
        // browse the grace notes group before
        msrBrowser<msrGraceNotesGroup> browser (v);
        browser.browse (*fNoteGraceNotesGroupBefore);
      }
    }
  }

  // ... ... ...
}
```

13.7 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:

```
  <defaults>
    <scaling>
      <millimeters>7.3</millimeters>
      <tenths>40</tenths>
    </scaling>
    <page-layout>
      <page-height>1534</page-height>
      <page-width>1151</page-width>
      <page-margins type="both">
        <left-margin>54.7945</left-margin>
        <right-margin>54.7945</right-margin>
        <top-margin>27.3973</top-margin>
        <bottom-margin>27.3973</bottom-margin>
      </page-margins>
    </page-layout>
    <system-layout>
      <system-margins>
        <left-margin>15</left-margin>
        <right-margin>0</right-margin>
```

```
      </system-margins>
      <system-distance>92.5</system-distance>
      <top-system-distance>27.5</top-system-distance>
    </system-layout>

  // ... ... ...

  <part id="P1">
    <measure number="1">
      <print>
        <system-layout>
          <system-margins>
            <left-margin>75.625</left-margin>
            <right-margin>0</right-margin>
          </system-margins>
          <top-system-distance>410.9375</top-system-distance>
        </system-layout>
        <staff-layout>
          <?DoletSibelius JustifyAllStaves=false?>
          <?DoletSibelius ExtraSpacesAbove=3?>
        </staff-layout>
        <measure-layout>
          <measure-distance>20</measure-distance>
        </measure-layout>
      </print>
```

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

It is assigned in:

```
void msr2mxsrTranslator::visitStart (S_msrPrintLayout& elt)
{
 // ... ... ...

 fOnGoingPrintLayout = true;
}

void msr2mxsrTranslator::visitEnd (S_msrPrintLayout& elt)
{
  // ... ... ...

  fOnGoingPrintLayout = false;
}
```

and checked for example in:

```
void msr2mxsrTranslator::visitStart (S_msrSystemLayout& elt)
{
  // ... ... ...

  // create a system layout element
  Sxmlelement
    systemLayoutElement =
      createMxmlElement (k_system_layout, "");

  if (fOnGoingPrintLayout) {
    // append it to the current print element
    fCurrentPrintElement->push (
      systemLayoutElement);
  }
  else {
    // don't append it at once to the score defaults element
    fScoreDefaultsSystemLayoutElement = systemLayoutElement;
  }
```

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 exemple in `mxsr2msrTranslator`, which uses `fCurrentPrintLayout` for this purpose:

```
void mxsr2msrTranslator::visitStart ( S_system_layout& elt )
{
  // ... ... ...

  // create the system layout
  fCurrentSystemLayout =
    msrSystemLayout::create (
      inputLineNumber);

  fOnGoingSystemLayout = true;
}

void mxsr2msrTranslator::visitEnd ( S_system_layout& elt )
{
  // ... ... ...

  if (fOnGoingPrint) {
    // set the current print layout's system layout
    fCurrentPrintLayout->
      setSystemLayout (
        fCurrentSystemLayout);
  }
  else {
    // set the MSR score system layout
    fMsrScore->
      setSystemLayout (
        fCurrentSystemLayout);
  }

  // forget about the current system layout
  fCurrentSystemLayout = nullptr;

  fOnGoingSystemLayout = false;
}
```

13.8 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:

```
    list<S_msrTuplet>         fOnGoingTupletsStack;
```

```
void lpsr2lilypondTranslator::visitStart (S_msrTuplet& elt)
{
  // ... ... ...

  if (fOnGoingTupletsStack.size ()) {
    // elt is a nested tuplet

    S_msrTuplet
      containingTuplet =
        fOnGoingTupletsStack.top ();

    // unapply containing tuplet factor,
```

```
    // i.e 3/2 inside 5/4 becomes 15/8 in MusicXML...
    elt->
      unapplySoundingFactorToTupletMembers (
        containingTuplet->
          getTupletFactor ());
  }

  // ... ... ...

  // push the tuplet on the tuplets stack
  fOnGoingTupletsStack (elt);

  // ... ... ...
}

void lpsr2lilypondTranslator::visitEnd (S_msrTuplet& elt)
{
  // ... ... ...

  // pop the tuplet from the tuplets stack
  fOnGoingTupletsStack ();

  // ... ... ...
}
```

13.9 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:

```
void msrPart::appendStaffDetailsToPart (
  S_msrStaffDetails staffDetails)
{
  // ... ... ...

  // register staff details in part
  fCurrentPartStaffDetails = staffDetails;

  // append staff details to registered staves
  for (
    map<int, S_msrStaff>::const_iterator i =
      getPartStaveNumbersToStavesMap.begin ();
    i != getPartStaveNumbersToStavesMap.end ();
    ++i
  ) {
    S_msrStaff
      staff = (*i).second;

    staff->
      appendStaffDetailsToStaff (
        staffDetails);
  } // for
}
```

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 cascadingthrough `printSlices ()` methods, see chapter 20, [MSR time-oriented represention], page 187.

Part III

Programming style and conventions

Chapter 14

Programming style and conventions

14.1 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 'globlal' files not related to any particular component, placed in src/mfutilities/ with an mf name prefix:

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public lpsrElement' *
formats/lpsr/lpsrStaves.h:29:class EXP lpsrNewStaffgroupBlock : public lpsrElement
formats/lpsr/lpsrStaves.h:87:class EXP lpsrNewStaffTuningBlock : public lpsrElement
formats/lpsr/lpsrStaves.h:150:class EXP lpsrNewStaffBlock : public lpsrElement
formats/lpsr/lpsrStaves.h:208:class EXP lpsrStaffBlock : public lpsrElement
formats/lpsr/lpsrVariables.h:27:class EXP lpsrVariableUseCommand : public lpsrElement
formats/lpsr/lpsrScores.h:35:class EXP lpsrScore : public lpsrElement
formats/lpsr/lpsrBarNumbers.h:26:class EXP lpsrBarNumberCheck : public lpsrElement
```

```
formats/lpsr/lpsrBarNumbers.h:85:class EXP lpsrBarCommand : public lpsrElement
formats/lpsr/lpsrLyrics.h:31:class EXP lpsrNewLyricsBlock : public lpsrElement
formats/lpsr/lpsrComments.h:25:class EXP lpsrComment : public lpsrElement
formats/lpsr/lpsrVoices.h:29:class EXP lpsrUseVoiceCommand : public lpsrElement
formats/lpsr/lpsrParts.h:27:class EXP lpsrPartBlock : public lpsrElement
formats/lpsr/lpsrPapers.h:32:class EXP lpsrPaper : public lpsrElement
formats/lpsr/lpsrPartGroups.h:29:class EXP lpsrPartGroupBlock : public lpsrElement
formats/lpsr/lpsrParallelMusic.h:28:class EXP lpsrParallelMusicBLock : public lpsrElement
formats/lpsr/lpsrLayouts.h:23:class EXP lpsrLayout : public lpsrElement
formats/lpsr/lpsrHeaders.h:27:class EXP lpsrHeader : public lpsrElement
formats/lpsr/lpsrScheme.h:29:class EXP lpsrSchemeVariable : public lpsrElement
formats/lpsr/lpsrScheme.h:140:class EXP lpsrSchemeFunction : public lpsrElement
formats/lpsr/lpsrBookBlockElements.h:35:class EXP lpsrBookBlockElement : public
    lpsrElement
formats/lpsr/lpsrBookBlockElements.h:237:class EXP lpsrBookBlock : public lpsrElement
formats/lpsr/lpsrContexts.h:30:class EXP lpsrContext : public lpsrElement
```

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

```
./formats/msdl/msdlBasicTypes.h
./formats/msdl/msdlBasicTypes.cpp
./formats/msr/msrBasicTypes.cpp
./formats/msr/msrBasicTypes.h
./formats/lpsr/lpsrBasicTypes.cpp
./formats/lpsr/lpsrBasicTypes.h
./formats/bsr/bsrBasicTypes.h
./formats/bsr/bsrBasicTypes.cpp
./oah/oahBasicTypes.h
./oah/oahBasicTypes.cpp
./formatsgeneration/msrGeneration/msrGenerationBasicTypes.cpp
./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`

14.2 Adding C++ files

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

When building MusicFormats with:

```
cd build
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 in 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`

14.3 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:

```
jacquesmenu@macmini: ~/musicformats-git-dev > type rmcache
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:

```
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:

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

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

14.4 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.

14.5 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++11'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`:

```
    for (auto e : mfEnumAll<msrHarmonyKind> ()) {
      // create the harmony structure
      S_msrHarmonyStructure
        harmonyStructure =
          msrHarmonyStructure::create (
            e);

      // register it in the map
      gGlobalHarmonyStructuresMap [e] =
        harmonyStructure;
    } // for
```

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

```
template< typename T >
class mfEnumAll
{
  public:

    class Iterator
    {
      public:

        Iterator (int value)
          : fIterationIndex (value)
            {}

        T operator* (void) const
            { return (T) fIterationIndex; }

        void operator++ (void)
            {  ++fIterationIndex; }

        Bool operator != (Iterator rhs)
            {  return fIterationIndex != rhs.fIterationIndex; }

      private:

        int fIterationIndex;
     };
};
```

14.6 Sanity checks

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

```
    // get voice to insert harmonies into
    S_msrVoice
      voiceToInsertHarmoniesInto =
        fCurrentPart->
          getPartHarmoniesVoice ();

    // sanity check
    mfAssert (
      __FILE__, __LINE__,
      voiceToInsertHarmoniesInto != nullptr,
      "voiceToInsertHarmoniesInto is null");
```

14.7 JMI comments

Comments containg `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.

14.8 Exported symbols

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

```
class EXP smartable {
  // ... ... ...
};
```

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

14.9 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`:

```
class EXP smartable {
  private:
    unsigned  refCount;
  public:
    //! gives the reference count of the object
    unsigned refs() const          { return refCount; }
    //! addReference increments the ref count and checks for refCount overflow
    void addReference()            { refCount++; assert(refCount != 0); }
    //! removeReference delete the object when refCount is zero
    void removeReference()       { if (--refCount == 0) delete this; }

  protected:
    smartable() : refCount(0) {}
    smartable(const smartable&): refCount(0) {}
    //! destructor checks for non-zero refCount
    virtual ~smartable()     { assert (refCount == 0); }
    smartable& operator=(const smartable&) { return *this; }
};

A smart pointer type is created with template class {\tt SMARTP}, for example:
template<class T> class SMARTP {
class EXP msrElement : public smartable
{
  // ... ... ...
};
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:

```
class EXP waeHandler : public smartable
{
  public:

    // creation
    // -------------------------------------------------------

    static SMARTP<waeHandler> create ();

  public:

    // constructors/destructor
    // -------------------------------------------------------

                            waeHandler ();

    virtual                 ~waeHandler ();

  public:

    // set and get
    // -------------------------------------------------------

  public:

    // public services
    // -------------------------------------------------------

  public:

    // print
    // -------------------------------------------------------

    string                  asString () const;

    void                    print (ostream& os) const;

  private:

    // private fields
    // -------------------------------------------------------
};
typedef SMARTP<waeHandler> S_waeHandler;
EXP ostream& operator<< (ostream& os, const S_waeHandler& elt);
```

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

```
S_waeHandler waeHandler::create ()
{
  waeHandler* o =
    new waeHandler ();
  assert (o != nullptr);
  return o;
}
```

14.10 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 ad 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:

```
typedef SMARTP<msrHarmonyDegree> S_msrHarmonyDegree;
EXP ostream& operator<< (ostream& os, const S_msrHarmonyDegree& elt);
```

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

14.11 #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.

14.12 Identifiers naming conventions

The following rules apply:

- all enumeration type names describing variants in classes end in 'Kind';
- all enumeration constants start with 'k';
- 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 defined with #define because creating global constants variables and accessing them would be too cumbersome, such as:

```
    \#define msrPart::K_PART_HARMONIES_STAFF_NUMBER       10
    \#define msrPart::K_PART_HARMONIES_VOICE_NUMBER       11

    \#define msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER  20
    \#define msrPart::K_PART_FIGURED_BASS_VOICE_NUMBER  21
```

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

```
    map<string, string>   fPartsRenamingMap;
```

or

```
                          fStringToDalSegnoKindMapVariable;
```

or

```
    map<string, Sxmlelement>  fPartMeasureNumbersToElementsMap;
```

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

```
    // creation from MusicXML
    // ------------------------------------------------------

    static SMARTP<msrHarmonyDegree> create (
                            int                       inputLineNumber,
                            int                       harmonyDegreeValue,
                            msrAlterationKind         harmonyDegreeAlterationKind,
                            msrHarmonyDegreeTypeKind harmonyDegreeTypeKind);

  protected:

    // constructors/destructor
    // ------------------------------------------------------

                          msrHarmonyDegree (
                            int                       inputLineNumber,
                            int                       harmonyDegreeValue,
                            msrAlterationKind         harmonyDegreeAlterationKind,
                            msrHarmonyDegreeTypeKind harmonyDegreeTypeKind);

    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.

14.13 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 exception classes are derived from , that contains:

```
class EXP mfException: public exception
{
  public:

    // constructors/destructor
    // ------------------------------------------------------

                           mfException (
                             string const& exceptionDescription = "",
                             int           exceptionNumber = 0,
                             int           exceptionLevel = 0
                           ) throw ()
                             : fExceptionDescription (
                                 "mfException: " + exceptionDescription),
                               fExceptionNumber (exceptionNumber),
                               fExceptionLevel (exceptionLevel)
                           {}

  // ... ... ...

  private:

    string                 fExceptionDescription;

    int                    fExceptionNumber;
    int                    fExceptionLevel;
};
```

An example of exception is:

```
class EXP mxsr2msrException: public mfException
{
  public:
    mxsr2msrException (string const& exceptionDescription) throw ()
      : mfException (exceptionDescription)
    {}
};
typedef SMARTP<musicxmlException> S_musicxmlException;
```

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

```
void oahAtomExpectingAValue::applyElement (ostream& os)
{
  stringstream s;

  s <<
    "Applying atom expecting a value '" <<
    fetchNames () <<
    "' without a value";

  oahInternalError (s.str ());
```

```
}
```

and:

```
    case msrPedal::k_NoPedalType:
      {
        // should not occur

        stringstream s;

        s <<
          "msrPedal '" <<
          elt->asShortString () <<
          "' has no pedal type";

        msrInternalError (
          gGlobalServiceRunData->getInputSourceName (),
          inputLineNumber,
          __FILE__, __LINE__,
          s.str ());
      }
      break;
```

Another one is:

```
void mxsr2msrTranslator::visitEnd ( S_accordion_registration& elt )
{
  int inputLineNumber =
    elt->getInputLineNumber ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
    gLogStream <<
      "--> End visiting S_accordion_registration" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  // An accordion-registration element needs to have
  // at least one of the child elements present

  if (fCurrentAccordionNumbersCounter == 0) {
    musicxmlWarning (
      gGlobalServiceRunData->getInputSourceName (),
      inputLineNumber,
      "accordion-registration has 0 child element, ignoring it");
  }

  else {
    // create the accordion registration
    S_msrAccordionRegistration
      accordionRegistration =
        msrAccordionRegistration::create (
          inputLineNumber,
          fCurrentAccordionHigh,
          fCurrentAccordionMiddle,
          fCurrentAccordionLow);

    // append it to the current part
    fCurrentPart->
      appendAccordionRegistrationToPart (
        accordionRegistration);
  }
}
```

14.14 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:

```
__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:

```
class __declspec(dllexport) CExampleExport : public CObject
{ ... 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:

```
#ifndef __exports__
#define __exports__

#if defined(WIN32) // && !defined (GCC)

# ifdef MSVC
#  pragma warning (disable : 4267)
#  pragma warning (disable : 4275)
#  pragma warning (disable : 4251)
#  pragma warning (disable : 4786)
#  pragma warning (disable : 4251)
#  pragma warning (disable : 4275)
# endif

# ifdef LIBMUSICXML_EXPORTS
#  define EXP __declspec(dllexport)

# elif defined(LIBMUSICXML_STATIC)
#  define EXP

# else
#  define EXP __declspec(dllimport)
# endif

#else

# ifdef LIBMUSICXML_EXPORTS
#  define EXP __attribute__ ((visibility("default")))
# else
#  define EXP
# endif

#endif

#endif
```

14.15 Dynamic type checking

Enumeration type `s` 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:

```
  // handle the option
  if (
    // options group?
    S_oahGroup
      group =
        dynamic_cast<oahGroup*>(&(*element))
  ) {
    registerOahElementUse (
      group, optionNameUsed, ""); // "===group==="); // JMI to debug
  }

  else if (
    // options subgroup?
    S_oahSubGroup
      subGroup =
        dynamic_cast<oahSubGroup*>(&(*element))
  ) {
    registerOahElementUse (
      subGroup, optionNameUsed, ""); // "===subGroup==="); // JMI to debug
  }
```

14.16 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:

```
    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:

```
msrElement::msrElement (
  int inputLineNumber)
{
  fInputLineNumber = inputLineNumber;
}
```

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

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

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

```
S_msrStaff Mikrokosmos3WanderingGenerator::createStaffInPart (
  int staffNumber, S_msrPart part)
{
  // create the staff
  S_msrStaff
    staff =
      msrStaff::create (
        __LINE__, msrStaffKind::kStaffKindRegular, staffNumber, part);

  // append it to the part
  part ->
    addStaffToPartCloneByItsNumber ( // JMI NOT clone???
      staff);

  return staff;
}
```

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

```
  line <number>
```

with a single space before the number, as in:

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

This helps locating such occurrences in the debug process.

14.17 Static declarations

They are used for:

- classes methods such as method `msrTie::create ()`, method `msrTempo::createTempoPerMinute ()` and method `msrTemp::tempoKindAsString ()`;
- classes constant fields such as constant `msrStanza::K_NO_STANZA_NUMBER`, to be prefered 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 `gGlobalOStreamIndenter` and global variable `gGlobalTimingItem` that would be too cumbersome to pass to each and every method or function that uses them.

14.18 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:

```
void initializeBSR ()
{
  // protect library against multiple initializations
  static Bool pPrivateThisMethodHasBeenRun (false);

  if (! pPrivateThisMethodHasBeenRun) {
    initializeBsrBasicTypes ();

    pPrivateThisMethodHasBeenRun = true;
  }
}
```

14.19 Enumeration types

All enumeration type `s` use the C++11 '`enum class`' feature, such as:

```
enum class msrSlurTypeKind {
  k_NoSlur,

  kSlurTypeRegularStart, kSlurTypePhrasingStart,
  kSlurTypeContinue,
  kSlurTypeRegularStop, kSlurTypePhrasingStop
};
```

This prevents enumeration constants name conflicts across enumeration type `s`, 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_No*` are used to indicate a value that *has not been set yet*. There are always the first on in the corresponding enumeration, to benefit from the the C++11 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:

```
    // repeat winged
    enum msrBarLineRepeatWingedKind {
      kBarLineRepeatWingedNone,

      kBarLineRepeatWingedStraight, kBarLineRepeatWingedCurved,
      kBarLineRepeatWingedDoubleStraight, kBarLineRepeatWingedDoubleCurved
    };
```

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

```
void mxsr2msrTranslator::visitStart ( S_repeat& elt )
{
    // ... ... ...

    string winged = elt->getAttributeValue ("winged");

    fCurrentBarLineRepeatWingedKind =
      msrBarLine::kBarLineRepeatWingedNone; // default value

    if (winged.size ()) {
      if       (winged == "none") {
        fCurrentBarLineRepeatWingedKind =
          msrBarLine::kBarLineRepeatWingedNone;
      }
```

```
      else if (winged == "straight") {
        fCurrentBarLineRepeatWingedKind =
          msrBarLine::kBarLineRepeatWingedStraight;
      }
      else  if (winged == "curved") {
        fCurrentBarLineRepeatWingedKind =
          msrBarLine::kBarLineRepeatWingedCurved;
      }
      else  if (winged == "double-straight") {
        fCurrentBarLineRepeatWingedKind =
          msrBarLine::kBarLineRepeatWingedDoubleStraight;
      }
      else  if (winged == "double-curved") {
        fCurrentBarLineRepeatWingedKind =
          msrBarLine::kBarLineRepeatWingedDoubleCurved;
      }
      else {
        stringstream s;

        s <<
          "repeat winged \"" << winged <<
          "\" is unknown";

        musicxmlError (
          gGlobalServiceRunData->getInputSourceName (),
          inputLineNumber,
          __FILE__, __LINE__,
          s.str ());
      }
    }

    // ... ... ...
```

14.20 yes/no enumerations types

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

```
    void                    setCombinedBooleanVariables (Bool value);
```

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

This is why we use enum classes such as:

```
enum class msrVoiceCreateInitialLastSegmentKind {
  kCreateInitialLastSegmentYes,
  kCreateInitialLastSegmentNo
};
```

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

```
  fPartHarmoniesVoice =
    msrVoice::create (
      inputLineNumber,
      msrVoiceKind::kVoiceKindHarmonies,
      partHarmoniesVoiceNumber,
      msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes,
      fPartHarmoniesStaff);
```

14.21 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 developper'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 developper, through constructors. This also avoids in particular long sequences of initializations in the passes constructors.

14.22 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:

```
// harmonies
//______________________________________________________________________________
enum class msrHarmonyKind {
  k_NoHarmony,

  // MusicXML harmonies

  kHarmonyMajor, kHarmonyMinor,
  kHarmonyAugmented, kHarmonyDiminished,

  // ... ... ...

  // other

  kHarmonyOther,

  kHarmonyNone,

  // aliases
  // ------------------------------------

  AllFirst = k_NoHarmony,
  AllLast  = kHarmonyNone,

  // ... ... ...
};
```

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

```
//______________________________________________________________________________
/*
  https://stackoverflow.com/questions/8498300/allow-for-range-based-for-with-enum-classes
*/
template< typename T >
class mfEnumAll
{
  public:

    class Iterator
    {
```

```
      public:

        Iterator (int value)
          : fIterationIndex (value)
            {}

        T operator* (void) const
            { return (T) fIterationIndex; }

        void operator++ (void)
            {  ++fIterationIndex; }

        Bool operator != (Iterator rhs)
            {  return fIterationIndex != rhs.fIterationIndex; }

      private:

        int fIterationIndex;
    };
};

template< typename T >
typename mfEnumAll<T>::Iterator begin (mfEnumAll<T>)
{
  return typename mfEnumAll<T>::Iterator ((int)T::AllFirst);
}

template< typename T >
typename mfEnumAll<T>::Iterator end (mfEnumAll<T>)
{
  return typename mfEnumAll<T>::Iterator (((int)T::AllLast) + 1);
}
```

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

```
void initializeHarmonyStructuresMap ()
{
  // protect library against multiple initializations
  static Bool pPrivateThisMethodHasBeenRun (false);

  if (! pPrivateThisMethodHasBeenRun) {
    for (auto e : mfEnumAll<msrHarmonyKind> ()) {
      // create the harmony structure
      S_msrHarmonyStructure
        harmonyStructure =
          msrHarmonyStructure::create (
            e);

      // register it in the map
      gGlobalHarmonyStructuresMap [e] =
        harmonyStructure;
    } // for

    pPrivateThisMethodHasBeenRun = true;
  }
}
```

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

```
    for (auto e : EnumNonSeparators<msdlTokenKind> ()) {
      string
        nonSeparatorTokenAsMsdlString  =
          msdlTokenKindAsMsdlString (
```

```
            e,
            keywordsLanguageKind);

      // ... ... ...
    } // for
```

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

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

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

```
void msrHarmonyStructure::printAllHarmoniesStructures (ostream& os)
{
  os <<
    "All the known harmonies structures are:" <<
    endl << endl;

  ++gIndenter;

  for (auto e : EnumTrueHarmonies<msrHarmonyKind> ()) {
    // create the harmony intervals
    S_msrHarmonyStructure
      harmonyStructure =
        msrHarmonyStructure::create (
          e);

    // print it
    os <<
      harmonyStructure <<
      endl;
  } // for

  --gIndenter;
}
```

14.23 Rational numbers

MusicFormats uses rationals for notes sounding and display whole notes and positions in measures, among others. Class `rational` is defined by `libmusicxml2` in libmusicxml/src/lib!rational.h/.cpp:

```
class EXP rational {

   private:

        long int fNumerator;
        long int fDenominator;

        // Used by rationalise()
        long int gcd(long int a, long int b);

    public:

        rational(long int num = 0, long int denom = 1);
        rational(const rational& d);
        rational(const string &str);
```

```
16
17   // ... ... ...
18 };
```

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

14.24 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.

14.25 `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 justs 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       string             tempoPerMinute,
13       msrTempoParenthesizedKind
14                          tempoParenthesizedKind,
15       msrPlacementKind   tempoPlacementKind);
16
17     static SMARTP<msrTempo> createTempoBeatUnitEquivalent (
18       int                inputLineNumber,
```

```
      msrDottedDuration tempoBeatUnit,
      msrDottedDuration tempoEquivalentBeatUnit,
      msrTempoParenthesizedKind
                        tempoParenthesizedKind,
      msrPlacementKind  tempoPlacementKind);

    static SMARTP<msrTempo> createTempoNotesRelationship (
      int               inputLineNumber,
      S_msrTempoNotesRelationshipshipElements
                        tempoNotesRelationshipLeftElements,
      msrTempoNotesRelationshipKind
                        tempoNotesRelationshipKind,
      S_msrTempoNotesRelationshipshipElements
                        tempoNotesRelationshipRightElements,
      msrTempoParenthesizedKind
                        tempoParenthesizedKind,
      msrPlacementKind  tempoPlacementKind);

    // ... ... ...
};
```

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

```
class EXP msrKey : public msrMeasureElement
{
    // ... ... ...

    static SMARTP<msrKey> createTraditional (
                            int                       inputLineNumber,
                            msrQuarterTonesPitchKind  keyTonicPitchKind,
                            msrModeKind               modeKind,
                            int                       keyCancel);

    static SMARTP<msrKey> createHumdrumScot (
      int                    inputLineNumber);

    // ... ... ...
};
```

14.26 get*(), set*() and fetch*() methods

As is usual, classes privates 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:

```
    // set and get
    // ------------------------------------------------------

    void                    setTieKind (msrTieKind tieKind)
                                { fTieKind = tieKind; }

    msrTieKind              getTieKind () const
                                { return fTieKind; }

    void                    setTiePlacementKind (msrPlacementKind placementKind)
                                { fTiePlacementKind = placementKind; }

    msrPlacementKind        getTiePlacementKind () const
                                { 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:

```
S_msrStaff msrSegment::fetchSegmentStaffUpLink () const
{
  S_msrStaff result;

  if (fSegmentVoiceUpLink) {
    result =
      fSegmentVoiceUpLink->
        getVoiceStaffUpLink ();
  }

  return result;
}
```

14.27 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:

```
msrPart::msrPart (
  int            inputLineNumber,
  string         partID,
  S_msrPartGroup partPartGroupUpLink)
    : msrPartGroupElement (inputLineNumber)
{
  // replace spaces in partID to set fPartID
  for_each (
    partID.begin (),
    partID.end (),
    mfStringSpaceReplacer (fPartID, '_'));

/* JMI
  // sanity check
  mfAssert (
    __FILE__, __LINE__,
    partPartGroupUpLink != nullptr,
    "partPartGroupUpLink is null");
    */

  // set part number
  fPartAbsoluteNumber = ++gPartsCounter;

  // set part's part group upLink
  fPartPartGroupUpLink = partPartGroupUpLink;

  // do other initializations
  initializePart ();
}
```

Some finalize*() () methods exist.

14.28 *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:

```
// placement
//______________________________________________________________________________
enum class msrPlacementKind {
  k_NoPlacement,

  kPlacementAbove, kPlacementBelow
};
```

```
// placement
//______________________________________________________________________________
msrPlacementKind msrPlacementKindFromString (
  int    inputLineNumber,
  string placementString)
{
  msrPlacementKind result = msrPlacementKind::k_NoPlacement; // default value

  if      (placementString == "above")
    result = msrPlacementKind::kPlacementAbove;
  else if (placementString == "below")
    result = msrPlacementKind::kPlacementBelow;
  else {
    if (placementString.size ()) {
      stringstream s;

      s <<
        "placement \"" << placementString <<
        "\" should be 'above' or 'below'";

      musicxmlError (
        gGlobalServiceRunData->getInputSourceName (),
        inputLineNumber,
        __FILE__, __LINE__,
        s.str ());
    }
  }

  return result;
}
```

```
string msrPlacementKindAsString (
  msrPlacementKind placementKind)
{
  string result;

  switch (placementKind) {
    case msrPlacementKind::k_NoPlacement:
      result = "noPlacement";
      break;
    case msrPlacementKind::kPlacementAbove:
      result = "placementAbove";
      break;
    case msrPlacementKind::kPlacementBelow:
      result = "placementBelow";
      break;
  } // switch

  return result;
}
```

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

14.29 translate*() methods and convert*() functions

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

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

```
//________________________________________________________________________
class EXP msr2msrTranslator :

  // MSR score

  public visitor<S_msrScore>,

  // ... ... ...

{
  public:

                          msr2msrTranslator ();

    virtual               ~msr2msrTranslator ();

    S_msrScore            translateMsrToMsr (
                            S_msrScore theMsrScore);
  // ... ... ...

};
```

```
S_msrScore msr2msrTranslator::translateMsrToMsr (
  S_msrScore theMsrScore)
{
  // sanity check
  mfAssert (
    __FILE__, __LINE__,
    theMsrScore != nullptr,
    "theMsrScore is null");

  // the MSR score we're visiting
  fVisitedMsrScore = theMsrScore;

  // create the resulting MSR score
  fResultingNewMsrScore =
    msrScore::create (
      K_NO_INPUT_LINE_NUMBER,
      "msrScore::create()");

  // create a msrScore browser
  msrBrowser<msrScore> browser (this);

  // browse the visited score with the browser
  browser.browse (*fVisitedMsrScore);

  // forget about the visited MSR score
  fVisitedMsrScore = nullptr;

  return fResultingNewMsrScore;
}
```

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

```
S_msrScore translateMsrToMsr (
  S_msrScore         originalMsrScore,
  S_msrOahGroup      msrOpts,
  S_msr2msrOahGroup  msr2msrOpts,
  const string&      passNumber,
```

```
  const string&     passDescription)
{
  // ... ... ...

  // the msr2msrTranslator
  msr2msrTranslator
    translator;

  // build the resulting MSR score
  S_msrScore
    resultingNewMsrScore =
      translator.translateMsrToMsr (
        originalMsrScore);

  // ... ... ...
}
```

14.30 context arguments

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

```
void msrMeasureRepeat::displayMeasureRepeat (
  int    inputLineNumber,
  const string& context)
{
  gLogStream <<
    endl <<
    "*********>> MeasureRepeat " <<
    ", measureRepeatMeasuresNumber: '" <<
    fMeasureRepeatMeasuresNumber <<
    ", measureRepeatSlashesNumber: '" <<
    fMeasureRepeatSlashesNumber <<
    "', voice:" <<
    endl <<
    fMeasureRepeatVoiceUpLink->getVoiceName () <<
    " (" << context << ")" <<
    ", line " << inputLineNumber <<
    " contains:" <<
    endl;

  ++gIndenter;
  print (gLogStream);
  --gIndenter;

  gLogStream <<
    " <<*********" <<
    endl << endl;
```

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

```
    displayVoiceMeasureRepeatAndVoice (
      inputLineNumber,
      "createMeasureRepeatFromItsFirstMeasures() 1");
```

14.31 Sorting and compare*() methods

MusicFormats sometimes needs to sort some data structures:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r '\.sort (' *
oah/oahBasicTypes.cpp:  optionsMapElementsNamesList.sort ();
passes/lpsr2lilypond/lpsr2lilypondTranslator.cpp:    frameFrameNotesList.sort (
formats/msr/msrMeasuresSlices.cpp:  fSliceNotesFlatList.sort (
formats/msr/msrMeasuresSlices.cpp:  fSliceNoteEventsList.sort (
formats/msr/msrStaves.cpp:        fStaffAllVoicesList.sort (
formats/msr/msrStaves.cpp:        fStaffAllVoicesList.sort (
formats/msr/msrStaves.cpp:  fStaffRegularVoicesList.sort (
formats/msr/msrMeasures.cpp:    fMeasureElementsList.sort (
formats/msr/msrMeasures.cpp:    fMeasureElementsList.sort (
formats/msr/msrParts.cpp:    fPartAllStavesList.sort (
formats/msr/msrParts.cpp:    fPartAllStavesList.sort (
formats/lpsr/lpsrParts.cpp:    fPartBlockElementsList.sort (
```

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r compare * | grep '\.h'
oah/oahBasicTypes.h:    const multiset<S_oahElement, compareOahElements>&
oah/oahBasicTypes.h:    multiset<S_oahElement, compareOahElements>
oah/oahElements.h:struct compareOahElements {
formats/msr/msrParts.h:    static bool
    compareStavesToHaveFiguredBassElementsBelowCorrespondingPart (
formats/msr/msrNotes.h:    static bool           compareNotesByIncreasingPositionInMeasure
    (
formats/msr/msrMeasureElements.h:    static bool
    compareMeasureElementsByIncreasingPositionInMeasure (
formats/msr/msrStaves.h:    static bool           compareVoicesByIncreasingNumber (
formats/msr/msrStaves.h:    static bool
    compareVoicesToHaveHarmoniesAboveCorrespondingVoice (
formats/msr/msrStaves.h:    static bool
    compareVoicesToHaveFiguredBassElementsBelowCorrespondingVoice (
formats/msr/msrMeasuresSlices.h:    static bool
    compareNotesEventsByIncreasingPositionInMeasure (
formats/msr/msrMeasuresSlices.h:    static bool
    compareSimultaneousNotesChunksByIncreasingPositionInMeasure (
formats/lpsr/lpsrParts.h:    static bool
    compareElementsToHaveHarmoniesAboveCorrespondingStaff (
formats/lpsr/lpsrParts.h:    static bool           compareStaffBlockWithOtherElement (
formats/lpsr/lpsrParts.h:    static bool
    compareChordNamesContextWithOtherElement (
utilities/mfutilities.h:    // compare indentation value
```

An example is:

```
bool msrPart::compareStavesToHaveFiguredBassElementsBelowCorrespondingPart (
  const S_msrStaff& first,
  const S_msrStaff& second)
{
  int
    firstStaffNumber =
      first->getStaffNumber (),
    secondStaffNumber =
      second->getStaffNumber ();

  if (firstStaffNumber > msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER) {
    firstStaffNumber -= msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER + 1;
  }
  if (secondStaffNumber > msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER) {
    secondStaffNumber -= msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER + 1;
  }

  bool result =
    firstStaffNumber > secondStaffNumber;

  return result;
}
```

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`:

```
class mfcVersionNumber: public smartable
{
  // ... ... ...

  public:

    // public services
    // ------------------------------------------------------

    Bool                  operator== (const mfcVersionNumber& other) const;

    Bool                  operator!= (const mfcVersionNumber& other) const;

    Bool                  operator<  (const mfcVersionNumber& other) const;

    Bool                  operator>= (const mfcVersionNumber& other) const;

    Bool                  operator>  (const mfcVersionNumber& other) const;

    Bool                  operator<= (const mfcVersionNumber& other) const;

  public:

    // print
    // ------------------------------------------------------

    string                asString () const;

    void                  print (ostream& os) const;

  private:

    // fields
    // ------------------------------------------------------

    int                   fMajorNumber;
    int                   fMinorNumber;
    int                   fPatchNumber;
```

```
    string                  fPreRelease;
};
```

15.2 Versions descriptions

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

```
class mfcVersionDescr : public smartable
{
  // ... ... ...

  private:

    // fields
    // ------------------------------------------------------

    S_mfcVersionNumber    fVersionNumber;
    string                fVersionDate;
    list<string>          fVersionDescriptionItems;
};
```

15.3 Versions histories

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

```
class mfcVersionsHistory : public smartable
{
  // ... ... ...

  public:

    // public services
    // ------------------------------------------------------

    void                  appendVersionDescrToHistory (
                            S_mfcVersionDescr versionDescr);

    S_mfcVersionDescr     fetchCurrentVersion () const;

    S_mfcVersionNumber    fetchCurrentVersionNumber () const;

  // ... ... ...

  protected:

    // protected fields
    // ------------------------------------------------------

    list<S_mfcVersionDescr>
                          fVersionsList;
};
```

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

```
S_mfcVersionDescr mfcVersionsHistory::fetchCurrentVersion () const
{
  // sanity check
  mfAssert (
    __FILE__, __LINE__,
    fVersionsList.size () > 0,
```

```
    "fVersionsList is empty");

  return fVersionsList.back ();
}
```

15.4 Components descriptions

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

```
enum class mfcComponenKind {
  kComponentRepresentation,
  kComponentPass,
  kComponentGenerator,
  kComponentConverter,
  kComponentLibrary
};
```

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

```
class mfcComponentDescr : public smartable
{
  // ... ... ...

  public:

    // public services
    // ------------------------------------------------------

    S_mfcVersionDescr     fetchComponentCurrentVersion () const
                              {
                                return
                                  fVersionsHistory->
                                     fetchCurrentVersion();
                              }

  // ... ... ...

  public:

    // print
    // ------------------------------------------------------

    string                asString () const;

    string                currentVersionNumberAndDateAsString () const;

    virtual void          print (ostream& os) const;

    virtual void          printVersion (ostream& os) const;
    virtual void          printHistory (ostream& os) const;

  protected:

    // protected services
    // ------------------------------------------------------

    virtual void          printOwnHistory (ostream& os) const;

  protected:

    // protected fields
    // ------------------------------------------------------
```

```
44
45      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 `mfcComponentDescr::printOwnHistory ()`. Class `mfcMultiComponent` is described below.

15.5 Multi-components

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

```
1  class mfcMultiComponent : public mfcComponentDescr
2  {
3    // ... ... ...
4
5    protected:
6
7      // protected fields
8      // ------------------------------------------------------
9
10     list<S_mfcRepresentationComponent>
11                          fRepresentationComponentsList;
12     list<S_mfcPassComponent>
13                          fPassComponentsList;
14
15     // should the version number be at least equal to
16     // the ones of the components?
17     mfcMultiComponentEntropicityKind
18                          fComponentEntropicityKind;
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:

```
enum class mfcMultiComponentEntropicityKind {
  kComponentEntropicityYes,
  kComponentEntropicityNo
};
```

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:

```
enum class mfcMultiComponentUsedFromTheCLIKind {
  kComponentUsedFromTheCLIYes,
  kComponentUsedFromTheCLINo
};
```

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:

```
jacquesmenu@macmini: ~ > xml2ly -version
Command line version of musicxml2lilypond converter v0.9.51 (October 12 2021)

Representations versions:
  MXSR
    v0.9.5 (October 6 2021)
  MSR
    v0.9.51 (October 14 2021)
  LPSR
    v0.9.5 (October 6 2021)

Passes versions:
  mxsr2msr
    v0.9.5 (October 6 2021)
  msr2msr
    v0.9.5 (October 6 2021)
  msr2lpsr
    v0.9.5 (October 6 2021)
  lpsr2lilypond
    v0.9.5 (October 6 2021)
```

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

```
jacquesmenu@macmini: ~ > xml2brl -history
Command line version of musicxml2braille converter v0.9.51 (October 12 2021)

Own history:
  v0.9.5 (October 6 2021):
    Start of sequential versions numbering

  v0.9.51 (October 12 2021):
    Fixed trace OAH issue

Representations history:
  MXSR
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

  MSR
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

    v0.9.51 (October 14 2021):
```

```
      Refined MSR names and summary display options

  BSR
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

Passes history:
  mxsr2msr
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

  msr2msr
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

  msr2bsr
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

  bsr2bsr
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

  bsr2braille
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering
```

15.6 Versions history creation

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`:

```
S_mfcRepresentationComponent EXP createMsrRepresentationComponent ()
{
  static S_mfcRepresentationComponent pRepresentationComponent;

  // protect library against multiple initializations
  if (! pRepresentationComponent) {

#ifdef TRACING_IS_ENABLED
    if (gGlobalOahEarlyOptions.getEarlyTraceComponents ()) {
      gLogStream <<
        "Initializing MSR format component" <<
        endl;
    }
#endif

    // create the format component
    pRepresentationComponent =
      mfcRepresentationComponent::create (
        "MSR");
```

```
20
21      // populate it
22      pRepresentationComponent->
23        appendVersionDescrToComponent (
24          mfcVersionDescr::create (
25            mfcVersionNumber::createFromString ("0.9.50"),
26            "October 6, 2021",
27            list<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            list<string> {
38              "Refined MSR names and summary display options"
39            }
40        ));
41    }
42
43    return pRepresentationComponent;
44  }
```

The conversion of MusicXML to MXSRdoes 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         endl;
13     }
14 #endif
15
16     // create the converter component
17     pConverterComponent =
18       mfcConverterComponent::create (
19         "musicxml2braille",
20         mfcMultiComponentEntropicityKind::kComponentEntropicityNo,
21         mfcMultiComponentUsedFromTheCLIKind::kComponentUsedFromTheCLIYes); // 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",
```

```
          list<string> {
            "Start of sequential versions numbering"
          }
      ));

    pConverterComponent->
      appendVersionDescrToComponent (
        mfcVersionDescr::create (
          mfcVersionNumber::createFromString ("0.9.51"),
          "October 12, 2021",
          list<string> {
            "Fixed trace OAH issue"
          }
      ));

    // populate the converter's formats list
    pConverterComponent->
      appendRepresentationToMultiComponent (
        createMxsrRepresentationComponent ());
    pConverterComponent->
      appendRepresentationToMultiComponent (
        createMsrRepresentationComponent ());
    pConverterComponent->
      appendRepresentationToMultiComponent (
        createBsrRepresentationComponent ());

    pConverterComponent->
      appendPassToMultiComponent (
        createMxsr2msrComponent ());

    pConverterComponent->
      appendPassToMultiComponent (
        createMsr2msrComponent ());

    pConverterComponent->
      appendPassToMultiComponent (
        createMsr2bsrComponent ());

    pConverterComponent->
      appendPassToMultiComponent (
        createBsr2bsrComponent ());

    pConverterComponent->
      appendPassToMultiComponent (
        createBsr2brailleComponent ());
  }

  return pConverterComponent;
}
```

15.6.3 MusicFormats library component creation

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

```
S_mfcLibraryComponent EXP createLibraryComponent ()
{
  static S_mfcLibraryComponent pLibraryComponent;

  // protect library against multiple initializations
  if (! pLibraryComponent) {

#ifdef TRACING_IS_ENABLED
    if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
      gLogStream <<
```

```
        "Creating the  MFC library component" <<
        endl;
    }
#endif

    // create the library's history
    pLibraryComponent =
      mfcLibraryComponent::create (
        "musicformats",
        mfcMultiComponentEntropicityKind::kComponentEntropicityNo,
        mfcMultiComponentUsedFromTheCLIKind::kComponentUsedFromTheCLIYes); // JMI ???

    // populate the library's own history
    pLibraryComponent->
      appendVersionDescrToComponent (
        mfcVersionDescr::create (
          mfcVersionNumber::createFromString ("0.9.50"),
          "October 6, 2021",
          list<string> {
            "Start of sequential versions numbering"
          }
      ));

    pLibraryComponent->
      appendVersionDescrToComponent (
        mfcVersionDescr::create (
          mfcVersionNumber::createFromString ("0.9.51"),
          "October 12, 2021",
          list<string> {
            "Adding a version number to the MusicFormats library",
            "Fixed trace OAH issue in the musicxml2* converters)"
          }
      ));

    pLibraryComponent->
      appendVersionDescrToComponent (
        mfcVersionDescr::create (
          mfcVersionNumber::createFromString ("0.9.52"),
          "October 12, 2021",
          list<string> {
            "Added MusicFormats library versions history to '-hist, -history'"
          }
      ));

    pLibraryComponent->
      appendVersionDescrToComponent (
        mfcVersionDescr::create (
          mfcVersionNumber::createFromString ("0.9.53"),
          "October 22, 2021",
          list<string> {
            "Replaced bool by class Bool in variables and fields",
            "Created MFC (MusicFormats components)"
          }
      ));

    pLibraryComponent->
      appendVersionDescrToComponent (
        mfcVersionDescr::create (
          mfcVersionNumber::createFromString ("0.9.54"),
          "Novermber 6, 2021",
          list<string> {
            "Replaced cout and cerr by gOutputStream and gLogStream respectively in the
    CLI samples",
            "Finalized components numbering (MFC)"
          }
      ));

```

```
77      // populate the library's components history
78      pLibraryComponent->
79        appendRepresentationToMultiComponent (
80          createMsrRepresentationComponent ());
81      pLibraryComponent->
82        appendRepresentationToMultiComponent (
83          createLpsrRepresentationComponent ());
84      pLibraryComponent->
85        appendRepresentationToMultiComponent (
86          createBsrRepresentationComponent ());
87      pLibraryComponent->
88        appendRepresentationToMultiComponent (
89          createMxsrRepresentationComponent ());
90
91      pLibraryComponent->
92        appendPassToMultiComponent (
93          createMsr2msrComponent ());
94
95      pLibraryComponent->
96        appendPassToMultiComponent (
97          createMsr2lpsrComponent ());
98      pLibraryComponent->
99        appendPassToMultiComponent (
100         createLpsr2lilypondComponent ());
101
102     pLibraryComponent->
103       appendPassToMultiComponent (
104         createMsr2bsrComponent ());
105     pLibraryComponent->
106       appendPassToMultiComponent (
107         createBsr2bsrComponent ());
108     pLibraryComponent->
109       appendPassToMultiComponent (
110         createBsr2brailleComponent ());
111
112     pLibraryComponent->
113       appendPassToMultiComponent (
114         createMsr2mxsrComponent ());
115
116     pLibraryComponent->
117       appendPassToMultiComponent (
118         createMxsr2musicxmlComponent ());
119
120     pLibraryComponent->
121       appendPassToMultiComponent (
122         createMxsr2guidoComponent ());
123   }
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:

```
//_______________________________________________________________________________
class EXP oahHandler : public smartable
{
  // ... ... ...

  protected:

    // protected initialization
    // ------------------------------------------------------

    virtual void          initializeHandlerMultiComponent () = 0;

  public:

    // set and get
    // ------------------------------------------------------

    // ... ... ...

    S_mfcMultiComponent   getHandlerMultiComponent () const
                            { return fHandlerMultiComponent; }

  // ... ... ...

  protected:

    // protected fields
    // ------------------------------------------------------

    // ... ... ...

    // compound versions
    S_mfcMultiComponent   fHandlerMultiComponent;
};
```

Field `oahHandler::fHandlerMultiComponent` is set in the `oahHandler` subclasses constructors by a call to the overriden `initializeHandlerMultiComponent ()`.

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

```
xml2xmlInsiderHandler::xml2xmlInsiderHandler (
  const string& serviceName,
  string handlerHeader)
  : oahInsiderHandler (
      serviceName,
      handlerHeader,
R"(
          Welcome to the MusicXML to MusicXML converter
          delivered as part of the MusicFormats library.

      --- https://github.com/jacques-menu/musicformats ---
)",
R"(
Usage: xml2xml [[option]* [MusicXMLFile|-] [[option]*
)")
{
  // ... ... ...

  // initialize the multi-component
  initializeHandlerMultiComponent ();

  // ... ... ...
}
```

The overriden `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`:

```
void Mikrokosmos3WanderingInsiderHandler::initializeHandlerMultiComponent ()
{
  fHandlerMultiComponent =
    createMikrokosmos3WanderingGeneratorComponent ();
}
```

15.7 Accessing versions in regular handlers

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

```
class EXP oahRegularHandler : public oahHandler
/*
  A regular OAH handler relies on the existence of so-called 'insider' handler,
  that contains all the options values gathered from the user,
  grouped according to the internal representations and passes used.

  The variables containing the values of the options chosen by the user
  are actually held by the insider handler.
*/
{
  // ... ... ... ...

  protected:

    // protected initialization
    // ------------------------------------------------------

  // ... ... ... ...

    void                    initializeHandlerMultiComponent () override
                              {
                                fHandlerMultiComponent =
                                  fInsiderHandler->
                                    getHandlerMultiComponent ();
                              }

  // ... ... ... ...
};
```

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:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxml > xml2xml -version
Command line version of musicxml2musicxml converter v0.9.51 (October 12 2021)

Representations versions:
  MXSR
    v0.9.5 (October 6 2021)
  MSR
    v0.9.51 (October 14 2021)

Passes versions:
  mxsr2msr
    v0.9.5 (October 6 2021)
  msr2msr
    v0.9.5 (October 6 2021)
  msr2mxsr
    v0.9.5 (October 6 2021)
  mxsr2musicxml
    v0.9.5 (October 6 2021)
```

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxml > xml2gmn -history
Command line version of musicxml2guido converter v0.9.51 (October 12 2021)

Own history:
  v0.9.5 (October 6 2021):
    Start of sequential versions numbering

  v0.9.51 (October 12 2021):
    Fixed trace OAH issue

Representations history:
  MXSR
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

  MSR
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

    v0.9.51 (October 14 2021):
      Refined MSR names and summary display options

Passes history:
  mxsr2msr
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

  msr2msr
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

  msr2mxsr
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering

  mxsr2guido
    v0.9.5 (October 6 2021):
      Start of sequential versions numbering
```

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

```
class EXP oahVersionAtom : public oahPureHelpAtomWithoutAValue
{
  // ... ... ...

  public:

    // public services
    // ------------------------------------------------------

    void                  applyElement (ostream& os) override;

  // ... ... ...

  public:

    // print
    // ------------------------------------------------------

    // ... ... ...

    void                  printVersion (ostream& os) const;
};
```

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

```
void oahVersionAtom::applyElement (ostream& os)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
    gLogStream <<
      "==> option '" << fetchNames () << "' is a oahVersionAtom" <<
      endl;
  }
#endif

  int saveIndent = gIndenter.getIndent ();

  gIndenter.resetToZero ();

  printVersion (os);

  gIndenter.setIndent (saveIndent);
}
```

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

```
void oahVersionAtom::printVersion (ostream& os) const
{
  // get the handler version
  S_mfcMultiComponent
    handlerMultiComponent =
      fetchAtomHandlerUpLink ()->
        getHandlerMultiComponent ();

  // sanity check
  mfAssert (
    __FILE__, __LINE__,
    handlerMultiComponent != nullptr,
    "handlerMultiComponent is null");

  handlerMultiComponent->
    printVersion (os);
}
```

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:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxml > xml2ly --lilypond-generation-
    infos basic/HelloWorld.xml
\version "2.22.0"

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

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

% Generated by xml2ly v0.9.51 (October 12 2021)
% on Thursday 2021-11-11 @ 11:15:56 CET
% from "basic/HelloWorld.xml"

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

Class `lpsrScore` contains an MFC component field:

```
class EXP lpsrScore : public lpsrElement
{
  // ... ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    // ... ... ... ...

    // the multi-component
    // ------------------------------------------------------
    S_mfcMultiComponent   fMultiComponent;

  // ... ... ... ...
};
```

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

```
lpsrScore::lpsrScore (
  int                 inputLineNumber,
  S_msrScore          theMsrScore,
  S_mfcMultiComponent multiComponent)
    : lpsrElement (inputLineNumber)
{
  // ... ... ... ...

  fMsrScore = theMsrScore;

  fMultiComponent = multiComponent;

  // should the initial comments about the service and the options used
  // be generated?
  if (gGlobalLpsr2lilypondOahGroup->getXml2lyInfos ()) {
    // create the 'input source name and translation date' comment
    {
      stringstream s;

```

```
      s <<
        "Generated by " <<
        gGlobalOahOahGroup->getOahOahGroupServiceName () <<
        " " <<
        fMultiComponent->
          currentVersionNumberAndDateAsString () <<
        endl <<

        "% on " <<
        gGlobalServiceRunData->getTranslationDateFull () <<
        endl <<

        "% from ";

      if (gGlobalServiceRunData->getInputSourceName () == "-") {
        s << "standard input";
      }
      else {
        s << "\"" << gGlobalServiceRunData->getInputSourceName () << "\"";
      }

      fInputSourceNameComment =
        lpsrComment::create (
          inputLineNumber,
          s.str (),
          lpsrComment::kGapAfterwardsNo);
    }

    // ... ... ... ...
  }

  // ... ... ... ...
}
```

15.8.3 Current version numbers in passes

Another case is that of the generation of MusicXML output:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxml > xml2xml -musicxml-generation-
    infos basic/HelloWorld.xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
      "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.1">
    <!--
=================================================
Created by xml2xml v0.9.5 (October 6 2021)
on Thursday 2021-11-11 @ 11:04:06 CET
from basic/HelloWorld.xml
=================================================
-->
    <work>
        <work-number/>
        <work-title>Hello World!</work-title>
    </work>
    <movement-number/>
    <movement-title/>
    <identification>
        <encoding>
            <software>xml2xml v0.9.5 (October 6 2021), https://github.com/jacques-menu/
    musicformats</software>
            <encoding-date>2021-11-10</encoding-date>
        </encoding>
        <miscellaneous>
```

```
            <miscellaneous-field name="description"/>
        </miscellaneous>
    </identification>

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

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

```
void msr2mxsrTranslator::visitStart (S_msrScore& elt)
{
  // ... ... ... ...

  // get the pass component
  S_mfcPassComponent
    passComponent =
      createMsr2mxsrComponent ();

  // get the pass component current version number and date
  string
    passComponentCurrentVersionNumberAndDateAsString =
      passComponent->
        currentVersionNumberAndDateAsString ();

  // create the initial creation comment
  stringstream s;
  s <<
    endl <<
    "%================================================" <<
    endl <<
    "Created by " <<
    gGlobalOahOahGroup->getOahOahGroupServiceName () <<
    " " <<
    passComponentCurrentVersionNumberAndDateAsString <<
    endl <<

    "on " <<
    gGlobalServiceRunData->getTranslationDateFull () <<
    endl <<

    "from " <<
    gGlobalServiceRunData->getInputSourceName () <<
    endl <<

    "%================================================" <<
    endl;

  // append the initial creation comment to the score part wise element
  fResultingMusicxmlelement->push (createMxmlelement (kComment, s.str ()));

  // create a software element
  Sxmlelement
    softwareElement =
      createMxmlelement (
        k_software,
        gGlobalOahOahGroup->getOahOahGroupServiceName ()
          + " "
          + passComponentCurrentVersionNumberAndDateAsString +
          ", https://github.com/jacques-menu/musicformats");

  // append it to the identification encoding
  appendToScoreIdentificationEncoding (softwareElement);

  // ... ... ... ...
}
```

Part V

Options and help

Chapter 16

Options and help (OAH)

OAH is a powerful way of representating 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 "whaaaah!" 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 subclasses 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 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 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 (ostream& os) override;
15                                // reports an error
16
17      virtual void            applyAtomWithValue (
18                                const string& theString,
19                                ostream&      os) = 0;
20
21      virtual void            applyAtomWithDefaultValue (ostream& os);
22                                // used only if fElementValueKind
23                                // is oahElementValueKind::kElementValueImplicit
24                                // or oahElementValueKind::kElementValueOptional
25  };
```

The classes derived from `oahAtomExpectingAValue` are:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grh 'public oahAtomExpectingAValue'
oah/oahBasicTypes.h:class EXP oahAtomStoringAValue : public oahAtomExpectingAValue
oah/oahBasicTypes.h:class EXP oahPureHelpAtomExpectingAValue : public
    oahAtomExpectingAValue
passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondRelativeOctaveEntryAtom : public
     oahAtomExpectingAValue
passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondFixedOctaveEntryAtom : public
    oahAtomExpectingAValue
```

16.3.1 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                              ostream& os,
12                              int fieldWidth) const;
13     virtual void           printAtomWithVariableEssentialsShort (
14                              ostream& os,
15                              int fieldWidth) const;
16
17     void                   print (ostream& os) const override;
18     void                   printShort (ostream& os) const override;
19
20     void                   printHelp (ostream& os) const override;
21
22     virtual void           printAtomWithVariableOptionsValues (
23                              ostream& os,
24                              int      valueFieldWidth) const override;
25
26   protected:
27
28     // protected fields
29     // ------------------------------------------------------
30
31     string                 fValueSpecification;
32
33     string                 fVariableName;
34     Bool                   fSetByUser;
```

```
};
```

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

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

```
void oahIntegerAtom::setIntegerVariable (int value)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
    gLogStream <<
      "Setting option '" <<
      fetchNames () <<
      "' integer variable to '" <<
      value <<
      "'" <<
      endl;
  }
#endif

  fIntegerVariable = value;
  fSetByUser = true;
}
```

16.3.2 The `oahBooleanAtom` special case

Class `oahBooleanAtom` has its own `fSetByUser` 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:

```
//______________________________________________________________________________
class EXP oahBooleanAtom : public oahAtom
{
/*
  an atom controlling a Bool variable variableName,
  but expecting no value to be supplied:
  the variable is false initially,
  and is set to true by the mere occurrence of the atom
*/

  // ... ... ...

  protected:

    // protected fields
    // ------------------------------------------------------

    string                fVariableName;
    Bool&                 fBooleanVariable;
    Bool                  fSetByUser;
};
```

16.3.3 Checking whether an option has been selected

This in done by calls to the `getSetByUser ()` methods.

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

```
void lpsr2lilypondTranslator::generateLilypondVersion ()
{
  // LilyPond version
  Bool
    lilypondVersionHasBeenSet =
      gGlobalLpsr2lilypondOahGroup->
        getLilypondVersionAtom ()->
          getSetByUser ();

  string
    lilypondVersion =
      lilypondVersionHasBeenSet
        ? gGlobalLpsr2lilypondOahGroup->
            getLilypondVersion ()
        : gGlobalLpsr2lilypondOahGroup->
           getLilypondVersionDefaultValue ();

  fLilypondCodeStream <<
    "\\version \"" <<
    lilypondVersion <<
    "\"" <<
    endl << endl;
}
```

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

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

16.3.4 The `oahAtomStoringAValue` subclasses

The classes derived from `oahAtomStoringAValue` are:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grh 'public oahAtomStoringAValue'
oah/harmoniesExtraOah.h:class EXP extraShowAllHarmoniesStructuresAtom : public
    oahAtomStoringAValue
oah/harmoniesExtraOah.h:class EXP extraShowAllHarmoniesContentsAtom : public
    oahAtomStoringAValue
oah/harmoniesExtraOah.h:class EXP extraShowHarmonyDetailsAtom : public
    oahAtomStoringAValue
oah/harmoniesExtraOah.h:class EXP extraShowHarmonyAnalysisAtom : public
    oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahIntegerAtom : public oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahFloatAtom : public oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahStringAtom : public oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahRationalAtom : public oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahNaturalNumbersSetElementAtom : public
    oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahRGBColorAtom : public oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahIntSetElementAtom : public oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahStringSetElementAtom : public oahAtomStoringAValue
```

```
oah/oahAtomsCollection.h:class EXP oahStringToIntMapElementAtom : public
    oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahStringAndIntegerAtom : public oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahStringAndTwoIntegersAtom : public
    oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahLengthUnitKindAtom : public oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahLengthAtom : public oahAtomStoringAValue
oah/oahAtomsCollection.h:class EXP oahMidiTempoAtom : public oahAtomStoringAValue
formatsgeneration/brailleGeneration/brailleGenerationOah.h:class EXP brailleOutputKindAtom
     : public oahAtomStoringAValue
formatsgeneration/brailleGeneration/brailleGenerationOah.h:class EXP brailleUTFKindAtom :
    public oahAtomStoringAValue
formatsgeneration/brailleGeneration/brailleGenerationOah.h:class EXP
    brailleByteOrderingKindAtom : public oahAtomStoringAValue
formatsgeneration/msrGeneration/msrGenerationBasicTypes.h:class EXP
    msrGenerationAPIKindAtom : public oahAtomStoringAValue
formatsgeneration/multiGeneration/mfMultiGenerationOah.h:class EXP
    mfMultiGenerationOutputKindAtom : public oahAtomStoringAValue
passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondScoreOutputKindAtom : public
    oahAtomStoringAValue
passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondTransposePartNameAtom : public
    oahAtomStoringAValue
passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondTransposePartIDAtom : public
    oahAtomStoringAValue
passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondAccidentalStyleKindAtom : public
     oahAtomStoringAValue
passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondChordsDisplayAtom : public
    oahAtomStoringAValue
passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondLyricsDurationsKindAtom : public
     oahAtomStoringAValue
passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP
    lilypondDynamicsTextSpannersStyleKindAtom : public oahAtomStoringAValue
passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondBreakPageAfterMeasureNumberAtom
    : public oahAtomStoringAValue
passes/msr2lpsr/msr2lpsrOah.h:class EXP msr2lpsrScoreOutputKindAtom : public
    oahAtomStoringAValue
passes/msr2msr/msr2msrOah.h:class EXP msrIgnorePartAtom : public oahAtomStoringAValue
passes/msr2msr/msr2msrOah.h:class EXP msrKeepPartAtom : public oahAtomStoringAValue
passes/mxsr2msr/mxsr2msrOah.h:class EXP msrReplaceClefAtom : public oahAtomStoringAValue
formats/bsr/bsrOah.h:class EXP bsrFacSimileKindAtom : public oahAtomStoringAValue
formats/bsr/bsrOah.h:class EXP bsrTextsLanguageAtom : public oahAtomStoringAValue
formats/lpsr/lpsrOah.h:class EXP lpsrPitchesLanguageAtom : public oahAtomStoringAValue
formats/lpsr/lpsrOah.h:class EXP lpsrChordsLanguageAtom : public oahAtomStoringAValue
formats/lpsr/lpsrOah.h:class EXP lpsrTransposeAtom : public oahAtomStoringAValue
formats/msdl/msdlInputOah.h:class EXP msdlKeywordsLanguageAtom : public
    oahAtomStoringAValue
formats/msdl/msdlInputOah.h:class EXP msdlCommentsTypeAtom : public oahAtomStoringAValue
formats/msdl/msdlInputOah.h:class EXP msdlUserLanguageAtom : public oahAtomStoringAValue
formats/msdl/msdlInputOah.h:class EXP msdlPitchesLanguageAtom : public
    oahAtomStoringAValue
formats/msr/msrOah.h:class EXP msrPitchesLanguageAtom : public oahAtomStoringAValue
formats/msr/msrOah.h:class EXP msrRenamePartAtom : public oahAtomStoringAValue
```

16.4 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.

16.4.1 Pure help atoms without a value

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

```
//_____________________________________________________________________________
class EXP oahPureHelpAtomWithoutAValue : public oahAtom
{
  // ... ... ...

  protected:

    // protected fields
    // ------------------------------------------------------

    string                 fHelpAtomWithoutAValueServiceName;
};
```

The actual pure help atoms without a value are:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grh 'public oahPureHelpAtomWithoutAValue
    '
oah/oahAtomsCollection.h:class EXP oahOptionsUsageAtom : public
    oahPureHelpAtomWithoutAValue
oah/oahAtomsCollection.h:class EXP oahHelpAtom : public oahPureHelpAtomWithoutAValue
oah/oahAtomsCollection.h:class EXP oahHelpSummaryAtom : public
    oahPureHelpAtomWithoutAValue
oah/oahAtomsCollection.h:class EXP oahAboutAtom : public oahPureHelpAtomWithoutAValue
oah/oahAtomsCollection.h:class EXP oahVersionAtom : public oahPureHelpAtomWithoutAValue
oah/oahAtomsCollection.h:class EXP oahLibraryVersionAtom : public
    oahPureHelpAtomWithoutAValue
oah/oahAtomsCollection.h:class EXP oahHistoryAtom : public oahPureHelpAtomWithoutAValue
oah/oahAtomsCollection.h:class EXP oahLibraryHistoryAtom : public
    oahPureHelpAtomWithoutAValue
oah/oahAtomsCollection.h:class EXP oahContactAtom : public oahPureHelpAtomWithoutAValue
```

16.4.2 Pure help atoms expecting a value

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

```
class EXP oahPureHelpAtomExpectingAValue : public oahAtomExpectingAValue
{
  // ... ... ...

  protected:

    // protected fields
    // ------------------------------------------------------

    string                 fHelpAtomExpectingAValueServiceName; // JMI ???
};
```

The actual pure help atoms expecting a value are:

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

16.5 Options implicitly storing a value

There are options in multiGeneration to select the generated output :

```
jacquesmenu@macmini > msdlconverter --help-generate-output
--- Help for subgroup "Generated output" in group "Generated output group" ---
  Generated output group (-help-generate-output-group, -hgc-group):
  ---------------------------
    Generated output (-help-generate-output, -hgo):
      -guido
            Generate Guido code as output.
      -lilypond
            Generate LilyPond code as output.
      -braille
            Generate braille code as output.
      -musicxml
            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:

```
class EXP oahAtomImplicitlyStoringAValue : public oahAtom
/*
  a purely virtual common ancestor for all atom classes
  that store a value in a variable
  withoud taking it from argv or an oahOptionsVector
*/
{
  // ... ... ...

  protected:

    // protected fields
    // ------------------------------------------------------

    string                fVariableName;
    Bool                  fSetByUser;
};
```

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

```
class EXP mfMultiGenerationOutputKindAtom : public oahAtomImplicitlyStoringAValue
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    mfMultiGenerationOutputKind&
                          fMultiGenerationOutputKindVariable;
};
```

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

```
mfMultiGenerationOutputKindAtom::mfMultiGenerationOutputKindAtom (
  const string&             longName,
  const string&             shortName,
  const string&             description,
  const string&             variableName,
  mfMultiGenerationOutputKind& mfMultiGenerationOutputKindVariable)
  : oahAtomImplicitlyStoringAValue (
      longName,
      shortName,
      description,
      variableName,
      oahElementValueKind::kElementValueWithout),
    fMultiGenerationOutputKindVariable ( // this is where the value is supplied
      mfMultiGenerationOutputKindVariable)
{}
```

16.6 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 ()` associatiates the value to the (preceding) field `oahHandler::f`
  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 it value, which is the next element in `argv`.

16.7 Basic OAH types

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class oahBasicTypes.h
// PRE-declarations for class mutual dependencies
class oahAtom;
class oahSubGroup;
class oahGroup;
class oahHandler;
enum class oahOptionsDefaultValuesStyle {
```

```
enum class oahHandlerUsedThruKind {
class oahPrefix;
class EXP oahPrefix : public smartable
  a common ancestor for all atom classes,
  this class contains  only an uplink to the containing subgroup
class EXP oahAtom : public oahElement
/* this class is purely virtual
class EXP oahAtomExpectingAValue : public oahAtom
  a common ancestor for all atom classes
/* this class is purely virtual
  a common ancestor for all atom classes
class EXP oahAtomStoringAValue : public oahAtomExpectingAValue
/* this class is purely virtual
class EXP oahPureHelpAtomWithoutAValue : public oahAtom
/* this class is purely virtual
class EXP oahPureHelpAtomExpectingAValue : public oahAtomExpectingAValue
/* this class is purely virtual
class EXP oahSubGroup : public oahElement
class EXP oahGroup : public oahElement
class EXP oahHandler : public smartable
 /* this class is purely virtual
enum class oahOptionalValuesStyleKind {
class EXP oahAtomWithoutAValue : public oahAtom
/ * this class is purely virtual
```

16.8 Prefixes handling

16.9 `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/mflibrarymfMusicformatsError.h:

```
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 ans populated from `argc/argv` with `convertArgcArgvToOption`
defined in src/oah/oahBasicTypes.h:

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

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

```
    // options and arguments handling
    oahElementHelpOnlyKind
                          handleOptionsFromOptionsAndArguments (
                            string                  serviceName,
                            const oahOptionsVector& theOptionsVector);

    oahElementHelpOnlyKind
                          handleOptionsAndArgumentsFromArgcArgv (
                            int    argc,
                            char* argv[]);

    virtual void          checkOptionsAndArgumentsConsistency ();

    virtual void          checkOptionsAndArguments () const = 0;
```

16.10 Applying options

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

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

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

```
    virtual void          applyAtomWithValue (
                            const string& theString,
                            ostream& os) = 0;

    virtual void          applyAtomWithDefaultValue (ostream& os);
                            // used only if fElementValueKind
                            // is oahElementValueKind::kElementValueImplicit
                            // or oahElementValueKind::kElementValueOptional
```

There are two methods for that:

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

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

16.11 A OAH atoms collection

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class oahAtomsCollection.h
class EXP oahAtomAlias : public oahAtom
class EXP oahMacroAtom : public oahAtom
class EXP oahOptionsUsageAtom : public oahPureHelpAtomWithoutAValue
class EXP oahHelpAtom : public oahPureHelpAtomWithoutAValue
class EXP oahHelpSummaryAtom : public oahPureHelpAtomWithoutAValue
class EXP oahAboutAtom : public oahPureHelpAtomWithoutAValue
class EXP oahVersionAtom : public oahPureHelpAtomWithoutAValue
class EXP oahContactAtom : public oahPureHelpAtomWithoutAValue
class EXP oahBooleanAtom : public oahAtom
class EXP oahTwoBooleansAtom : public oahBooleanAtom
class EXP oahThreeBooleansAtom : public oahBooleanAtom
class EXP oahCombinedBooleansAtom : public oahAtom
```

```
class EXP oahCommonPrefixBooleansAtom : public oahAtom
class EXP oahIntegerAtom : public oahAtomStoringAValue
class EXP oahTwoIntegersAtom : public oahIntegerAtom
class EXP oahFloatAtom : public oahAtomStoringAValue
class EXP oahStringAtom : public oahAtomStoringAValue
class EXP oahFactorizedStringAtom : public oahAtom
class EXP oahStringWithDefaultValueAtom : public oahStringAtom
class EXP oahRationalAtom : public oahAtomStoringAValue
class EXP oahNaturalNumbersSetElementAtom : public oahAtomStoringAValue
class EXP oahRGBColorAtom : public oahAtomStoringAValue
class EXP oahIntSetElementAtom : public oahAtomStoringAValue
class EXP oahStringSetElementAtom : public oahAtomStoringAValue
class EXP oahStringToIntMapElementAtom : public oahAtomStoringAValue
class EXP oahStringAndIntegerAtom : public oahAtomStoringAValue
class EXP oahStringAndTwoIntegersAtom : public oahAtomStoringAValue
class EXP oahLengthUnitKindAtom : public oahAtomStoringAValue
class EXP oahLengthAtom : public oahAtomStoringAValue
class EXP oahMidiTempoAtom : public oahAtomStoringAValue
class EXP oahOptionNameHelpAtom : public oahStringWithDefaultValueAtom
class EXP oahQueryOptionNameAtom : public oahPureHelpAtomExpectingAValue
class EXP oahFindStringAtom : public oahPureHelpAtomExpectingAValue
```

See chapter 37, [The OAH atoms collection], page 238, for more details.

16.12 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:

```
class EXP mxsr2msrOahGroup : public oahGroup

    Bool                    fBeamAllGraceNotes;

    Bool                    getBeamAllGraceNotes () const
                                { return fBeamAllGraceNotes; }
```

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

```
void mxsr2msrOahGroup::initializeNotesOptions ()

  // beam all grace notes
  // ------------------------------------

  fBeamAllGraceNotes = false;

  S_oahBooleanAtom
    beamAllGraceNotesAtom =
      oahBooleanAtom::create (
        "beamagn", "beam-all-grace-notes",
R"(Add a beam to all grace notes)",
        "beamAllGraceNotes",
        fBeamAllGraceNotes);
  subGroup->
    appendAtomToSubGroup (
      beamAllGraceNotesAtom);
```

And that's it.

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

```
void mxsr2msrTranslator::visitStart ( S_grace& elt )

  // should all grace notes be beamed?
  if (gGlobalMxsr2msrOahGroup->getBeamAllGraceNotes ()) {
    fCurrentGraceIsBeamed = true;
  }

void mxsr2msrTranslator::handleStandaloneOrDoubleTremoloNoteOrGraceNoteOrRest (
  S_msrNote newNote)

      // create grace notes group
      fPendingGraceNotesGroup =
        msrGraceNotesGroup::create (
          inputLineNumber,
          msrGraceNotesGroup::kGraceNotesGroupBefore, // default value
          fCurrentGraceIsSlashed,
          fCurrentGraceIsBeamed,
          fCurrentMeasureNumber);
```

16.13 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:

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

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

```
jacquesmenu@macmini > xml2ly -find grace
5 occurrences of string "grace" have been found:
   1:
    -hgraces-group, -help-grace-notes-group

   2:
    -hgraces, -help-grace-notes

   3:
    -slashagn, -slash-all-grace-notes
    Add a slash to all grace notes
   4:
    -sluragn, -slur-all-grace-notes
    Add a slur to all grace notes
   5:
    -beamagn, -beam-all-grace-notes
    Add a beam to all grace notes
```

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

16.14 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 is was cumbersome to browse them without a user-oriented view by topics.

Class `oahRegularHandler` relies on the corresponding insider handler:

```
  protected:

    // protected fields
    // ------------------------------------------------------

    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:

```
void xml2xmlRegularHandler::createInformationsRegularGroup ()
{
  // group

  S_oahGroup
    group =
      oahGroup::create (
        "Informations group",
        "help-informations-group", "hinfos-group",
        "",
        oahElementVisibilityKind::kElementVisibilityWhole);
  appendGroupToRegulalHandler (group);

  // subgroup

  S_oahSubGroup
    subGroup =
      oahSubGroup::create (
        "Informations",
        "help-informations", "hinfos",
        "",
        oahElementVisibilityKind::kElementVisibilityWhole,
        group);
  group->
    appendSubGroupToGroup (subGroup);

  // atoms from the insider handler

  registerAtomInRegularSubgroup ("about", subGroup);
  registerAtomInRegularSubgroup ("version", subGroup);
  registerAtomInRegularSubgroup ("version-full", subGroup);
  registerAtomInRegularSubgroup ("history", subGroup);
  registerAtomInRegularSubgroup ("mf-version", subGroup);
  registerAtomInRegularSubgroup ("mf-history", subGroup);
  registerAtomInRegularSubgroup ("contact", subGroup);
  registerAtomInRegularSubgroup ("display-prefixes", subGroup);
  registerAtomInRegularSubgroup ("display-single-character-options", subGroup);

```

```
  registerAtomInRegularSubgroup ("display-cpu-usage", subGroup);
}
```

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:

```
    // create an msdlConverter insider OAH handler
    // ------------------------------------------------------

    S_msdlConverterInsiderHandler
      insiderOahHandler =
        msdlConverterInsiderHandler::create (
          serviceName,
          serviceName + " insider OAH handler with argc/argv",
          multiGenerationOutputKind);

    // the OAH handler to be used, a regular handler is the default
    // ------------------------------------------------------

    if (insiderOption) {
      // use the insider msdlConverter OAH handler
      handler = insiderOahHandler;
    }
    else {
      // create a regular msdlConverter OAH handler
      handler =
        msdlConverterRegularHandler::create (
          serviceName,
          serviceName + " regular OAH handler with argc/argv",
          insiderOahHandler,
          multiGenerationOutputKind);
    }
```

16.15 Deciphering the options and arguments

16.15.1 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:

```
int main (int argc, char*  argv[])
//------------------------------------------------------------------------
{
  // ... ... ...

  // create the global run data
  // ------------------------------------------------------

  gGlobalServiceRunData =
    mfServiceRunData::create (serviceName);

  // ... ... ...
}
```

- the API functions receive an `mfOptionsAndArguments` as an argument, here in src/converters/musicxml2musi

```
EXP mfMusicformatsError musicxmlfile2musicxml (
  const char*             fileName,
  mfOptionsAndArguments& handlerOptionsAndArguments,
  std::ostream&           out,
  std::ostream&           err)
{
  SXMLFile
    sxmlfile =
      createSXMLFileFromFile (
        fileName,
        "Pass 1",
        "Create an MXSR reading a MusicXML file");

  if (sxmlfile) {
    return
      xmlFile2musicxmlWithOptionsAndArguments (
        sxmlfile,
        handlerOptionsAndArguments,
        out,
        err);
  }

  return mfMusicformatsError::kErrorInvalidFile;
}
```

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, im this list are applied.

Class `oahHandler` contains:

```
    // elements uses
    list<S_oahElementUse> fElementUsesList;

    // atoms waiting for a value
    S_oahAtomExpectingAValue
                          fPendingArgvAtomExpectingAValue;
    string                fNameUsedForPendingArgvAtomExpectingAValue;
```

16.15.2 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:

```
enum class oahElementHelpOnlyKind {
  kElementHelpOnlyYes,
  kElementHelpOnlyNo
};
```

16.15.3 Applying options

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

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

The heart of it is:

```
oahElementHelpOnlyKind oahHandler::applyOptionsFromElementUsesList ()
{
  // ... ... ...

        // the heart of it
        if (
          // group?
          S_oahGroup
            group =
              dynamic_cast<oahGroup*>(&(*elementUsed))
        ) {
          group->
            applyElement (
              gOutputStream);
        }

        else if (
          // subgroup?
          S_oahSubGroup
            subGroup =
              dynamic_cast<oahSubGroup*>(&(*elementUsed))
        ) {
          subGroup->
            applyElement (
              gOutputStream);
        }

        else {
          // this is an atom

          S_oahAtom
            atom =
              dynamic_cast<oahAtom*>(&(*elementUsed));

          oahElementValueKind
            atomValueKind =
              atom->
                getElementValueKind ();

          if (
            // atom expecting a value?
            S_oahAtomExpectingAValue
              atomExpectingAValue =
                dynamic_cast<oahAtomExpectingAValue*>(&(*elementUsed))
          ) {
            switch (atomValueKind) {
              case oahElementValueKind::kElementValueWithout:
                {
                  stringstream s;

                  s <<
                    "Atom with value " <<
                    atomExpectingAValue->fetchNamesBetweenQuotes () <<
                    " has been registered as without value";

                  oahInternalError (s.str ());
                }
```

```
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                  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               endl;
107
108             ++gIndenter;
109
110             gLogStream <<
111               atom <<
112               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         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 }
```

16.15.4 Early handling of some options

Debugging OAH needs the trace handling tracing options option `-trace-oah, -toah` and option `-trace-oah-details, -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 string K_INSIDER_OPTION_LONG_NAME;
5  EXP extern const 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
```

```
                              { return fEarlyInsiderOption; }

  // ... ... ...

 private:

    // fields
    // ------------------------------------------------------

    Bool                   fEarlyInsiderOption;

    // ... ... ...
}
```

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

```
// ... ... ...

const string K_INSIDER_OPTION_LONG_NAME  = "insider";
const string K_INSIDER_OPTION_SHORT_NAME = "ins";

void oahEarlyOptions::setEarlyInsiderOption ()
{
  if (fTraceEarlyOptions) {
    gLogStream <<
      "Setting fEarlyInsiderOption" <<
      endl;
  }

  fEarlyInsiderOption = true;
}

// ... ... ...
```

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

```
void oahEarlyOptions::applyEarlyOptionIfRelevant (
  const string& argumentWithoutDashToBeUsed,
  const string& optionValue)
{
  // this is OAH handling pass 1
  if (
    isEarlyOptionRecognized (
      argumentWithoutDashToBeUsed, K_INSIDER_OPTION_LONG_NAME)
      ||
    isEarlyOptionRecognized (
      argumentWithoutDashToBeUsed, K_INSIDER_OPTION_SHORT_NAME)
  ) {
    setEarlyInsiderOption ();
  }

  // ... ... ...
}
```

16.16 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.

16.16.1 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 `mxsr2msrOah` and `msr2mxsrOah`. The corresponding options are thus placed in src/formats/mxsr/mxsrOah.h/.cpp:

16.16.2 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:

```
class EXP oahOahGroup : public oahGroup
{
    void                    setReverseNamesDisplayOrder ()
                                { fReverseNamesDisplayOrder = true; }
    Bool                    getReverseNamesDisplayOrder () const
                                { return fReverseNamesDisplayOrder; }

    // ... ... ...

    Bool                    fReverseNamesDisplayOrder;

    // ... ... ...
};
```

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

```
void oahOahGroup::initializeOahBasicHelpOptions (
  string serviceName)
{
  // ... ... ...

  // reverse names display order

  fReverseNamesDisplayOrder = false;

  subGroup->
    appendAtomToSubGroup (
      oahBooleanAtom::create (
        "rndo", "reverse-names-display-order",
```

```
R"(Write the short names before the long ones.)",
        "reverseNamesDisplayOrder",
        fReverseNamesDisplayOrder));

  // ... ... ...
}
```

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

```
void oahOahGroup::printOahOahValues (int valueFieldWidth)
{
  gLogStream <<
    "The basic options are:" <<
    endl;

  // ... ... ...

  // options and help display
  // ------------------------------------

  gLogStream << left <<
    setw (valueFieldWidth) << "Options trace and display:" <<
    endl;

  ++gIndenter;

  gLogStream << left <<
    setw (valueFieldWidth) << "fReverseNamesDisplayOrder" << " : " <<
    fReverseNamesDisplayOrder <<
    endl <<

  // ... ... ...
```

- 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 ()`:

```
string oahElement::fetchNames () const
{
  stringstream s;

  if (
    fShortName.size ()
        &&
    fLongName.size ()
  ) {
    if (gGlobalOahOahGroup->getReverseNamesDisplayOrder ()) {
      s <<
        "-" << fShortName <<
        ", " <<
        "-" << fLongName;
    }
    else {
      s <<
        "-" << fLongName <<
        ", " <<
        "-" << fShortName;
    }
  }

  else {
    if (fShortName.size ()) {
      s <<
      "-" << fShortName;
    }
    if (fLongName.size ()) {
      s <<
```

```
        "-" << fLongName;
    }
  }

  return s.str ();
}
```

- and finally, all `*RegularHandler::createOahRegularGroup ()` methods were augmented with:

```
 void msdl2brailleRegularHandler::createOahRegularGroup ()
{
  // ... ... ...

  registerAtomInRegularSubgroup ("reverse-names-display-order", subGroup);

  // ... ... ...
}
```

16.16.3 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:

```
class mfcMultiComponent : public smartable
{
  // ... ... ...

  public:

    // print
    // ------------------------------------------------------

    void                  print (ostream& os) const;
    void                  printHistory (ostream& os) const;

  // ... ... ...
};
```

Then the next thing has been to clone class `oahVersionAtom` in src/oah/oahAtomCollection.h/.cpp, renaming `printVersion ()` to `printHistory ()`:

```
//______________________________________________________________________________
class EXP oahHistoryAtom : public oahPureHelpAtomWithoutAValue
{
  // ... ... ...

  public:

    // public services
    // ------------------------------------------------------

    void                  applyElement (ostream& os) override;

  public:

    // visitors
    // ------------------------------------------------------

    void                  acceptIn  (basevisitor* v) override;
    void                  acceptOut (basevisitor* v) override;

```

```
    void                    browseData (basevisitor* v) override;

    // print
    // ------------------------------------------------------

    void                    print (ostream& os) const override;

    void                    printHistory (ostream& os) const;
};
```

Then in method `oahHistoryAtom::printHistory ()`, the call to `printVersion ()` has been replaced by a call to `printHistory ()`:

```
void oahHistoryAtom::applyElement (ostream& os)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
    gLogStream <<
      "==> option '" << fetchNames () << "' is a oahHistoryAtom" <<
      endl;
  }
#endif

  int saveIndent = gIndenter.getIndent ();

  gIndenter.resetToZero ();

  printHistory (os);

  gIndenter.setIndent (saveIndent);
}
```

method `:: ()` has the be adapted as:

```
void oahHistoryAtom::printHistory (ostream& os) const
{
  // get the handler history
  S_mfcMultiComponent
    handlerMultiComponent =
      fetchAtomHandlerUpLink ()->
        getHandlerMultiComponent ();

  // sanity check
  mfAssert (
    __FILE__, __LINE__,
    handlerMultiComponent != nullptr,
    "handlerMultiComponent is null");

  handlerMultiComponent->
    printHistory (os);
}
```

Then a new option has been added in method `oahOahGroup::initializeOahBasicHelpOptions ()`, in src/oah/oahOah.cpp:

```
  // history

  subGroup->
    appendAtomToSubGroup (
      oahHistoryAtom::create (
        "hist", "history",
        regex_replace (
R"(Display EXECUTABLE_NAME's history.)",
          regex ("EXECUTABLE_NAME"),
          serviceName),
        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`:

```
  registerAtomInRegularSubgroup ("version", subGroup);
  registerAtomInRegularSubgroup ("version-full", subGroup);
  registerAtomInRegularSubgroup ("history", subGroup);
  registerAtomInRegularSubgroup ("mf-version", subGroup);
  registerAtomInRegularSubgroup ("mf-history", subGroup);
```

16.16.4 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:

```
class EXP msrLength : public smartable
{
  // ... ... ...

    // public services
    // ------------------------------------------------------

    Bool                    operator== (const msrLength& other) const
                                {
                                  // JMI convert to same length unit kind before comparing
    BLARK
                                  return
                                    fLengthUnitKind == other.fLengthUnitKind
                                     &&
                                    fLengthValue == other.fLengthValue;
                                }

    Bool                    operator!= (const msrLength& other) const
                                { return ! ((*this) == other); }

    void                    convertToLengthUnit (
                              msrLengthUnitKind lengthUnitKind);

  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    msrLengthUnitKind       fLengthUnitKind;
    float                   fLengthValue;
};
```

Enumeration type is defined in src/formats/msr/msrBasicTypes.h as:

```
enum class msrLengthUnitKind {
  kUnitInch, kUnitCentimeter, kUnitMillimeter
};
```

Here is the declaration of class `oahLengthAtom` in src/oah/oahAtomsCollection.h:

```
class EXP oahLengthAtom : public oahAtomStoringAValue
{
/*
  an atom controlling a length variable
*/

  // ... ... ...

  public:

    // set and get
    // ------------------------------------------------------

    void                   setLengthVariable (
                             msrLength value)
                               {
                                 fLengthVariable = value;
                                 fSetByUser = true;
                               }

    msrLength              getLengthVariable () const
                               { return fLengthVariable; }

  public:

    // public services
    // ------------------------------------------------------

    void                   applyAtomWithValue (
                             const string& theString,
                             ostream&      os) override;

  // ... ... ...

  public:

    // print
    // ------------------------------------------------------

  // ... ... ...

    void                   printAtomWithVariableOptionsValues (
                             ostream& os,
                             int      valueFieldWidth) const override;

  private:

    // private fields
    // ------------------------------------------------------

    msrLength&             fLengthVariable;
};
```

Method `oahLengthAtom::applyAtomWithValue ()` in src/oah/oahAtomsCollection.cpp deciphers the string supplied by the user and stores it the class `msrLength` variable:

```
void oahLengthAtom::applyAtomWithValue (
  const string& theString,
  ostream&      os)
{
  // ... ... ...

  regex e (regularExpression);
  smatch sm;

```

```
  regex_match (theString, sm, e);

  unsigned int smSize = sm.size ();

  // ... ... ...

  if (smSize == 4) {
    // leave the low level details to the STL...
    float floatValue;
    {
      stringstream s;
      // concatenate the integer and decimal parts
      s << sm [ 1 ] << sm [ 2 ];
      s >> floatValue;
    }

    string lengthUnitName = sm [ 3 ];

    // is lengthUnitName known in the length unit names map?
    map<string, msrLengthUnitKind>::const_iterator
      it =
        gGlobalMsrLengthUnitKindsMap.find (
          lengthUnitName);

    if (it == gGlobalMsrLengthUnitKindsMap.end ()) {
      // no, length unit name is unknown in the map

      stringstream s;

      s <<
        "length unit name \"" << lengthUnitName <<
        "\" is unknown" <<
        endl <<
        "The " <<
        gGlobalMsrLengthUnitKindsMap.size () <<
        " known length unit names are:" <<
        endl;

      ++gIndenter;

      s <<
        existingMsrLengthUnitKinds (K_NAMES_LIST_MAX_LENGTH);

      --gIndenter;

      oahError (s.str ());
    }

    setLengthVariable (
      msrLength (
        (*it).second,
        floatValue));
  }

  else {
    stringstream s;

    s <<
      "length value \"" << theString <<
      "\" for option '" << fetchNames () <<
      "' is ill-formed";

    oahError (s.str ());
  }
}
```

Method `oahLengthAtom::printAtomWithVariableOptionsValues ()` is in charge of displaying the length value when option `-display-options-valuesdov` is chosen:

```
void oahLengthAtom::printAtomWithVariableOptionsValues (
  ostream& os,
  int      valueFieldWidth) const
{
  os << left <<
    setw (valueFieldWidth) <<
    fVariableName <<
    " : " <<
    fLengthVariable.asString ();
  if (fSetByUser) {
    os <<
      ", set by user";
  }
  os << endl;
}
```

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:

```
  // paper height

  fPaperHeight.setLengthUnitKind (msrLengthUnitKind::kUnitMillimeter);
  fPaperHeight.setLengthValue (297);

  fPaperHeightAtom =
    oahLengthAtom::create (
      "paper-height", "",
R"(Set the LilyPond 'paper-height' paper variable to HEIGHT in the LilyPond code.
HEIGHT should be a positive floating point or integer number,
immediately followed by a unit name, i.e. 'in', 'mm' or 'cm'.
By default, LilyPond uses 297 mm (A4 format).)",
      "HEIGHT",
      "paperHeight",
      fPaperHeight);
  subGroup->
    appendAtomToSubGroup (
      fPaperHeightAtom);
```

16.17 Extra options

The description of music scores in MusicFormats is quite rich, and it was easy (and temptating...) to offer options such as:

```
jacquesmenu@macmini > xml2ly -query show-harmony-analysis
--- Help for atom "show-harmony-analysis" in subgroup "Harmony analysis" of group "Extra"
    ---
    -sca, -show-harmony-analysis HARMONY_SPEC
          Write an analysis of the harmony for the given diatonic (semitones) pitch
          in the current language and the given harmony to standard output.
          HARMONY_SPEC can be:
          'ROOT_DIATONIC_PITCH HARMONY_NAME INVERSION'
          or
          "ROOT_DIATONIC_PITCH = HARMONY_NAME INVERSION"
          Using double quotes allows for shell variables substitutions, as in:
          HARMONY="maj7"
          INVERSION=2
          xml2ly -show-harmony-analysis "bes ${HARMONY} ${INVERSION}"
```

This is done in src/oah/harmoniesExtraOah.h/.cpp. It suffices to call function `createGlobalHarmoniesExtraOahG`

```
#ifdef EXTRA_OAH_IS_ENABLED
  // create the extra OAH group
  appendGroupToHandler (
    createGlobalHarmoniesExtraOahGroup ());
#endif
```

Macro EXTRA_OAH_IS_ENABLED is defined or not in src/oah/enableHarmoniesExtraOahIfDesired.h:

```
// comment the following definition if no extra options are wanted

#ifndef EXTRA_OAH_IS_ENABLED
  #define EXTRA_OAH_IS_ENABLED
#endif
```

16.18 man pages generation

MusicFormats can create man pages for its command line tools by browing their OAH hierarchy. This has not been finalized yet.

16.19 Specific global OAH groups

Some informations need to be available globally in the MusicFormats library, such a the conversion date and command line. They are groupeds in src/oah/generalOah.h/.cpp:

```
class EXP generalOahGroup : public oahGroup
{
  // ... ... ...

  private:

    // translation date
    // ------------------------------------

    string                fTranslationDateFull;
    string                fTranslationDateYYYYMMDD;

    // warning and error handling
    // ------------------------------------

    Bool                  fQuiet;

    Bool                  fDontShowErrors;
    Bool                  fDontQuitOnErrors;

    Bool                  fDisplaySourceCodePositions;

    // CPU usage
    // ------------------------------------

    Bool                  fDisplayCPUusage;
};
```

There are also harmonies-specific options grouped in src/oah/harmoniesExtraOah.h/.cpp. They are available as icing on the cakeicing on the cake independently of any conversion activity:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class harmoniesExtraOah.h
class EXP extraShowAllHarmoniesStructuresAtom : public oahAtomStoringAValue
class EXP extraShowAllHarmoniesContentsAtom : public oahAtomStoringAValue
class EXP extraShowHarmonyDetailsAtom : public oahAtomStoringAValue
class EXP extraShowHarmonyAnalysisAtom : public oahAtomStoringAValue
class EXP harmoniesExtraOahGroup : public oahGroup
```

```
class EXP harmoniesExtraOahGroup : public oahGroup
{
  // ... ... ...

  public:

    // fields
    // ------------------------------------------------------

    string                fHarmoniesRootAsString;
};
```

16.20 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

17.2 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.

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:

```
    // articulations
    list<S_msrArticulation>
                          fChordArticulations;

    // spanners
    list<S_msrSpanner>    fChordSpanners;

    // single tremolo
    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.

17.3 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. t doesn't contains smart-pointers to data;
- otherwise, a new description is built, sharing some some non smart-pointers fieds with the existing one. This newborn clone is then populated with whatever is needed.

For example, in src/passes/msr2lpsr/, the S_msrBarLine values found in the MSR data are used also in the LPSR data:

```
void msr2lpsrTranslator::visitStart (S_msrBarLine& elt)
{
#ifdef TRACING_IS_ENABLED
  int inputLineNumber =
    elt->getInputLineNumber ();
#endif

#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrBarLine" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  // ... ... ...

  // append the barLine to the current voice clone
  fCurrentVoiceClone->
    appendBarLineToVoice (elt);
}
```

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:

```
S_msrTuplet msrTuplet::createTupletNewbornClone ()
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceTuplets ()) {
    gLogStream <<
      "Creating a newborn clone of tuplet " <<
      asString () <<
      endl;
  }
#endif

  S_msrTuplet
    newbornClone =
      msrTuplet::create (
        fInputLineNumber,
        fMeasureElementMeasureNumber,
        fTupletNumber,
        fTupletBracketKind,
        fTupletLineShapeKind,
        fTupletShowNumberKind,
        fTupletShowTypeKind,
        fTupletFactor,
        fMemberNotesSoundingWholeNotes,
        fMemberNotesDisplayWholeNotes);

  return newbornClone;
}
```

17.4 Deep clones

Some classes in MusicFormats, such as class `msrVoice` in src/formats/msr/msrVoices.h/.cpp, have a `*DeepClone ()` method:

```
    SMARTP<msrVoice> createVoiceDeepClone (
                          int           inputLineNumber,
                          msrVoiceKind  voiceKind,
                          int           voiceNumber,
                          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:

```
class EXP msrTimeSignature : public msrMeasureElement
{
  public:

    // creation from MusicXML
    // ------------------------------------------------------

    static SMARTP<msrTimeSignature> create (
                          int                          inputLineNumber,
                          msrTimeSignatureSymbolKind timeSignatureSymbolKind);

    // creation from the applications
    // ------------------------------------------------------

    static SMARTP<msrTimeSignature> createTwoEightsTime (
                          int inputLineNumber);

    // ... ... ...

    // creation from the applications
    // ------------------------------------------------------

    static SMARTP<msrTimeSignature> createTimeFromString (
                          int    inputLineNumber,
                          string timeString);

    // ... ... ...
```

The definitions in in src/formats/msr/msrTimeSignature.cpp are:

```
S_msrTimeSignature msrTimeSignature::create (
  int                          inputLineNumber,
  msrTimeSignatureSymbolKind timeSignatureSymbolKind)
{
  msrTimeSignature* o =
    new msrTimeSignature (
      inputLineNumber, timeSignatureSymbolKind);
  assert (o != nullptr);
  return o;
}

msrTimeSignature::msrTimeSignature (
  int                          inputLineNumber,
```

```
  msrTimeSignatureSymbolKind timeSignatureSymbolKind)
    : msrMeasureElement (inputLineNumber)
{
  fTimeSignatureSymbolKind = timeSignatureSymbolKind;

  fTimeIsCompound = false;
}
```

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

```
class msdlScanner : public smartable
{
  public:

    // creation
    // ------------------------------------------------------

    static SMARTP<msdlScanner> create (istream& inputStream);

  public:

    // constructors/destructor
    // ------------------------------------------------------

                           msdlScanner (istream& inputStream);

  // ... ... ...
};
```

This leads to the following in in src/formats/msdl/msdlScanner.cpp:

```
S_msdlScanner msdlScanner::create (istream& inputStream)
{
  msdlScanner* o =
    new msdlScanner (inputStream);
  assert (o != nullptr);
  return o;
}

msdlScanner::msdlScanner (istream& inputStream)
    : fInputStream (
        inputStream),
      fCurrentToken (
        ),
      fCurrentTokenKind (
        fCurrentToken.getTokenKindToModify ()),
      fCurrentTokenDescription (
        fCurrentToken.getTokenDescriptionToModify ())
{
  // trace
#ifdef TRACING_IS_ENABLED
  fTraceTokens        = gGlobalMsdl2msrOahGroup->getTraceTokens ();
  fTraceTokensDetails = gGlobalMsdl2msrOahGroup->getTraceTokensDetails ();
#endif

  // ... ... ...
}
```

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:

```
class EXP msr2msrTranslator :

  public visitor<S_msrScore>,

  // rights

  public visitor<S_msrIdentification>,

  public visitor<S_msrCredit>,
  public visitor<S_msrCreditWords>,

    // ... ... ...
};
```

Then there are visitStart () and/or visitEnd () methods to handle the corresponding elements:

```
void msr2msrTranslator::visitStart (S_msrIdentification& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrIdentification" <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  ++gIndenter;

  // set the current identification
  fCurrentIdentification = elt;

  // store it in the resulting MSR score
  fResultingNewMsrScore->
    setIdentification (
      fCurrentIdentification);

  fOnGoingIdentification = true;
}
```

```
void msr2msrTranslator::visitEnd (S_msrIdentification& elt)
{
  fOnGoingIdentification = false;

  --gIndenter;

#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> End visiting msrIdentification" <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif
}
```

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:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxml > xml2ly -find visitors
3 occurrences of string "visitors" have been found:
   1:
    -tmxmltvis, -trace-mxsr-visitors
    Write a trace of the MusicXML tree visiting activity to standard error.
   2:
    -tmsrvis, -trace-msr-visitors
    Write a trace of the MSR graphs visiting activity to standard error.
   3:
    -tlpsrvis, -trace-lpsr-visitors
    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:

```
class EXP mfIndentedOstream: public ostream, public smartable
{
/*
Reference for this class:
  https://stackoverflow.com/questions/2212776/overload-handling-of-stdendl

Usage:
  mfIndentedOstream myStream (cout);

  myStream <<
    1 << 2 << 3 << endl <<
    5 << 6 << endl <<
    7 << 8 << endl;
*/

  public:

    // creation
    // ------------------------------------------------------

    static SMARTP<mfIndentedOstream> create (
      ostream&        theOStream,
      mfOutputIndenter& theIndenter)
    {
      mfIndentedOstream* o = new mfIndentedOstream (
        theOStream,
        theIndenter);
      assert (o != nullptr);

      return o;
    }

    // constructors/destructor
    // ------------------------------------------------------

                          mfIndentedOstream (
                            ostream&        theOStream,
                            mfOutputIndenter& theIndenter)
                            : ostream (
                                & fIndentedStreamBuf),
                              fIndentedStreamBuf (
                                theOStream,
                                theIndenter)
                              {}

    virtual               ~mfIndentedOstream () {};

```

```
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                    incrIdentation ()
62                               { ++ (fIndentedStreamBuf.getOutputIndenter ()); }
63
64     void                    decrIdentation ()
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 exception at the time of this writing is the MIDI information, see subsection 25.1.1, [MusicXML coverage], page 206;
- 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

```
%void lpsr2lilypondTranslator::generateCodeForNoteRegularInMeasure (
%  S_msrNote note)
%{
%  int inputLineNumber =
%    note->getInputLineNumber ();
%
%#ifdef TRACING_IS_ENABLED
%  if (gGlobalTracingOahGroup->getTraceNotes ()) {
%    stringstream s;
%
%    s <<
%      endl <<
%      "% --> generating code for noteRegularInMeasure " <<
%      note->asString () <<
%      ", line " << inputLineNumber <<
%      endl;
%
%    gLogStream          << s.str ();
%    fLilypondCodeStream << s.str ();
%  }
%#endif
%
%
```

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 represention], page 187, 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`:

```
    static SMARTP<msrSlur> create (
                            int                inputLineNumber,
                            int                slurNumber,
                            msrSlurTypeKind    slurTypeKind,
                            msrLineTypeKind    slurLineTypeKind,
                            msrPlacementKind   slurPlacementKind);

    // ... ... ...

    // private fields
    // ------------------------------------------------------


    int                    fSlurNumber;

    msrSlurTypeKind        fSlurTypeKind;

    msrLineTypeKind        fSlurLineTypeKind;

    msrPlacementKind       fSlurPlacementKind;
```

Other data are computed by the MSR private methods. For example, in `src/formats/msr/msrVoices.h`:

```
    // there can only be 4 regular voices in a staff
    // (those that can contain beamed notes)
    // and we need a number for the orientation of beams
    int                    fRegularVoiceStaffSequentialNumber;

    // ... ... ...

    // fVoiceShortestNoteDuration and fVoiceShortestNoteTupletFactor
    // are used to compute a number of divisions per quarter note
    // if needed, such as when generating MusicXML from MSR
    rational               fVoiceShortestNoteDuration;
    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`:

```
    rational               fCurrentMeasureWholeNotesDuration;
                             // this increases when musical elements
                             // 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:

```
    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 thoughout MSR are defined in src/formats/msr/msrBasicTypes.h/.cpp:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr >  egrep -rIn  '^// '
    msrBasicTypes.h
msrBasicTypes.h:29:// input line numbers
msrBasicTypes.h:34:// names lists max length
msrBasicTypes.h:35:// ------------------------------------------------------
msrBasicTypes.h:39:// XMLLang
msrBasicTypes.h:52:// diatonic pitches
msrBasicTypes.h:69:// alterations
msrBasicTypes.h:90:// accidentals
msrBasicTypes.h:124:// editorial accidentals
... ... ...
msrBasicTypes.h:1840:// moments
msrBasicTypes.h:1938:// tuplet factors
msrBasicTypes.h:2024:// harmonies intervals
msrBasicTypes.h:2134:// harmonies structure
msrBasicTypes.h:2231:// harmonies contents
msrBasicTypes.h:2320:// harmonies details and analysis
msrBasicTypes.h:2333:// RGB colors
msrBasicTypes.h:2391:// AlphaRGB colors
msrBasicTypes.h:2444:// score notation kinds
msrBasicTypes.h:2455:// global variables
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:

```
class EXP msrIdentification : public msrElement
{
 // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    // work
    string                 fWorkNumber;

    // ... ... ...

    // creators

    // ... ... ...

    list<string>           fSoftwaresList;

  // ... ... ...
};
```

This information is stored in distinct elements in MusicXML:

```
<score-partwise>
  <work>
    <work-number>K. 331</work-number>
    <work-title>Piano Sonata in A Major</work-title>
  </work>
  <identification>
    <creator type="composer">Wolfgang Amadeus Mozart</creator>
    <rights>Copyright © 2003 Recordare LLC</rights>
    <encoding>
      <software>Finale 2003 for Windows</software>
      <software>Dolet for Finale 1.3</software>
      <encoding-date>2003-03-14</encoding-date>
    </encoding>
```

The same occurs for MusicXML's `<direction/>` elements, that contain distinct subelements `<words/>` and `<metronome/>`:

```
      <direction>
        <direction-type>
          <words>Adagio</words>
        </direction-type>
        <direction-type>
          <metronome>
            <beat-unit>long</beat-unit>
            <per-minute>100</per-minute>
          </metronome>
        </direction-type>
      </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`:

```
<!ELEMENT direction-type (rehearsalMark+ | segno+ | coda+ |
  (words | symbol)+ | wedge | dynamics+ | dashes |
  bracket | pedal | metronome | octave-shift | harp-pedals |
  damp | damp-all | eyeglasses | string-mute |
  scordatura | image | principal-voice | percussion+ |
  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 41, [Tempos handling], page 269 and section 19.18, [Tempos], page 162 for more details.

19.3 Lengths

There are several cases where a length is used in MSR, hence:

```
enum class msrLengthUnitKind {
  kUnitInch, kUnitCentimeter, kUnitMillimeter
};
```

```
class EXP msrLength : public smartable
{
    // ... ... ...

    msrLengthUnitKind     fLengthUnitKind;
    float                 fLengthValue;
```

19.4 Positions in measures and moments

Such positions are represented by rational numbers such as `3/8`, `1/1` being a whole note.

Class `msrMoment` stores a position in a measure, with a relative offset since harmonies can be placed on a note during its sounding time:

```
    rational              fWrittenPositionInMeseasure;
    rational              fSoundingRelativeOffset;
```

19.5 Rests and skips

A skip is an invisible rest, i.e. the meaning is the same as that in LilyPond. Skips are created to it 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.

19.6 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 155 : the eighth rests in the second measure of voice 1 of the first staff should be be placed on the middle line of the staff, as `MuseScore` does.

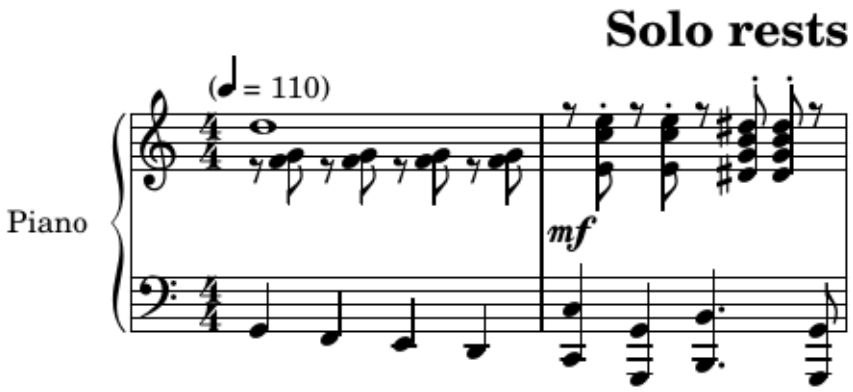


Figure 19.1: The solo rests problem

19.7 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 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/ms
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:

```
    // the measures in the slice
    vector<S_msrMeasure>  fSliceMeasuresVector;
```

From this, the following other descriptions are derived:

```
    // notes flat list
    list<S_msrNote>       fSliceNotesFlatList;

    // note events list
    list<S_msrNoteEvent>  fSliceNoteEventsList;

    // simultaneous notes chunks list
    list<S_msrSimultaneousNotesChunk>
                          fSliceSimultaneousNotesChunksList;
```

Note events are distinguished with enumeration type `msrNoteEventKind`:

```
//______________________________________________________________________
enum class msrNoteEventKind {
  kNoteEventStart,
  kNoteEventStop
};
```

Class `msrNoteEvent` contains:

```
    rational              fNoteEventPositionInMeasure;
    S_msrNote             fNoteEventNote;
    msrNoteEventKind      fNoteEventKind;
```

19.8 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:

```
// spanner types
//______________________________________________________________________________
enum class msrSpannerTypeKind {
  k_NoSpannerType,

  kSpannerTypeStart, kSpannerTypeContinue, kSpannerTypeStop
};
```

19.9 Uplinks and sidelinks

An uplink is a direct pointer from one class instance to one that contains it. class msrNote contains:

```
    S_msrChord             fNoteDirectChordUpLink;
    S_msrGraceNotesGroup   fNoteDirectGraceNotesGroupUpLink;
    S_msrTuplet            fNoteDirectTupletUpLink;
    S_msrMeasure           fNoteDirectMeasureUpLink;
```

A sidelink is used in ligatures and spanners, so that each end of the structure can reference the other one:

```
msrLigatures.h:    S_msrLigature          fLigatureOtherEndSideLink; // two-way
```

```
msrSpanners.h:    S_msrSpanner           fSpannerOtherEndSideLink; // two-way
```

19.10 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:

```
    rational               fMeasureElementSoundingWholeNotes;
```

In a tuplet, the sounding durations is different than the written durations, so we store the sounding

```
    // whole notes
    rational               fNoteDisplayWholeNotes;

    int                    fNoteDotsNumber;

    msrDurationKind        fNoteGraphicDurationKind;

    msrTupletFactor        fNoteTupletFactor;

    msrQuarterTonesPitchKind
                           fNoteQuarterTonesDisplayPitchKind;
    msrOctaveKind          fNoteDisplayOctaveKind;
                                 // for unpitched notes
                                 // and pitched rests
```

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:

```
    void                    print (ostream& os) const override;

    string                  asString () const override;
    string                  asStringShort () const override;
```

There are also more specific methods such as:

```
    void                    printShort (ostream& os) const override;

    void                    printSummary (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:

```
ostream& operator<< (ostream& os, const S_msrElement& elt)
{
  elt->print (os);
  return os;
}
```

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

```
void msrElement::print (ostream& os) const
{
  os << asString () << endl;
}
```

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:

```
string msrTranspose::asString () const
{
  stringstream s;

  s <<
    "[Transpose" <<
    ", diatonic = " << fTransposeDiatonic <<
    ", chromatic = " << fTransposeChromatic <<
    ", transposeOctaveChange = " << fTransposeOctaveChange <<
    ", transposeDouble = " << fTransposeDouble <<
    ", line " << fInputLineNumber <<
    "]";

  return s.str ();
}
```

A typical sequence to produce indented output is:

```
void msrTranspose::print (ostream& os) const
{
  const int fieldWidth = 22;

  os <<
    "Transpose" <<
    ", line " << fInputLineNumber <<
    endl;

  ++gIndenter;

  os << left <<
    setw (fieldWidth) <<
    "transposeDiatonic" << " = " << fTransposeDiatonic <<
    endl <<
    setw (fieldWidth) <<
    "transposeChromatic" << " = " << fTransposeChromatic <<
    endl <<
    setw (fieldWidth) <<
    "transposeOctaveChange" << " = " << fTransposeOctaveChange <<
    endl <<
    setw (fieldWidth) <<
    "transposeDouble" << " = " << fTransposeDouble <<
    endl << endl;

  --gIndenter;
}
```

The main indented output streams are:

```
#define gOutputStream *gGlobalOutputIndentedOstream
#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 quater tones pitches, and conversions are done wherever needed.

```
// diatonic pitches
//_______________________________________________________________________________
enum class msrDiatonicPitchKind {
  k_NoDiatonicPitch,

  // starting at C for LilyPond relative octave calculations
  kDiatonicPitchC,
  kDiatonicPitchD, kDiatonicPitchE, kDiatonicPitchF,
  kDiatonicPitchG, kDiatonicPitchA, kDiatonicPitchB
};
```

```
// semi tones pitches
//_______________________________________________________________________________
enum class msrSemiTonesPitchKind {
  kSTP_NoSemiTonesPitch,

  kSTP_C_TripleFlat,
  kSTP_C_DoubleFlat, kSTP_C_Flat,
  kSTP_C_Natural,
  kSTP_C_Sharp, kSTP_C_DoubleSharp,
  kSTP_C_TripleSharp,

```

```
  kSTP_D_TripleFlat,
  kSTP_D_DoubleFlat, kSTP_D_Flat,
  kSTP_D_Natural,
  kSTP_D_Sharp, kSTP_D_DoubleSharp,
  kSTP_D_TripleSharp,

  kSTP_E_TripleFlat,
  kSTP_E_DoubleFlat, kSTP_E_Flat,
  kSTP_E_Natural,
  kSTP_E_Sharp, kSTP_E_DoubleSharp,
  kSTP_E_TripleSharp,

  kSTP_F_TripleFlat,
  kSTP_F_DoubleleFlat, kSTP_F_Flat,
  kSTP_F_Natural,
  kSTP_F_Sharp, kSTP_F_DoubleSharp,
  kSTP_F_TripleSharp,

  kSTP_G_TripleFlat,
  kSTP_G_DoubleFlat, kSTP_G_Flat,
  kSTP_G_Natural,
  kSTP_G_Sharp, kSTP_G_DoubleSharp,
  kSTP_G_TripleSharp,

  kSTP_A_TripleFlat,
  kSTP_A_DoubleFlat, kSTP_A_Flat,
  kSTP_A_Natural,
  kSTP_A_Sharp, kSTP_A_DoubleSharp,
  kSTP_A_TripleSharp,

  kSTP_B_TripleFlat,
  kSTP_B_DoubleFlat, kSTP_B_Flat,
  kSTP_B_Natural,
  kSTP_B_Sharp, kSTP_B_DoubleSharp,
  kSTP_B_TripleSharp
};
```

```
// quarter tones pitches
//______________________________________________________________________________
enum class msrQuarterTonesPitchKind {
  k_NoQuarterTonesPitch,

  kQTP_Rest, kQTP_Skip,

  kQTP_A_TripleFlat,
  kQTP_A_DoubleFlat, kQTP_A_SesquiFlat, kQTP_A_Flat, kQTP_A_SemiFlat,
  kQTP_A_Natural,
  kQTP_A_SemiSharp, kQTP_A_Sharp, kQTP_A_SesquiSharp, kQTP_A_DoubleSharp,
  kQTP_A_TripleSharp,

  kQTP_B_TripleFlat,
  kQTP_B_DoubleFlat, kQTP_B_SesquiFlat, kQTP_B_Flat, kQTP_B_SemiFlat,
  kQTP_B_Natural,
  kQTP_B_SemiSharp, kQTP_B_Sharp, kQTP_B_SesquiSharp, kQTP_B_DoubleSharp,
  kQTP_B_TripleSharp,

  kQTP_C_TripleFlat,
  kQTP_C_DoubleFlat, kQTP_C_SesquiFlat, kQTP_C_Flat, kQTP_C_SemiFlat,
  kQTP_C_Natural,
  kQTP_C_SemiSharp, kQTP_C_Sharp, kQTP_C_SesquiSharp, kQTP_C_DoubleSharp,
  kQTP_C_TripleSharp,

  kQTP_D_TripleFlat,
  kQTP_D_DoubleFlat, kQTP_D_SesquiFlat, kQTP_D_Flat, kQTP_D_SemiFlat,
  kQTP_D_Natural,
  kQTP_D_SemiSharp, kQTP_D_Sharp, kQTP_D_SesquiSharp, kQTP_D_DoubleSharp,
```

```
  kQTP_D_TripleSharp,

  kQTP_E_TripleFlat,
  kQTP_E_DoubleFlat, kQTP_E_SesquiFlat, kQTP_E_Flat, kQTP_E_SemiFlat,
  kQTP_E_Natural,
  kQTP_E_SemiSharp, kQTP_E_Sharp, kQTP_E_SesquiSharp, kQTP_E_DoubleSharp,
  kQTP_E_TripleSharp,

  kQTP_F_TripleFlat,
  kQTP_F_DoubleFlat, kQTP_F_SesquiFlat, kQTP_F_Flat, kQTP_F_SemiFlat,
  kQTP_F_Natural,
  kQTP_F_SemiSharp, kQTP_F_Sharp, kQTP_F_SesquiSharp, kQTP_F_DoubleSharp,
  kQTP_F_TripleSharp,

  kQTP_G_TripleFlat,
  kQTP_G_DoubleFlat, kQTP_G_SesquiFlat, kQTP_G_Flat, kQTP_G_SemiFlat,
  kQTP_G_Natural,
  kQTP_G_SemiSharp, kQTP_G_Sharp, kQTP_G_SesquiSharp, kQTP_G_DoubleSharp,
  kQTP_G_TripleSharp
};
```

19.13 Octaves

They are represented with enumeration type :

```
// octaves
//_______________________________________________________________________
enum class msrOctaveKind {
  k_NoOctave,

  kOctave0, kOctave1, kOctave2, kOctave3,
  kOctave4, // that of middle C
  kOctave5, kOctave6, kOctave7, kOctave8, kOctave9
};
```

For the needs of LilyPond and MSDL, MSR also contains a description of how to enter octaves:

```
// octave entry
//_______________________________________________________________________
enum class msrOctaveEntryKind {
  kOctaveEntryRelative,
  kOctaveEntryAbsolute,
  kOctaveEntryFixed
};
```

19.14 Durations

MusicFormats represents durations with enumeration type `msrDurationKind`, defined in `src/formats/msr/msrBasic`

```
// durations
//_______________________________________________________________________
enum class msrDurationKind {
  k_NoDuration,

  // from longest to shortest for the algorithms
  kMaxima, kLonga, kBreve, kWhole, kHalf,
  kQuarter,
  kEighth, k16th, k32nd, k64th, k128th, k256th, k512th, k1024th
};
```

19.15 Alterations

```
// alterations
//______________________________________________________________________________
enum class msrAlterationKind {
  k_NoAlteration,

  kAlterationTripleFlat, kAlterationDoubleFlat, kAlterationSesquiFlat,
  kAlterationFlat, kAlterationSemiFlat,
  kAlterationNatural,
  kAlterationSemiSharp, kAlterationSharp, kAlterationSesquiSharp,
  kAlterationDoubleSharp, kAlterationTripleSharp
```

19.16 Accidentals

```
// accidentals
//______________________________________________________________________________
enum class msrAccidentalKind {
  kAccidentalNone,

  kAccidentalSharp, kAccidentalNatural,
  kAccidentalFlat, kAccidentalDoubleSharp,
  kAccidentalSharpSharp,
  kAccidentalFlatFlat, kAccidentalNaturalSharp,
  kAccidentalNaturalFlat, kAccidentalQuarterFlat,
  kAccidentalQuarterSharp,kAccidentalThreeQuartersFlat,
  kAccidentalThreeQuartersSharp,

  kAccidentalSharpDown, kAccidentalSharpUp,
  kAccidentalNaturalDown, kAccidentalNaturalUp,
  kAccidentalFlatDown, kAccidentalFlatUp,
  kAccidentalTripleSharp, kAccidentalTripleFlat,
  kAccidentalSlashQuarterSharp, kAccidentalSlashSharp,
  kAccidentalSlashFlat, kAccidentalDoubleSlashFlat,
  kAccidentalSharp_1, kAccidentalSharp_2,
  kAccidentalSharp_3,  kAccidentalSharp_5,
  kAccidentalFlat_1, kAccidentalFlat_2,
  kAccidentalFlat_3, kAccidentalFlat_4,
  kAccidentalSori, kAccidentalKoron,

  kAccidentalOther
};
```

19.17 Durations

They are represented in MSR with the enumeration type `msrDurationKind` enumeration type, defined in src/formats/msr/msrBasicTypes.h:

```
// durations
//______________________________________________________________________________
enum class msrDurationKind {
  k_NoDuration,

  // from longest to shortest for the algorithms
  kMaxima, kLonga, kBreve, kWhole, kHalf,
  kQuarter,
  kEighth, k16th, k32nd, k64th, k128th, k256th, k512th, k1024th
};
```

Class `msrDottedDuration` contains:

```
    msrDurationKind        fDurationKind;
    int                    fDotsNumber;
```

19.18 Tempos

There are thus several kinds of tempos in MSR, with variants represented by enumeration type `msrTempoKind` in src/formats/msr/msrTempos.h:

```
class EXP msrTempo : public msrMeasureElement
{
  public:

    // data types
    // ------------------------------------------------------

    enum msrTempoKind {
      k_NoTempoKind,
      kTempoBeatUnitsWordsOnly,
      kTempoBeatUnitsPerMinute,
      kTempoBeatUnitsEquivalence,
      kTempoNotesRelationship
    };

    // ... ... ...

    enum msrTempoParenthesizedKind {
      kTempoParenthesizedYes, kTempoParenthesizedNo
    };

    // ... ... ...

    enum msrTempoNotesRelationshipKind {
      kTempoNotesRelationshipNone, kTempoNotesRelationshipEquals
    };

    // ... ... ...
};
```

19.18.1 Tempos notes

A tempo indication can contain a note a notes in a tuplet. Such notes are described by class `msrTempoNote`:

```
class EXP msrTempoNote : public msrElement
{
  public:

    // creation from MusicXML
    // ------------------------------------------------------

    static SMARTP<msrTempoNote> create (
                            int              inputLineNumber,
                            const rational&  tempoNoteWholeNotes,
                            Bool             tempoNoteBelongsToATuplet);

  protected:

    // constructors/destructor
    // ------------------------------------------------------

```

```
                          msrTempoNote (
                            int             inputLineNumber,
                            const rational& tempoNoteWholeNotes,
                            Bool            tempoNoteBelongsToATuplet);

    // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    rational               fTempoNoteWholeNotes;

    list<S_msrBeam>        fTempoNoteBeams;

    Bool                   fTempoNoteBelongsToATuplet;
};
```

19.18.2 Tempos tuplets

A tuplet in a tempo representation is described by class `msrTempoTuplet`:

```
//______________________________________________________________________________
class EXP msrTempoTuplet : public msrElement
{
  public:

    // data types
    // ------------------------------------------------------

    enum msrTempoTupletTypeKind {
      kTempoTupletTypeNone,
      kTempoTupletTypeStart, kTempoTupletTypeStop
    };

    // ... ... ...

    enum msrTempoTupletBracketKind {
      kTempoTupletBracketYes, kTempoTupletBracketNo
    };

    // ... ... ...

    enum msrTempoTupletShowNumberKind {
      kTempoTupletShowNumberActual,
      kTempoTupletShowNumberBoth,
      kTempoTupletShowNumberNone
    };

    // ... ... ...

    // creation from MusicXML
    // ------------------------------------------------------

    static SMARTP<msrTempoTuplet> create (
      int                          inputLineNumber,
      int                          tempoTupletNumber,
      msrTempoTupletBracketKind    tempoTupletBracketKind,
      msrTempoTupletShowNumberKind tempoTupletShowNumberKind,
      msrTupletFactor              tempoTupletFactor,
      rational                     memberNotesDisplayWholeNotes);

  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      list<S_msrElement>      fTempoTupletElements;
76  };
```

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      msrTempoKind            fTempoKind;
11
12      list<S_msrWords>        fTempoWordsList;
13
14      msrDottedDuration       fTempoBeatUnit;
15
16      string                  fTempoPerMinute; // '90' or '132-156' for example
17      msrDottedDuration       fTempoEquivalentBeatUnit;
18
19      S_msrTempoNotesRelationshipshipElements
20                              fTempoNotesRelationshipLeftElements;
21      msrTempoNotesRelationshipKind  fTempoNotesRelationshipKind;
22      S_msrTempoNotesRelationshipshipElements
23                              fTempoNotesRelationshipRightElements;
24
```

```
    msrTempoParenthesizedKind
                            fTempoParenthesizedKind;

    msrPlacementKind        fTempoPlacementKind;
};
```

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 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::kTemp`
- field `msrTempo::fTempoParenthesizedKind` indicates whether the tempo indication is parenthesized;
- field `msrTempo::fTempoPlacementKind` tells whether the tempo is to be places above or below the staff, constant `msrPlacementKind::kPlacementAbove` by default.

19.19 Clefs

Clefs are distinguished using enumeration type `msrClefKind`:

```
// clefs
//______________________________________________________________________________

enum class msrClefKind {
  k_NoClef,

  kClefTreble,
  kClefSoprano, kClefMezzoSoprano, kClefAlto, kClefTenor, kClefBaritone, kClefBass,
  kClefTrebleLine1,
  kClefTrebleMinus15, kClefTrebleMinus8, kClefTreblePlus8, kClefTreblePlus15,

  kClefBassMinus15, kClefBassMinus8, kClefBassPlus8, kClefBassPlus15,

  kClefVarbaritone,

  kClefTablature4, kClefTablature5, kClefTablature6, kClefTablature7,

  kClefPercussion,

  kClefJianpu
};
```

Class `msrClef` contains:

```
    msrClefKind             fClefKind;
    int                     fClefStaffNumber;
```

19.20 Keys

MSR, as MusicXML, suports 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:

```
    msrDiatonicPitchKind   fKeyDiatonicPitchKind;
    msrAlterationKind      fKeyAlterationKind;
    msrOctaveKind          fKeyOctaveKind;
```

An example is at figure 19.2, [Humdrum-Scot keys], page 166. It has been produced by:

```
xml2ly -auto-output-file-name keys/HumdrumScotKeys.xml
```

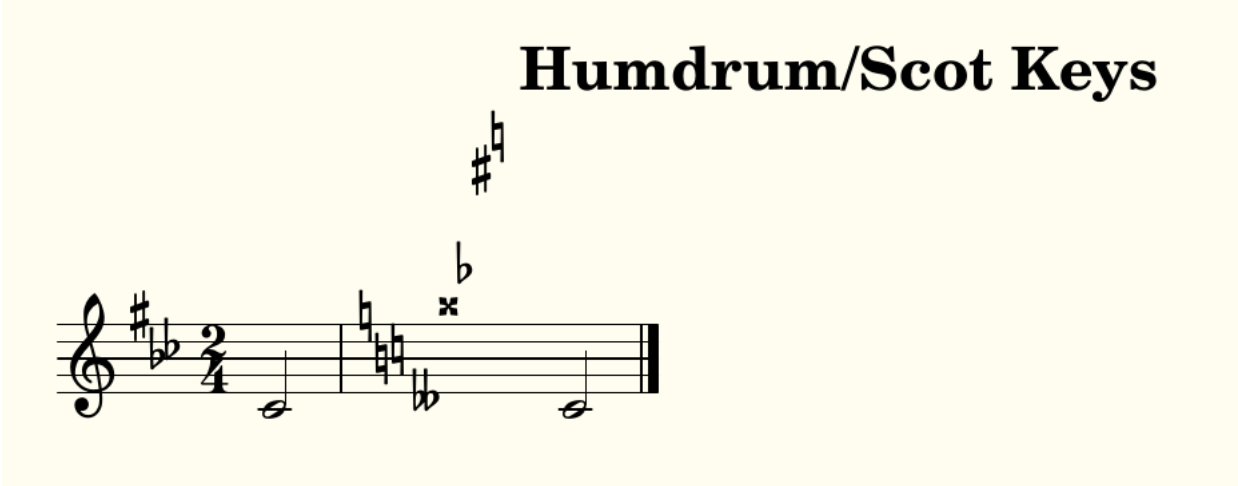


Figure 19.2: Humdrum-Scot keys

Class `msrKey` thus contains:

```
    // private fields
    // ------------------------------------------------------

    msrKeyKind             fKeyKind;

    // traditional keys

    msrQuarterTonesPitchKind
                           fKeyTonicQuarterTonesPitchKind;
    msrModeKind            fModeKind;
    int                    fKeyCancel;

    // Humdrum/Scot keys

    vector<S_msrHumdrumScotKeyItem>
                           fHumdrumScotKeyItemsVector;
    Bool                   fKeyItemsOctavesAreSpecified;
```

19.21 Time signatures

The variants in time signatures are distinguished by enumeration type enumeration type `msrTimeSignatureSymbolKi`

```
// time symbols
//______________________________________________________________________________
enum class msrTimeSignatureSymbolKind {
  kTimeSignatureSymbolNone,
  kTimeSignatureSymbolCommon,
  kTimeSignatureSymbolCut,
  kTimeSignatureSymbolNote,
  kTimeSignatureSymbolDottedNote,
  kTimeSignatureSymbolSingleNumber,
  kTimeSignatureSymbolSenzaMisura
};
```

A time signature can also be structured, and this is described by those two types:

```
enum class msrTimeSignatureSeparatorKind {
  kTimeSignatureSeparatorNone,
  kTimeSignatureSeparatorHorizontal,
  kTimeSignatureSeparatorDiagonal,
  kTimeSignatureSeparatorVertical,
  kTimeSignatureSeparatorAdjacent
};
```

```
enum class msrTimeSignatureRelationKind {
  kTimeSignatureRelationNone,
  kTimeSignatureRelationParentheses,
  kTimeSignatureRelationBracket,
  kTimeSignatureRelationEquals,
  kTimeSignatureRelationSlash,
  kTimeSignatureRelationSpace,
  kTimeSignatureRelationHyphen
};
```

A brick that can be used in class `msrTimeSignature` is `msrTimeSignatureItem`, whose private fields are:

```
    vector<int>           fTimeSignatureBeatsNumbersVector; // 5+3+1 is possible
    int                   fTimeSignatureBeatValue;
```

Class `msrTimeSignature` contains:

```
    msrTimeSignatureSymbolKind
                          fTimeSignatureSymbolKind;

    vector<S_msrTimeSignatureItem>
                          fTimeSignatureItemsVector;

    // a time is compound if it contains several items
    // or if the only one has several beats numbers
    // i.e. 3/4 is not, (3+4)/8 is, and 2/4+3/4 is too
    Bool                  fTimeIsCompound;
```

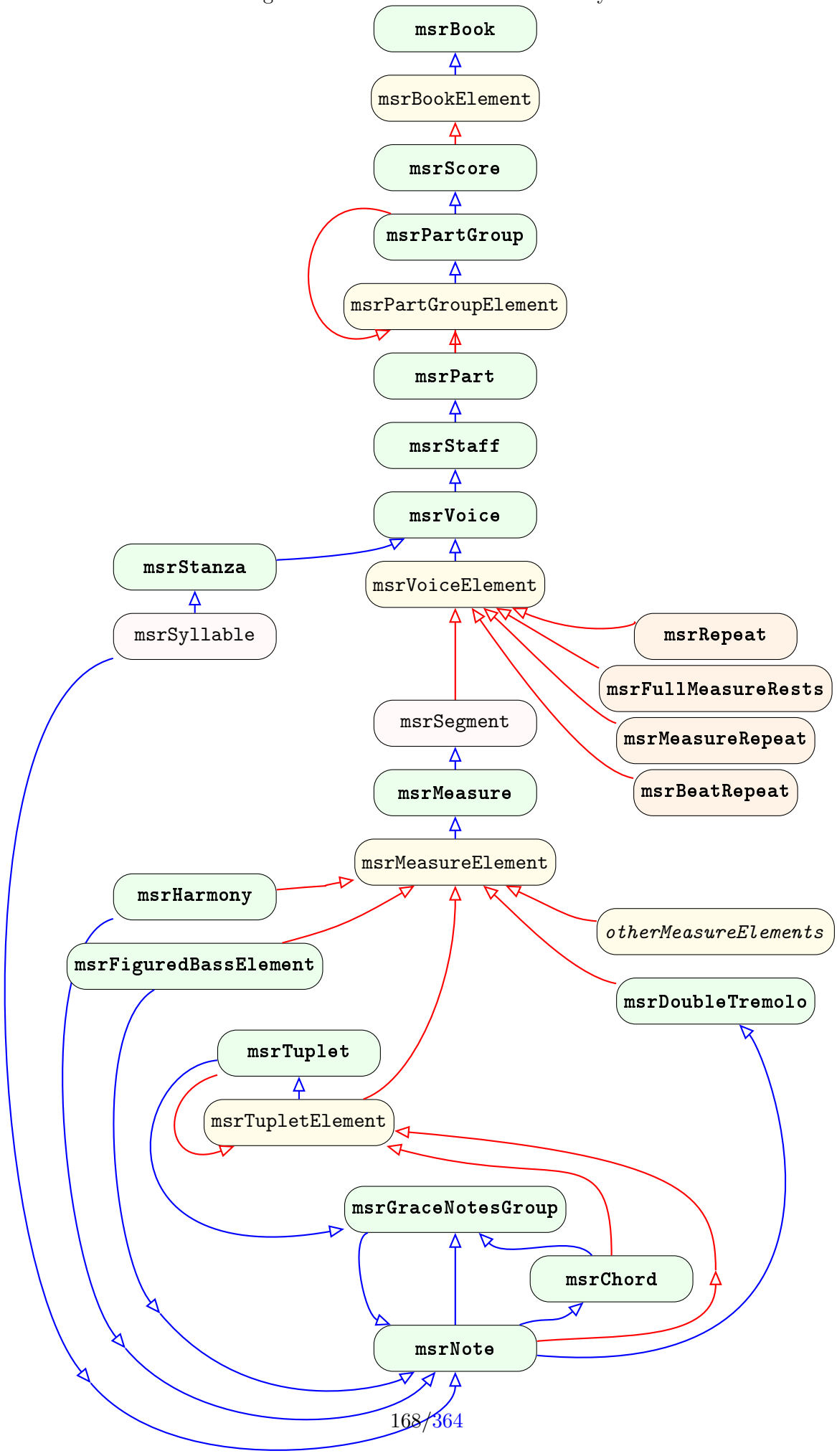
19.22 MSR classes inheritance

The picture at figure 19.3, [The MSR classes hierarchy], page 168, 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`;

Figure 19.3: The MSR classes hierarchy



- 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`;
- blue: a link from a class to another that uses smart pointers to one or more instances of the former. For example, an class `msrTuplet` may be an element of an class `msrGraceNotesGroup`.

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:
  - 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 `msrTranspose`
  - class `msrTupletElement`

19.23 Books

Books handling is presented at section 53, [Books handling], page 291.

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`:

```
    // book elements
    set<S_msrBookElement> fBookElementsSet;

    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 {...}` :

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrBook' *
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`:

```
    // part groups
    set<S_msrPartGroup>   fScorePartGroupsSet;

    list<S_msrPartGroup>  fPartGroupsList;
```

19.25 Part groups

Part groups handling is presented at section 19.25, [Part groups], page 171.

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:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrPartGroupElement' *
formats/msr/msrParts.h:class EXP msrPart : public msrPartGroupElement
formats/msr/msrPartGroups.h:class EXP msrPartGroup : public msrPartGroupElement
```

which are stored in a list:

```
    // allowing for both parts and (sub-)part groups as elements
    list<S_msrPartGroupElement>
                          fPartGroupElementsList;
```

19.26 Parts

Parts handling is presented at section 51, [Parts handling], page 288.

A part in MSR is composed of voices, stored in:

```
    // staves

    map<int, S_msrStaff>  getPartStaveNumbersToStavesMap;
    list<S_msrStaff>      fPartAllStavesList;

    // harmonies

    S_msrStaff            fPartHarmoniesStaff;
    S_msrVoice            fPartHarmoniesVoice;

    // figured bass

    S_msrStaff            fPartFiguredBassStaff;
    S_msrVoice            fPartFiguredBassVoice;

    // voices

    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:

```
    // the mapping of all the voices in the staff,
    // including harmonies and figured bass voices
    map<int, S_msrVoice>  fStaffVoiceNumbersToAllVoicesMap;

    // the mapping of voice numbers to regular voices
    map<int, S_msrVoice>  fStaffVoiceNumbersToRegularVoicesMap;

    // we need to handle the regular voice specifically
    // to assign them sequencing numbers from 1 to gMaxStaffVoices,
    // needed to set the beams orientation (up or down)
    int                   fStaffRegularVoicesCounter;

    // harmonies and figured bass elements should be placed %%%JMI
    // in the first regular voice of the staff, hence:
    list<S_msrVoice>      fStaffRegularVoicesList;

    // we need to sort the voices by increasing voice numbers,
    // but with harmonies voices right before the corresponding regular voices
    list<S_msrVoice>      fStaffAllVoicesList;
```

19.28 Voice elements

Voices contain instances of class `msrVoiceElement`, defined in src/formats/msr/msrVoiceElements.h/.cpp:

```
//______________________________________________________________________________
/*
  Various elements can found in voices,
  hence class msrVoiceElement
*/

class EXP msrVoiceElement : public msrElement
{
  public:

    // creation from MusicXML
    // ------------------------------------------------------

    // cloning
    // ------------------------------------------------------

  protected:

                          msrVoiceElement (
                            int inputLineNumber);

    virtual               ~msrVoiceElement ();

  /*
    The voice uplink is declared in the sub-classes,
    to allow for separate *.h files, C++ constraint
  */
};
```

The classes derived from class `msrVoiceElement` are:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'public msrVoiceElement
    ' *.h
msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
msrFullMeasureRests.h:class EXP msrFullMeasureRests : public msrVoiceElement
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 286.

A voice is conceptually a sequence of `S_msrVoiceElement`, that may be:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrVoiceElement' *
formats/msr/msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
formats/msr/msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
formats/msr/msrFullMeasureRests.h:class EXP msrFullMeasureRests : public msrVoiceElement
formats/msr/msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
formats/msr/msrSegments.h:class EXP msrSegment : public msrVoiceElement
```

More precisely and for technical reasons, an class `msrVoice` contains:

```
    // voice initial elements list

    list<S_msrVoiceElement>
                          fVoiceInitialElementsList;

    // voice first and last segments

    // fVoiceLastSegment contains the music
    // not yet stored in fVoiceInitialElementsList,
    // it is thus logically the end of the latter,
    // and is created implicitly for every voice.
    // It is needed 'outside' of the 'list<S_msrElement>'
    // because it is not a mere S_msrElement, but a S_msrSegment
    S_msrSegment          fVoiceLastSegment;

    // fVoiceFirstSegment is used to work around LilyPond issue #34
    S_msrSegment          fVoiceFirstSegment;
```

Each voice is described by a field of enumeration type `msrVoiceKind`, defined in src/formats/msr/msrBasicTypes.h:

```
enum class msrVoiceKind {
  kVoiceKindRegular,
  kVoiceKindDynamics,
  kVoiceKindHarmonies,  // for MusicXML <harmony/>, LilyPond ChordNames
  kVoiceKindFiguredBass // for MusicXML <figured-bass/>, LilyPond FiguredBass
};
```

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::fVoiceLastS` is moved into it and a new segment is created, see section 19.36, [Repeats], page 179.

Wether 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:

```
enum class msrVoiceCreateInitialLastSegmentKind {
  kCreateInitialLastSegmentYes,
  kCreateInitialLastSegmentNo
};
```

19.30 Measures

Measures handling is presented at section 38, [Measures handling], page 243.

A measure is a linear, flat sequence of class `msrMeasureElements`, some of which are structured, such as class `msrChord`. Class `msrMeausre` is defined in src/formats/msr/msrMeausre.h/.cpp.

The measure elements are defined in src/formats/msr/:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep  'public
    msrMeasureElement' *.h
msrBars.h:class EXP msrBarCheck : public msrMeasureElement
msrBars.h:class EXP msrBarNumberCheck : public msrMeasureElement
msrBars.h:class EXP msrBarLine : public msrMeasureElement
msrBreaks.h:class EXP msrLineBreak : public msrMeasureElement
msrBreaks.h:class EXP msrPageBreak : public msrMeasureElement
msrClefs.h:class EXP msrClef : public msrMeasureElement
msrCodas.h:class EXP msrCoda : public msrMeasureElement
msrDoubleTremolos.h:class EXP msrDoubleTremolo : public msrMeasureElement
msrEyeGlasses.h:class EXP msrEyeGlasses : public msrMeasureElement
msrFiguredBassElements.h:class EXP msrFiguredBassElement : public msrMeasureElement
msrHarmonies.h:class EXP msrHarmony : public msrMeasureElement
msrHiddenMeasureAndBarLines.h:class EXP msrHiddenMeasureAndBarLine : public
    msrMeasureElement
msrInstruments.h:class EXP msrScordatura : public msrMeasureElement
msrInstruments.h:class EXP msrAccordionRegistration : public msrMeasureElement
msrInstruments.h:class EXP msrHarpPedalsTuning : public msrMeasureElement
msrInstruments.h:class EXP msrPedal : public msrMeasureElement
msrInstruments.h:class EXP msrDamp : public msrMeasureElement
msrInstruments.h:class EXP msrDampAll : public msrMeasureElement
msrKeys.h:class EXP msrKey : public msrMeasureElement
msrLyrics.h:class EXP msrSyllable : public msrMeasureElement
msrMusicXMLSpecifics.h:class EXP msrPrintLayout : public msrMeasureElement
msrRehearsalMarks.h:class EXP msrRehearsalMark : public msrMeasureElement
msrSegnos.h:class EXP msrSegno : public msrMeasureElement
msrSegnos.h:class EXP msrDalSegno : public msrMeasureElement
msrStavesDetails.h:class EXP msrStaffDetails : public msrMeasureElement
msrTempos.h:class EXP msrTempo : public msrMeasureElement
msrTimeSignatures.h:class EXP msrTimeSignature : public msrMeasureElement
msrTranspositions.h:class EXP msrOctaveShift : public msrMeasureElement
msrTranspositions.h:class EXP msrTranspose : public msrMeasureElement
msrTupletElements.h:class EXP msrTupletElement : public msrMeasureElement
msrVoiceStaffChanges.h:class EXP msrVoiceStaffChange : public msrMeasureElement
```

In order to perform a time-wise analysis of the scores, MSR contains class `msrMeasusre` 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:

```
S_msrMeasure msrSegment::createAMeasureAndAppendItToSegment (
  int     inputLineNumber,
  string measureNumber,
  msrMeasureImplicitKind
          measureImplicitKind)
{
  // ... ... ...

  ++gIndenter;

  // determine new measure 'first in segment' kind
```

```
  msrMeasure::msrMeasureFirstInSegmentKind
    measureFirstInSegmentKind;

  if (fSegmentMeasuresList.size () == 0) {
    // this is the first measure in the segment
    measureFirstInSegmentKind =
      msrMeasure::kMeasureFirstInSegmentKindYes;
  }
  else {
    // this is not the first measure in the segment
    measureFirstInSegmentKind =
      msrMeasure::kMeasureFirstInSegmentKindNo;
  }

  // create a measure
  // ... ... ...

  S_msrMeasure
    result =
      msrMeasure::create (
        inputLineNumber,
        measureNumber,
        this);

  // set result's ordinal number
  result->
    setMeasureOrdinalNumberInVoice (
      fSegmentVoiceUpLink->
        incrementVoiceCurrentMeasureOrdinalNumber ());

  // append result to the segment
  appendMeasureToSegment (result);

  --gIndenter;

  return result;
}
```

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:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep Pattern *.h | grep
    class
msrBeatRepeats.h:class EXP msrBeatRepeatPattern : public msrElement
msrMeasureRepeats.h:class EXP msrMeasureRepeatPattern : public msrElement
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep Replicas *.h | grep
    class
msrBeatRepeats.h:class EXP msrBeatRepeatReplicas : public msrElement
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 277.

Class `msrBeatRepeat`, defined in `src/formats/msr/msrBeatRepeats.h/.cpp`, contains a pattern and replicas:

```
class EXP msrBeatRepeat : public msrVoiceElement
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    // upLinks
    S_msrVoice            fBeatRepeatVoiceUpLink;

    // numbers
    int                   fBeatRepeatMeasuresNumber;
    int                   fBeatRepeatSlashesNumber;

    // measures repeat pattern
    S_msrBeatRepeatPattern
                          fBeatRepeatPattern;

    // measures repeat replicas
    S_msrBeatRepeatReplicas
                          fBeatRepeatReplicas;

    // measures repeat build phase, used when building the measures repeat
    msrBeatRepeatBuildPhaseKind
                          fCurrentBeatRepeatBuildPhaseKind; // unused??? JMI
};
```

Class `msrBeatRepeatPattern` contains a segment and an uplink:

```
class EXP msrBeatRepeatPattern : public msrElement
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    // upLinks
    S_msrBeatRepeat      fBeatRepeatUpLink;

    // segment
    S_msrSegment          fBeatRepeatPatternSegment;
 };
```

Class `msrBeatRepeatReplicas` contains a segment and an uplink:

```
class EXP msrBeatRepeatReplicas : public msrElement
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

```

```
    // upLinks
    S_msrBeatRepeat      fBeatRepeatUpLink;

    // segment
    S_msrSegment          fBeatRepeatReplicasSegment;
};
```

19.33 Measure repeats

Measure repeats handling is presented at section 46, [Measure repeats handling], page 278.

Class `msrMeasureRepeat`, defined in `src/formats/msr/msrMeasureRepeat.h/.cpp`, contains a pattern and replicas:

```
class EXP msrMeasureRepeat : public msrVoiceElement
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    // upLinks
    S_msrVoice            fMeasureRepeatVoiceUpLink;

    // numbers
    int                   fMeasureRepeatMeasuresNumber;
    int                   fMeasureRepeatSlashesNumber;

    // measures repeat pattern
    S_msrMeasureRepeatPattern
                          fMeasureRepeatPattern;

    // measures repeat replicas
    S_msrMeasureRepeatReplicas
                          fMeasureRepeatReplicas;

    // measures repeat build phase, used when building the measures repeat
    msrMeasureRepeatBuildPhaseKind
                          fCurrentMeasureRepeatBuildPhaseKind;
};
```

Class `msrMeasureRepeatPattern` contains a segment and an uplink:

```
class EXP msrMeasureRepeatPattern : public msrElement
{
  // ... ... ...

 private:

    // private fields
    // ------------------------------------------------------

    // upLinks
    S_msrMeasureRepeat   fMeasureRepeatUpLink;

    // segment
    S_msrSegment          fMeasureRepeatPatternSegment;
};
```

Class `msrMeasureRepeatReplicas` contain a segment and an uplink:

```
class EXP msrMeasureRepeatReplicas : public msrElement
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    // upLinks
    S_msrMeasureRepeat   fMeasureRepeatUpLink;

    // segment
    S_msrSegment          fMeasureRepeatReplicasSegment;
};
```

19.34 Full measure rests

Full measure rests handling is presented at section 65, [Full measure rests handling], page 331.

Class `msrFullMeasureRests`, defined in `src/formats/msr/msrFullMeasureRests.h/.cpp`, essentially contains a class `msrFullMeasureRestsContents` instance and a mesure rests number:

```
class EXP msrFullMeasureRests : public msrVoiceElement
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    S_msrVoice            fFullMeasureRestsVoiceUpLink;

    int                   fFullMeasureRestsNumber;

    S_msrFullMeasureRestsContents
                          fFullMeasureRestsContents;

    string                fFullMeasureRestsNextMeasureNumber;
    int                   fFullMeasureRestsLastMeasurePuristNumber;

    // shortcut for efficiency
    rational              fFullMeasureRestsMeasureSoundingNotes;
};
```

Class `msrFullMeasureRestsContents` contains an class `msrSegment` instance and an uplink:

```
class EXP msrFullMeasureRestsContents : public msrElement
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    // upLink
    S_msrFullMeasureRests fFullMeasureRestsContentsFullMeasureRestsUpLink;

    S_msrSegment          fFullMeasureRestsContentsSegment;
};
```

19.35 Barlines

19.36 Repeats

Repeats handling is presented at section 48, [Repeats handling], page 280.

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:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep class msrRepeats.h
class msrRepeat;
class msrFullMeasureRests;
class msrMeasureRepeat;
class msrNote;
class EXP msrRepeatCommonPart : public msrElement
class EXP msrRepeatEnding : public msrElement
class EXP msrRepeat : public msrVoiceElement
class EXP msrRepeatDescr : public smartable
class EXP msrRepeatElement : public msrElement
```

Class `msrRepeat`, defined in msrBothmsrRepeats, contains an class `msrRepeatCommonPart`, followed by zero or more instances of class `msrRepeatEnding`:

```
class EXP msrRepeat : public msrVoiceElement
{
  public:

    // data types
    // ------------------------------------------------------

    enum msrRepeatExplicitStartKind {
      kRepeatExplicitStartNo,
      kRepeatExplicitStartYes
    };

    // ... ... ...

    // common part
    S_msrRepeatCommonPart fRepeatCommonPart;

    // repeat endings
    vector<S_msrRepeatEnding>
                          fRepeatEndings;
    int                   fRepeatEndingsInternalCounter;

    // immediately preceding and following repeats
    // detecting several repeats in a row helps LilyPond code generation
    // depending on the options JMI
    S_msrRepeat           fImmediatelyPrecedingRepeat;
    S_msrRepeat           fImmediatelyFollowingRepeat;
};
```

Class `msrRepeatCommonPart` contains a list of class `msrVoiceElement`:

```
  private:

    // private fields
    // ------------------------------------------------------

    // upLinks
    S_msrRepeat           fRepeatCommonPartRepeatUpLink;

    // elements list
```

```
    list<S_msrVoiceElement>
                            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:

```
enum class msrRepeatEndingKind {
  kRepeatEndingHooked,
  kRepeatEndingHookless
};
```

Class `msrRepeatEnding` contains a list of class `msrVoiceElement` too, as well as a enumeration type `msrRepeatEndingKind` field:

```
  private:

    // private fields
    // ------------------------------------------------------

    // upLinks
    S_msrRepeat            fRepeatEndingRepeatUpLink;

    // numbers
    string                 fRepeatEndingNumber; // may be "1, 2"
    int                    fRepeatEndingInternalNumber; // internally assigned

    // kind
    msrRepeatEndingKind    fRepeatEndingKind;

    // elements list
    list<S_msrVoiceElement>
                           fRepeatEndingElementsList;
```

19.37 Segments

Segments handling is presented at section 44, [Segments handling], page 272.

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 181, 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.

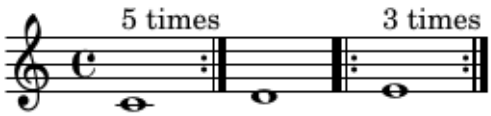


Figure 19.4: Three segments in a voice

19.38 Notes and rests

Class `msrNote` is complex class: it handles many variants, but using classes to represent the variants would be too cumbersone. As shown at figure 19.3, [The MSR classes hierarchy], page 168:

- 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 unpiched.

class `msrNote` thus uses enumeration type enumeration type `msrNoteKind`, defined in `src/formats/msr/msrBasicTyp` to distinguish them:

```
enum class msrNoteKind {
  k_NoNote,

  // in measures
  kNoteRegularInMeasure,
  kNoteRestInMeasure,
  kNoteSkipInMeasure, // an invisible rest
  kNoteUnpitchedInMeasure,

  // in chords
  kNoteRegularInChord,

  // in tuplets
  kNoteRegularInTuplet,
  kNoteRestInTuplet,
  kNoteUnpitchedInTuplet,

  // in grace notes groups
  kNoteRegularInGraceNotesGroup,
  kNoteSkipInGraceNotesGroup, // used to circumvent LilyPond issue #34

  // in chords in grace notes groups
  kNoteInChordInGraceNotesGroup,

  // in tuplets in grace notes groups
  kNoteInTupletInGraceNotesGroup,

  // in double-tremolos
  kNoteInDoubleTremolo
};
```

19.39 Articulations

19.40 Ornaments

19.41 Ties

19.42 Dynamics

19.43 Beams

19.44 Slurs

19.45 Grace notes groups

Grace notes groups handling is presented at section 59, [Grace notes groups handling], page 297.

19.46 Chords

A chord contains notes only, and can occur in measures, tuplets and grace notes groups, hence:

```
// chords
//______________________________________________________________________

enum class msrChordInKind {
  k_NoChordIn,

  kChordInMeasure,
  kChordInTuplet,
  kChordInGraceNotesGroup
};
```

19.47 Tuplets

Tuplets handling is presented at section 61, [Tuplets handling], page 299.

A tuplet can contain:

- notes and rests;
- chords;
- other tuplets.

Tuplets can occur in measures and other tuplets, hence enumeration type `msrTupletInKind`:

```
enum class msrTupletInKind {
  k_NoTupletIn,

  kTupletInMeasure,
  kTupletInTuplet
};
```

Tuplets factors are represented by class `msrTupletFactor`, defined in src/formats/msr/msrBasicTypes.h/.cpp.

```
class EXP msrTupletFactor
{
  // ... ... ...

  public:

    // public services
    // ------------------------------------------------------

    Bool                   isEqualToOne () const
                               {
                                 return
                                   fTupletActualNotes == fTupletNormalNotes;
                               }

    rational               asRational () const
                             {
                               return
                                 rational (
                                   fTupletActualNotes,
                                   fTupletNormalNotes);
                             }

  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    int                    fTupletActualNotes;
    int                    fTupletNormalNotes;
};
```

19.48 Harmonies and figured bass similarities

Harmonies and figured bass handling is presented at section 62, [Harmonies handling], page 300 and section 63, [Figured bass elements handling], page 316, respectively.

In MusicXML, harmonies and figured bass occur at the measure level:

```
      <harmony print-frame="no">
        <root>
          <root-step>C</root-step>
          </root>
        <kind text="m">minor</kind>
        </harmony>
      <note default-x="75.17" default-y="-35.00">
        <pitch>
          <step>F</step>
          <octave>4</octave>
          </pitch>
        <duration>2</duration>
        <voice>1</voice>
        <type>quarter</type>
        <stem>up</stem>
        </note>
```

```
      <harmony>
        <root>
          <root-step>F</root-step>
          <root-alter>1</root-alter>
        </root>
        <kind>major</kind>
        <inversion>2</inversion>
      </harmony>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
```

In MSR, the instances of class `msrHarmony` and class `msrFiguredBassElement` are present twice:

- each class `msrNote` instance contains the harmonies and figured bass attached to it:

```
class EXP msrNote : public msrTupletElement
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    // harmonies
    // ------------------------------------------------------

    list<S_msrHarmony>    fNoteHarmoniesList;

    // figured bass
    // ------------------------------------------------------

    list<S_msrFiguredBassElement>
                          fNoteFiguredBassElementsList;

  // ... ... ...
};
```

- each class `msrPart` instance contains a harmonies staff and voice, as well as a figured bass staff and voice:

```
class EXP msrPart : public msrPartGroupElement
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------
   // harmonies

    S_msrStaff            fPartHarmoniesStaff;
    S_msrVoice            fPartHarmoniesVoice;

    // figured bass

    S_msrStaff            fPartFiguredBassStaff;
    S_msrVoice            fPartFiguredBassVoice;
```

```
18
19    // ... ... ...
20  };
```

The way harmonies and figured bass elements are represented in MusicFormats is presented in the next two sections.

19.49 Harmonies

Harmonies handling is presented at section 62, [Harmonies handling], page 300.

19.50 Figured bass elements

Figured bass elements handling is presented at section 63, [Figured bass elements handling], page 316.

19.51 Lyrics

Lyrics handling is presented at section 64, [Lyrics handling], page 330.

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 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 msrSyllableExtendKind {
2       kSyllableExtendNone,
3       kSyllableExtendEmpty,
4       kSyllableExtendSingle,
5       kSyllableExtendStart, kSyllableExtendContinue, kSyllableExtendStop
6     };
```

Class `msrSyllable` contains:

```
    // syllable kind
    msrSyllableKind       fSyllableKind;

    // texts list
    list<string>          fSyllableTextsList;

    // extend kind
    msrSyllableExtendKind fSyllableExtendKind;

    // stanza number, may contain non-digits
    string                fSyllableStanzaNumber;

    // syllable whole notes
    rational              fSyllableWholeNotes;

    // syllable tuplet factor
    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:

```
    // contents
    vector<S_msrSyllable> fSyllables;

    Bool                  fStanzaTextPresent;
```

19.52 MIDI

MIDI handling is presented at section 66, [MIDI handling], page 332.

Chapter 20

MSR time-oriented represention

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:

```
enum class msrNoteEventKind {
  kNoteEventStart,
  kNoteEventStop
};
```

A note event is described in class :

```
class msrNoteEvent : public smartable
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    rational              fNoteEventPositionInMeasure;
    S_msrNote             fNoteEventNote;
    msrNoteEventKind      fNoteEventKind;
};
```

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:

```
class msrSimultaneousNotesChunk : public smartable
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    rational              fChunkPositionInMeasure;
    list<S_msrNote>       fChunkNotesList;
    rational              fChunkWholeNotes;
};
```

20.3 Measures slices

A measures slice, described by class `msrMeasuresSlice`, is a 'vertical' cut in the score across voices: is contains all the measures starting at the same time, one per voice:

```
class EXP msrMeasuresSlice : public smartable
{
  // ... ... ...

  protected:

    // protected fields
    // ------------------------------------------------------

    int                   fSlicePuristMeasureNumber;
    string                fSliceMeasureNumber;

    // the measures in the slice
    vector<S_msrMeasure>  fSliceMeasuresVector;

    // notes flat list
    list<S_msrNote>       fSliceNotesFlatList;

    // note events list
    list<S_msrNoteEvent>  fSliceNoteEventsList;

    // simultaneous notes chunks list
    list<S_msrSimultaneousNotesChunk>
                          fSliceSimultaneousNotesChunksList;
};
```

20.4 Measures slices sequences

A class `msrMeasuresSlicesSequence` contains a vector of `S_msrMeasuresSlice` instances:

```
class EXP msrMeasuresSlicesSequence : public smartable
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    string                fMeasuresOrigin;

    vector<S_msrMeasuresSlice>
                          fMeasuresSlicesVector;
};
```

A smart pointer to am `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 ()`:

```
void msrPart::collectPartMeasuresSlices (
  int inputLineNumber)
{
  // ... ... ...

  // create the part measures slices sequence
  fPartMeasuresSlicesSequence =
    msrMeasuresSlicesSequence::create (
      fPartName); // origin

  // populate it
  for (S_msrStaff staff : fPartAllStavesList) {
    // ... ... ...

    ++gIndenter;

    S_msrMeasuresSlicesSequence
      staffMeasuresSlicesSequence =
        staff->
          getStaffMeasuresSlicesSequence ();

    if (! staffMeasuresSlicesSequence) {
      stringstream s;

      s <<
        "The staffMeasuresSlicesSequence of staff \"" <<
        staff->getStaffName () <<
        "\" is null";

      musicxmlWarning (
        gGlobalServiceRunData->getInputSourceName (),
        inputLineNumber,
        s.str ());
    }
    else {
      fPartMeasuresSlicesSequence->
        mergeWithMeasuresSlicesSequence (
          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 (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:

```
void msrMeasuresSlicesSequence::identifySoloNotesAndRests ()
{
  // ... ... ...

  // collect the notes from the sequence's measures slices
  for (
    vector<S_msrMeasuresSlice>::const_iterator i =
      fMeasuresSlicesVector.begin ();
    i != fMeasuresSlicesVector.end ();
    ++i
  ) {
    S_msrMeasuresSlice measuresSlice = (*i);

    measuresSlice->
      collectNonSkipNotesFromMeasuresSliceMeasures ();
  } // for
}
```

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:

```
class EXP msrPathToVoice : public smartable
{
  // ... ... ...

  public:

    // public services
    // ------------------------------------------------------

    void                   appendPartGroup (S_msrPartGroup partGroup)
                               {
                                 fPartGroupsList.push_back (partGroup);
                               }

  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    S_msrBook              fBook;

    S_msrScore             fScore;

    // part groups can be nested
    list<S_msrPartGroup>   fPartGroupsList;

    S_msrPart              fPart;

    S_msrStaff             fStaff;

    S_msrVoice             fVoice;
};
```

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:

```
    // MSR data
    S_msrScore            fMsrScore;

    // ... ... ...

    // LilyPond stuff
    S_lpsrHeader          fScoreHeader;
    S_lpsrPaper           fScorePaper;
    S_lpsrLayout          fScoreLayout;

    // variables, voices and stanzas
    list<S_msrElement>    fScoreElementsList;

    // score LPSR book blocks list
    list<S_lpsrBookBlock> fScoreBookBlocksList;
    S_lpsrScoreBlock      fScoreScoreBlock; // JMI ???
```

22.1 LPSR basic types

Some types used thoughout LSPR are defined in src/formats/lpsr//lpsrBasicTypes.h/.cpp:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/lpsr >  egrep -rIn  '^// '
    lpsrBasicTypes.h
lpsrBasicTypes.h:28:// score output kinds
lpsrBasicTypes.h:50:// accidental styles
lpsrBasicTypes.h:53:// JMI there are new ones in LilyPond 2.22
lpsrBasicTypes.h:87:// chords languages
lpsrBasicTypes.h:107:// whole notes
lpsrBasicTypes.h:118:// dotted durations
lpsrBasicTypes.h:129:// rests measures
lpsrBasicTypes.h:135:// texts lists
lpsrBasicTypes.h:141:// pitches and octaves
lpsrBasicTypes.h:151:// lyrics durations
lpsrBasicTypes.h:168:// initialization
```

22.2 Adapting LilyPond code generation to the target version number

As of version 2.22, `compressFullBarRests` has been replaced by `compressEmptyMeasures` 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 thoughout BSR are defined in src/formats/bsr/bsrBasicTypes.h/.cpp:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/bsr >  egrep -rIn  '^// '
    bsrBasicTypes.h
bsrBasicTypes.h:23:// cell kinds
bsrBasicTypes.h:107:// lower case letters
bsrBasicTypes.h:139:// capitals
bsrBasicTypes.h:143://  kCellCapitalsSequenceSign, // { kCellCapitalsSign,
    kCellCapitalsSign };
bsrBasicTypes.h:145:// decimal digits
bsrBasicTypes.h:160:// lower decimal digits
bsrBasicTypes.h:174:// alterations
bsrBasicTypes.h:181:// augmentation dots
bsrBasicTypes.h:186:// arithmetic operators
bsrBasicTypes.h:195:// words
bsrBasicTypes.h:205:// braille cells
bsrBasicTypes.h:212:// braille output kinds
bsrBasicTypes.h:231:// chords languages
bsrBasicTypes.h:251:// brailling numbers
bsrBasicTypes.h:255:// brailling characters and strings
bsrBasicTypes.h:261:// writing UTF-16 to ostreams
bsrBasicTypes.h:273:// initialization
bsrBasicTypes.h:971:// constants
bsrBasicTypes.h:975:// computations
```

23.2 Representing cells

This is done basically with enumeration type enumeration type `bsrCellKind`, defined in src/formats/bsr/bsrBasicT

```
// cell kinds
//_______________________________________________________________________________
enum class bsrCellKind {
  kCellUnknown,

  // non 6dots values
  kCellEOL      , // L'\u000a'
  kCellEOP      , // L'\u000c'

  // 6dots values for Braille
```

```
  kDotsNone     , // L'\u2800'
  kDots1        , // L'\u2801'
  kDots2        , // L'\u2802'

  // ... ... ...

  kDots23456    , // L'\u283e'
  kDots123456     // L'\u283f'
};
```

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:

```
// lower case letters
//_____________________________________________________________________________
const bsrCellKind
  kCellA = bsrCellKind::kDots1,
  kCellB = bsrCellKind::kDots12,


  kCellY = bsrCellKind::kDots13456,
  kCellZ = bsrCellKind::kDots1356;
```

Capital sign:

```
// capitals
//_____________________________________________________________________________
const bsrCellKind
  kCellCapitalsSign = bsrCellKind::kDots46;
```

Decimal digits:

```
// decimal digits
//_____________________________________________________________________________
const bsrCellKind
  kCellNumberSign = bsrCellKind::kDots3456,
  kCell1 = kCellA,
  kCell2 = kCellB,
  kCell3 = kCellC,
  kCell4 = kCellD,
  kCell5 = kCellE,
  kCell6 = kCellF,
  kCell7 = kCellG,
  kCell8 = kCellH,
  kCell9 = kCellI,
  kCell0 = kCellJ;
```

Alterations:

```
// alterations
//_____________________________________________________________________________
const bsrCellKind
  kCellFlat    = bsrCellKind::kDots126,
  kCellNatural = bsrCellKind::kDots16,
  kCellSharp   = bsrCellKind::kDots146;
```

Augmentation dots:

```
// augmentation dots
//_____________________________________________________________________________
const bsrCellKind
  kCellAugmentationDot = bsrCellKind::kDots3;
```

Arithmetic operators:

```
// arithmetic operators
//_____________________________________________________________________________
const bsrCellKind
  kCell_ac_plus       = bsrCellKind::kDots235,
  kCell_ac_minus      = bsrCellKind::kDots36,
  kCell_ac_times      = bsrCellKind::kDots35,
  kCell_ac_dividedBy  = bsrCellKind::kDots25,
  kCell_ac_equals     = bsrCellKind::kDots2356;
```

Words:

```
// words
//_____________________________________________________________________________
const bsrCellKind
  kCellWordSign         = bsrCellKind::kDots345,

  kCellWordApostrophe   = bsrCellKind::kDots6,

  kCellParenthesis      = bsrCellKind::kDots2356,
  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:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/mxsr/
total 48
 0 drwxr-xr-x   6 jacquesmenu  staff   192 May 26 08:20:55 2021 ./
 0 drwxr-xr-x  10 jacquesmenu  staff   320 Jun 25 05:39:49 2021 ../
 8 -rw-r--r--@  1 jacquesmenu  staff  3292 Jun  6 06:35:19 2021 mxsr.cpp
 8 -rw-r--r--@  1 jacquesmenu  staff  1555 Jun  6 06:35:19 2021 mxsrGeneration.h
16 -rw-r--r--@  1 jacquesmenu  staff  7781 Jun  6 06:35:19 2021 mxsr0ah.cpp
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:

```
      <system-margins>
        <left-margin>15</left-margin>
        <right-margin>0</right-margin>
      </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:

```
<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:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 2.0 Partwise//EN"
                                "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:

```
class xmlelement;
class xmlattribute;

typedef SMARTP<xmlattribute>  Sxmlattribute;
typedef SMARTP<xmlelement>    Sxmlelement;
```

Class xmlattribute contains:

```
//--------------------------------------------------------------------------
class EXP xmlattribute : public smartable {
  //! the attribute name
  std::string fName;
  //! the attribute value
  std::string   fValue;

  // ... ... ...

    //! returns the attribute value as a int
    operator int () const;
    //! returns the attribute value as a long
    operator long () const;
    //! returns the attribute value as a float
    operator float () const;
```

Class xmlelement contains:

```
class EXP xmlelement : public ctree<xmlelement>, public visitable
{
  private:
    //! the element name
    std::string fName;
    //! the element value
    std::string fValue;
    //! list of the element attributes
    std::vector<Sxmlattribute> fAttributes;

  protected:
    // the element type
    int fType;
    // the input line number for messages to the user
    int fInputLineNumber;

  // ... ... ...

    //! returns the element value as a long
    operator long () const;
    //! returns the element value as a int
    operator int () const;
    //! returns the element value as a float
    operator float () const;
    //! elements comparison
```

```
    Bool operator ==(const xmlelement& elt) const;
    Bool operator !=(const xmlelement& elt) const { return !(*this == elt); }

    //! adds an attribute to the element
    long add (const Sxmlattribute& attr);

  // ... ... ...
};
```

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;

```
jacquesmenu@macmini: ~/musicformats-git-dev/libmusicxml/dtds/3.1/schema > ls -sal *.mod
 40 -rwxr-xr-x@ 1 jacquesmenu  staff  20238 Apr 22 15:49 attributes.mod
 16 -rwxr-xr-x  1 jacquesmenu  staff   4943 Apr 22 15:49 barLine.mod
 80 -rwxr-xr-x@ 1 jacquesmenu  staff  37932 Apr 22 15:49 common.mod
 88 -rwxr-xr-x@ 1 jacquesmenu  staff  41960 Apr 22 15:49 direction.mod
 16 -rwxr-xr-x@ 1 jacquesmenu  staff   4097 Apr 22 15:49 identity.mod
 24 -rwxr-xr-x@ 1 jacquesmenu  staff  10266 Apr 22 15:49 layout.mod
  8 -rwxr-xr-x@ 1 jacquesmenu  staff   2833 Apr 22 15:49 link.mod
104 -rwxr-xr-x@ 1 jacquesmenu  staff  51384 Apr 22 15:49 note.mod
 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:

```
enum {
  kNoElement,
  kComment,
  kProcessingInstruction,
  k_accent,
  k_accidental,
  k_accidental_mark,
  k_accidental_text,

  // ... ... ...

  k_work,
  k_work_number,
  k_work_title,
  kEndElement
};
```

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:

```
template <int elt> class musicxml : public xmlelement
{
  protected:
    musicxml (int inputLineNumber) : xmlelement (inputLineNumber) { fType = elt; }
};
```

The smart pointer s to the various elements are defined in libmusicxml/src/elements!typedef.h, using an anonymous enumeration type :

```
typedef SMARTP<musicxml<kComment> >          S_comment;
typedef SMARTP<musicxml<kProcessingInstruction> > S_processing_instruction;

typedef SMARTP<musicxml<k_accent> >    S_accent;
typedef SMARTP<musicxml<k_accidental> >    S_accidental;
typedef SMARTP<musicxml<k_accidental_mark> >     S_accidental_mark;
typedef SMARTP<musicxml<k_accidental_text> >     S_accidental_text;

// ... ... ...

typedef SMARTP<musicxml<k_work> >    S_work;
typedef SMARTP<musicxml<k_work_number> >     S_work_number;
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:

```
class EXP factory : public singleton<factory>{

  std::map<std::string, functor<Sxmlelement>*> fMap;
  std::map<int, const char*>  fType2Name;

  // ... ... ...
};
```

Those two maps are initialized in libmusicxml/samples/elements/factory.cpp:

```
factory::factory()
{
  fMap["comment"]      = new newElementFunctor<kComment>;
  fMap["pi"]         = new newElementFunctor<kProcessingInstruction>;
  fType2Name[kComment]   = "comment";
  fType2Name[kProcessingInstruction]   = "pi";

  fMap["accent"]   = new newElementFunctor<k_accent>;
  fMap["accidental"]   = new newElementFunctor<k_accidental>;
  fMap["accidental-mark"]    = new newElementFunctor<k_accidental_mark>;
  fMap["accidental-text"]    = new newElementFunctor<k_accidental_text>;

  // ... ... ...

  fMap["work"]   = new newElementFunctor<k_work>;
  fMap["work-number"]    = new newElementFunctor<k_work_number>;
  fMap["work-title"]   = new newElementFunctor<k_work_title>;

  fType2Name[k_accent]   = "accent";
  fType2Name[k_accidental]   = "accidental";
  fType2Name[k_accidental_mark]    = "accidental-mark";
  fType2Name[k_accidental_text]    = "accidental-text";

```

```
  // ... ... ...

  fType2Name[k_work]   = "work";
  fType2Name[k_work_number]    = "work-number";
  fType2Name[k_work_title]   = "work-title";
}
```

Class newElementFunctor is defined in to provide call operator as:

```
template<int elt>
class newElementFunctor : public functor<Sxmlelement>
{
  public:

    Sxmlelement operator ()()
        { return musicxml<elt>::new_musicxml (libmxmllineno); }
};
```

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/files/xmlfile.h:

```
//--------------------------------------------------------------------------
class EXP TXMLFile : public smartable
{
  private:
    TXMLDecl*             fXMLDecl;
    TDocType*             fDocType;
    Sxmlelement           fXMLTree;

  protected:
      TXMLFile () : fXMLDecl(0), fDocType(0) {}
    virtual ~TXMLFile () { delete fXMLDecl; delete fDocType; }

  public:
    static SMARTP<TXMLFile> create();

  public:
    TXMLDecl*     getXMLDecl ()     { return fXMLDecl; }
    TDocType*     getDocType ()     { return fDocType; }
    Sxmlelement   elements ()       { return fXMLTree; }

    void      set (Sxmlelement root)  { fXMLTree = root; }
    void      set (TXMLDecl * dec)  { fXMLDecl = dec; }
    void      set (TDocType * dt)   { fDocType = dt; }

    void      print (std::ostream& s);
};
typedef SMARTP<TXMLFile> SXMLFile;
```

24.5.1 Creating xmlelement trees from textual data

Reading MusicXML data creates instances of **xmlelement**. This is done by and instance of **xmlreader**, defined in libmusicxml/src/files/xmlreader.h/.cpp, which provides methods:

```
    SXMLFile readbuff(const char* file);
    SXMLFile read(const char* file);
    SXMLFile read(FILE* file);
```

These three functions are defined this way:

```
//--------------------------------------------------------------------------
SXMLFile xmlreader::readbuff(const char* buffer)
{
  fFile = TXMLFile::create();
  debug("read buffer", '-');
  return readbuffer (buffer, this) ? fFile : 0;
}

//--------------------------------------------------------------------------
SXMLFile xmlreader::read(const char* file)
{
  fFile = TXMLFile::create();
  debug("read", file);
  return readfile (file, this) ? fFile : 0;
}

//--------------------------------------------------------------------------
SXMLFile xmlreader::read(FILE* file)
{
  fFile = TXMLFile::create();
  return readstream (file, this) ? fFile : 0;
}
```

24.5.2 Printing xmlelement trees

An **xmlelement** can be printed by function **printMxsr ()**, defined in src/formats/mxsr/mxsr.h/.cpp:

```
void printMxsr (const Sxmlelement theMxsr, ostream& os)
{
  xmlvisitor v (os);
  tree_browser<xmlelement> browser (&v);
  browser.browse (*theMxsr);
}
```

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**:

```
//--------------------------------------------------------------------------
class EXP TXMLFile : public smartable
{
  private:
    TXMLDecl*             fXMLDecl;
    TDocType*             fDocType;
    Sxmlelement           fXMLTree;

  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/.c` 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 handled 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:

```
// %%%JMI     Bool                    fGenerateMsrVisitingInformation;
    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 the standard output stream, 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:

```
  public:

                          lpsr2lilypondTranslator (
                            S_msrOahGroup&  msrOpts,
                            S_lpsrOahGroup& lpsrOpts,
                            ostream&        lilypondCodeStream);

    virtual               ~lpsr2lilypondTranslator ();

    void                  translateLpsrToLilypondCode (
                            S_lpsrScore lpsrScore);
```

It contains these fields among others:

```
  private:

    // options
    // ------------------------------------------------------

    S_msrOahGroup         fMsrOahGroup;
    S_lpsrOahGroup        fLpsrOahGroup;

    // the LPSR score we're visiting
    // ------------------------------------------------------
    S_lpsrScore           fVisitedLpsrScore;

    // the output stream
    // ------------------------------------------------------

    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:

```
    // files includes
    Bool                    fJianpuFileIncludeIsNeeded;

    // Scheme modules
    Bool                    fScmAndAccregSchemeModulesAreNeeded;

    // Scheme functions
    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.

27.1 Basic principle

Lilypond generation is done in src/passes/bsr2braille/bsr2brailleTranslator.h/.cpp.

Class `bsr2brailleTranslator` provides:

```
  public:

                          bsr2brailleTranslator (
                            S_bsrScore      bsrScore,
                            const S_bsrOahGroup& bsrOpts,
                            ostream&        brailleOutputStream);

    virtual               ~bsr2brailleTranslator ();

    void                  translateBsrToBraille ();
```

It contains these fields among others:

```
  private:

    // options
    // ------------------------------------------------------

    S_bsrOahGroup         fBsrOahGroup;

    // the BSR score we're visiting
    // ------------------------------------------------------

    S_bsrScore            fVisitedBsrScore;

    // the braille generator used
    // ------------------------------------------------------

    S_bsrBrailleGenerator fBrailleGenerator;

    // the output stream
    // ------------------------------------------------------

    ostream&              fBrailleOutputStream;
```

27.2 Output files name and contents options

he contents options use the following enumeration types:

```
enum class bsrUTFKind {
  kUTF8, kUTF16
};

enum class bsrByteOrderingKind {
  kByteOrderingNone,
  kByteOrderingBigEndian, kByteOrderingSmallEndian
};
```

`xml2brl` supplies a option `-files` options subgroup:

```
jacquesmenu@macmini > xml2brl -query files
--- Help for subgroup "files" in group "Files group" ---
  Files group (-files-group):
  --------------------------
    Files (-files):
      -o, -output-file-name FILENAME
            Write Braille to file FILENAME instead of standard output.
      -aofn, -auto-output-file-name
            This option can only be used when reading from a file.
            Write MusicXML code to a file in the current working directory.
            The file name is derived from that of the input file,
            replacing any suffix after the '.' by 'xml'
            or adding '.xml' if none is present.
      -bok, -braille-output-kind OUTPUT_KIND
            Use OUTPUT_KIND to write the generated Braille to the output.
            The 4 output kinds available are:
            ascii, utf16, utf8 and utf8d.
            'utf8d' leads to every line in the braille score to be generated
            as a line of cells followed by a line of text showing the contents
            for debug purposes.
            The default is 'ascii'.
      -ueifn, -use-encoding-in-file-name
            Append a description of the encoding used
            and the presence of a BOM if any to the file name before the '.'.
      -bom, -byte-ordering-mark BOM_ENDIAN
            Generate an initial BOM_ENDIAN byte ordering mark (BOM)
            ahead of the Braille nusic code,
            which can be one of 'big' or 'small'.
            By default, a big endian BOM is generated.
```

27.3 Braille generators

The following classes are defined in src/formatsgeneration/brailleGeneration/brailleGeneration.h/.cpp

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formatsgeneration/brailleGeneration > grep
     class brailleGeneration.h
enum class bsrUTFKind {
enum class bsrByteOrderingKind {
class EXP bsrBrailleGenerator : public smartable
/* this class is purely virtual
class EXP bsrAsciiBrailleGenerator : public bsrBrailleGenerator
class EXP bsrUTF8BrailleGenerator : public bsrBrailleGenerator
class EXP bsrUTF8DebugBrailleGenerator : public bsrUTF8BrailleGenerator
class EXP bsrUTF16BigEndianBrailleGenerator : public bsrBrailleGenerator
class EXP bsrUTF16SmallEndianBrailleGenerator : public bsrBrailleGenerator
```

The base class `bsrBrailleGenerator` contains:

```
  public:

    // public services
    // ------------------------------------------------------

    virtual void          generateCodeForBrailleCell (
                            bsrCellKind cellKind) = 0;

    void                  generateCodeForCellsList (
                            S_bsrCellsList cellsList);

    virtual void          generateCodeForMusicHeading (
                            S_bsrMusicHeading musicHeading);

    virtual void          generateCodeForLineContents (
                            S_bsrLineContents lineContents);

    // ... ... ...

  protected:

    // protected fields
    // ------------------------------------------------------

    ostream&              fBrailleOutputStream;
```

27.4 Writing braille cells

Braille cells are output to an `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:

```
void bsrAsciiBrailleGenerator::generateCodeForBrailleCell (
  bsrCellKind cellKind)
{
  string stringForCell;

  switch (cellKind) {
    case bsrCellKind::kCellUnknown:
      {
        stringstream s;

        s <<
          "cannot generate code for braille cell '" <<
          bsrCellKindAsString (cellKind) <<
          "'";
        msrInternalError (
          gGlobalServiceRunData->getInputSourceName (),
          -999, // inputLineNumber, TICINO JMI
          __FILE__, __LINE__,
          s.str ());
      }
      break;

    case bsrCellKind::kCellEOL:    stringForCell = "\x0a"; break;
    case bsrCellKind::kCellEOP:    stringForCell = "\x0c"; break;

    case bsrCellKind::kDotsNone:   stringForCell = "\x20"; break;

    case bsrCellKind::kDots1:      stringForCell = "\x41"; break;
    case bsrCellKind::kDots2:      stringForCell = "\x31"; break;

```

```
    // ... ... ...

    case bsrCellKind::kDots23456:  stringForCell = "\x29"; break;
    case bsrCellKind::kDots123456: stringForCell = "\x3d"; break;
  } // switch

  fBrailleOutputStream <<
    stringForCell;
}
```

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:

```
  // create a direction element
  Sxmlelement directionElement = createMxmlelement (k_direction, "");

  // set it's "placement" attribute if relevant
  string
    placementString =
      msrPlacementKindAsMusicXMLString (placementKind);

  if (placementString.size ()) {
    directionElement->add (createMxmlAttribute ("placement",  placementString));
  }
```

This one supplies a value to the xmlelement it creates:

```
void msr2mxsrTranslator::visitStart (S_msrIdentification& elt)
{
  // composers
  const list<string>&
    composersList =
      elt->getComposersList ();

  for (
    list<string>::const_iterator i=composersList.begin ();
    i!=composersList.end ();
    ++i
  ) {
    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 ()`:

```
<clef number="2">
     <sign>G</sign>
     <line>2</line>
     <clef-octave-change>-1</clef-octave-change>
  </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:

```
  public:

                          msr2mxsrTranslator (
                            S_msrScore visitedMsrScore);

    virtual               ~msr2mxsrTranslator ();

    Sxmlelement           translateMsrToMxsr ();

  // ... ... ...

  private:

    // the MSR score we're visiting
    // ------------------------------------------------------
    S_msrScore                fVisitedMsrScore;


    // the MXSR we're building
    // ------------------------------------------------------
    Sxmlelement               fResultingMusicxmlelement;
```

The method `msr2mxsrTranslator::translateMsrToMxsr ()` method does the following:

```
//________________________________________________________________________
Sxmlelement msr2mxsrTranslator::translateMsrToMxsr ()
{
  // sanity check
  mfAssert (
    __FILE__, __LINE__,
    fVisitedMsrScore != nullptr,
    "fVisitedMsrScore is null");

  // create the current score part-wise element
  fResultingMusicxmlelement =
    createMxmlScorePartWiseElement ();

  // create a msrScore browser
  msrBrowser<msrScore> browser (this);

  // browse the visited score with the browser
  browser.browse (*fVisitedMsrScore);

  return fResultingMusicxmlelement;
}
```

28.5 Ancillary functions to create MXSR data

The function `createMxmlScorePartWiseElement ()` is defined in src/formats/mxsr/mxsr.h/.cpp:

```
//------------------------------------------------------------------------
Sxmlelement createMxmlScorePartWiseElement ()
{
  Sxmlelement result = factory::instance ().create (k_score_partwise);

  Sxmlattribute versionAttribute = createMxmlAttribute("version", "3.1");
  result->add (versionAttribute);

  return result;
}
```

Chapter 29

Guido code generation

Guido code is produced on the standard output stream, 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 214, 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`). Is was written in the first place to check the MSR API before writing the MSDL converter.

The score produced is shown at figure 30.1, [Zoltán Kodály's Mikrokosmos III Wandering], page 220.

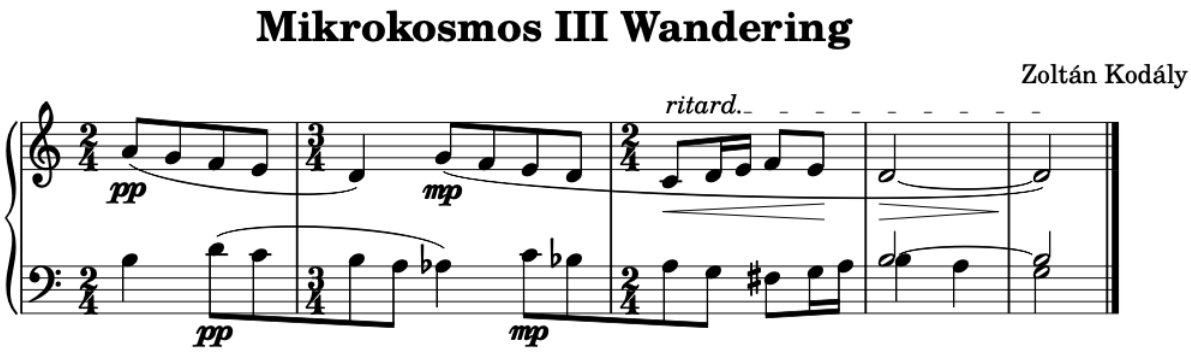


Figure 30.1: Zoltán Kodály's Mikrokosmos III Wandering

30.3 LilyPondIssue34

This service produces the same score as that obtained by:

```
xml2ly -auto-output-file-name gracenotes/LilyPondIssue34.xml
```

The resulting score is shown at figure 30.2, [The LilyPondIssue34 score], page 221.

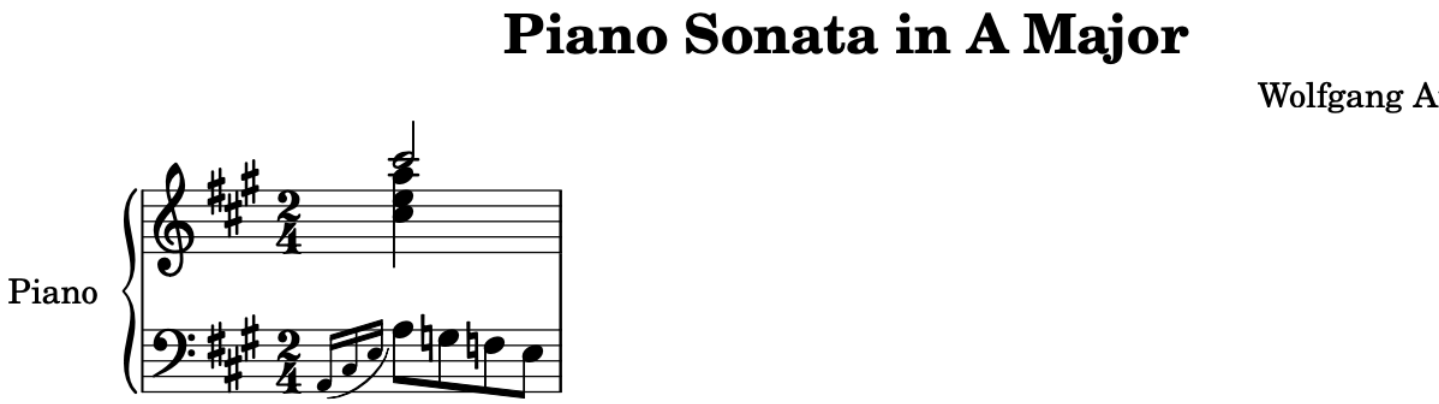


Figure 30.2: The LilyPondIssue34 score

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 in 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:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxml > xml2ly -about
What xml2ly does:

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

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

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

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 xml2gmn

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/`:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > ll interfaces/msrinterfaces/
total 32
 0 drwxr-xr-x  5 jacquesmenu  staff   160 May 26 08:20:55 2021 ./
 0 drwxr-xr-x  8 jacquesmenu  staff   256 Jun 25 05:59:13 2021 ../
 8 -rw-r--r--@ 1 jacquesmenu  staff    77 Apr 22 15:49:27 2021 README.md
16 -rw-r--r--@ 1 jacquesmenu  staff  5796 Jun 24 17:47:02 2021 msrInterface.cpp
 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/`:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > ll interfaces/lpsrinterfaces/
total 24
0 drwxr-xr-x  5 jacquesmenu  staff   160 Jun 13 07:36:53 2021 ./
0 drwxr-xr-x  8 jacquesmenu  staff   256 Jun 25 05:59:13 2021 ../
8 -rw-r--r--@ 1 jacquesmenu  staff    78 Jun 13 07:37:13 2021 README.md
8 -rw-r--r--@ 1 jacquesmenu  staff   670 Jun 13 07:41:01 2021 lpsrInterface.cpp
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/`:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > ll interfaces/msdlinterfaces/
total 8
0 drwxr-xr-x  3 jacquesmenu  staff    96 Jun 25 05:57:39 2021 ./
0 drwxr-xr-x  8 jacquesmenu  staff   256 Jun 25 05:59:13 2021 ../
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 , the `main ()` function does:

```
int main (int argc, char*  argv[])
//------------------------------------------------------------------------
{
  // setup signals catching
  // ------------------------------------------------------

  catchSignals ();

  // ... ... ...

  switch (multiGenerationOutputKind) {
    case mfMultiGenerationOutputKind::k_NoGeneration:
      // should not occur, unless the run is a pure help one
      break;

    // ... ... ...

    case mfMultiGenerationOutputKind::kGenerationGuido:
      err =
        msrScore2guidoWithHandler (
          theMsrScore,
          "Pass 2",
          "Convert the MSR score into a second MSR",
          "Pass 3",
          "Convert the second MSR into an MXSR",
          "Pass 4",
          "Convert the MXSR into Guido text",
          cout,
          cerr,
          handler);
      break;

  // ... ... ...
}
```

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.

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/converters/musicxml2musicxml > cat
    musicxml2musicxmlInterface.h
/*
  MusicFormats Library
  Copyright (C) Jacques Menu 2016-2022

  This Source Code Form is subject to the terms of the Mozilla Public
  License, v. 2.0. If a copy of the MPL was not distributed with this
  file, You can obtain one at http://mozilla.org/MPL/2.0/.

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

#ifndef ___musicxml2musicxmlInterface___
#define ___musicxml2musicxmlInterface___

#include "mfMusicformatsError.h" // for mfMusicformatsError


using namespace std;

namespace MusicFormats
{
/*
  The API functions with an options and arguments and no handler
  are declared in libmusicxml.h
*/

//_____________________________________________________________________________
EXP mfMusicformatsError convertMusicxmlFile2musicxmlWithHandler (
  const char*  fileName,
  ostream&     out,
  ostream&     err,
  S_oahHandler handler);

//_____________________________________________________________________________
EXP mfMusicformatsError convertMusicxmlFd2musicxmlWithHandler (
  FILE*        fd,
  ostream&     out,
  ostream&     err,
  S_oahHandler handler);

//_____________________________________________________________________________
EXP mfMusicformatsError convertMusicxmlString2musicxmlWithHandler (
  const char*  buffer,
```

```
45    ostream&      out,
46    ostream&      err,
47    S_oahHandler  handler);
48
49
50  }
51
52
53  #endif
```

Part XI

Selected topics

Chapter 36

Initializations

Some initialization activities in MusicFormats use the OAH facility. OAH should thus be initialized first.

36.1 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:

```
EXP extern S_generalOahGroup gGlobalServiceRunData;

//_____________________________________________________________________________
EXP S_generalOahGroup createGlobalGeneralOahGroup ();
```

```
S_generalOahGroup createGlobalGeneralOahGroup ()
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
    gLogStream <<
      "Creating global general OAH group" <<
      endl;
  }
#endif

  // protect library against multiple initializations
  if (! gGlobalServiceRunData) {
    // create the global general options group
    gGlobalServiceRunData =
      generalOahGroup::create ();
    assert (gGlobalServiceRunData != 0);
  }

  // return the global OAH group
  return gGlobalServiceRunData;
}
```

36.2 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:

```
void EXP initializeMSR ();
```

```
static S_mfcVersions pMsrRepresentationComponent;

static void initializeMsrRepresentationComponent ()
{
  // create the component
  pMsrRepresentationComponent =
    mfcVersions::create ("MSR");

  // populate it
  pMsrRepresentationComponent->
    appendVersionDescrToComponent (
      mfcVersionDescr::create (
        mfcVersionDescr::create (
          mfcVersionNumber::createFromString ("0.9.50"),
          "October 6, 2021",
          list<string> {
            "Start of sequential versions numbering"
          }
      )));
}

void initializeMSR ()
{
  // protect library against multiple initializations
  static Bool pPrivateThisMethodHasBeenRun (false);

  if (! pPrivateThisMethodHasBeenRun) {
    // initialize the history
    initializeMsrRepresentationComponent ();

    // initialize
    initializeMsrBasicTypes ();

    pPrivateThisMethodHasBeenRun = true;
  }
}
```

36.2.1 MSR initialization

src/formats/msr/msrBasicTypes.h/.cpp defines function `initializeMSRBasicTypes ()` for this initialization:

```
void initializeMsrBasicTypes ()
{
  // protect library against multiple initializations
  static Bool pPrivateThisMethodHasBeenRun (false);

  if (! pPrivateThisMethodHasBeenRun) {
#ifdef TRACING_IS_ENABLED
    if (gGlobalOahEarlyOptions.getEarlyTracingOah () && ! gGlobalOahEarlyOptions.
    getEarlyQuietOption ()) {
      gLogStream <<
```

```
        "Initializing MSR basic types handling" <<
        endl;
  }
#endif

    // languages handling
    // ------------------------------------------------------

    initializeQuarterTonesPitchesLanguageKinds ();

    // clefs handling
    // ------------------------------------------------------

    initializeClefKinds ();

    // harmonies handling
    // ------------------------------------------------------

    initializeHarmonyKinds ();

    // harmony structures handling
    // ------------------------------------------------------

    initializeHarmonyStructuresMap ();

    // MSR lengths handling
    // ------------------------------------------------------

    initializeMsrLengthUnitKindsMap ();

    // MSR margins types handling
    // ------------------------------------------------------

    initializeMsrMarginTypeKindsMap ();

    pPrivateThisMethodHasBeenRun = true;
  }
}
```

36.2.2 LPSR initialization

36.2.3 BSR initialization

36.3 Passes initializations

36.4 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:

```
void xml2lyInsiderHandler::createTheXml2lyOptionGroups (
  string serviceName)
{
  // ... ... ...

  // initialize options handling, phase 1
```

```
  // ------------------------------------------------------

  // create the OAH OAH group first
  appendGroupToHandler (
    createGlobalOahOahGroup (
      serviceName));

  // create the WAE OAH group
  appendGroupToHandler (
    createGlobalWaeOahGroup ());

#ifdef TRACING_IS_ENABLED
  // create the tracing OAH group
  appendGroupToHandler (
    createGlobalTracingOahGroup (
      this));
#endif

  // create the output file OAH group
  appendGroupToHandler (
    createGlobalOutputFileOahGroup ());

  // initialize the library
  // ------------------------------------------------------

  initializeMSR ();
  initializeLPSR ();

  // initialize options handling, phase 2
  // ------------------------------------------------------

  // create the MXSR OAH group
  appendGroupToHandler (
    createGlobalMxsrOahGroup ());

  // create the mxsr2msr OAH group
  appendGroupToHandler (
    createGlobalMxsr2msrOahGroup (
      this));

  // create the MSR OAH group
  appendGroupToHandler (
    createGlobalMsrOahGroup ());

  // create the msr2msr OAH group
  appendGroupToHandler (
    createGlobalMsr2msrOahGroup ());

  // create the msr2lpsr OAH group
  appendGroupToHandler (
    createGlobalMsr2lpsrOahGroup ());

  // create the LPSR OAH group
  appendGroupToHandler (
    createGlobalLpsrOahGroup ());

  // create the LilyPond generation OAH group
  appendGroupToHandler (
    createGlobalLpsr2lilypondOahGroup ());

#ifdef EXTRA_OAH_IS_ENABLED
  // create the extra OAH group
  appendGroupToHandler (
    createGlobalHarmoniesExtraOahGroup ());
#endif

  // create the global xml2ly OAH group only now,
```

```
74    // after the groups whose options it may use
75    // have been created
76    appendGroupToHandler (
77      createGlobalXml2lyInsiderOahGroup ());
78
79    // ... ... ...
80  }
```

Chapter 37

The OAH atoms collection

These handly general-purpose OAH atoms are used in MusicFormats itself. They are defined in src/oah/oahAtomsCol

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class oahAtomsCollection.h
class EXP oahAtomAlias : public oahAtom
class EXP oahMacroAtom : public oahAtom
class EXP oahOptionsUsageAtom : public oahPureHelpAtomWithoutAValue
class EXP oahHelpAtom : public oahPureHelpAtomWithoutAValue
class EXP oahHelpSummaryAtom : public oahPureHelpAtomWithoutAValue
class EXP oahAboutAtom : public oahPureHelpAtomWithoutAValue
class EXP oahVersionAtom : public oahPureHelpAtomWithoutAValue
class EXP oahContactAtom : public oahPureHelpAtomWithoutAValue
class EXP oahBooleanAtom : public oahAtom
class EXP oahTwoBooleansAtom : public oahBooleanAtom
class EXP oahThreeBooleansAtom : public oahBooleanAtom
class EXP oahCombinedBooleansAtom : public oahAtom
class EXP oahCommonPrefixBooleansAtom : public oahAtom
class EXP oahIntegerAtom : public oahAtomStoringAValue
class EXP oahTwoIntegersAtom : public oahIntegerAtom
class EXP oahFloatAtom : public oahAtomStoringAValue
class EXP oahStringAtom : public oahAtomStoringAValue
class EXP oahFactorizedStringAtom : public oahAtom
class EXP oahStringWithDefaultValueAtom : public oahStringAtom
class EXP oahRationalAtom : public oahAtomStoringAValue
class EXP oahNaturalNumbersSetElementAtom : public oahAtomStoringAValue
class EXP oahRGBColorAtom : public oahAtomStoringAValue
class EXP oahIntSetElementAtom : public oahAtomStoringAValue
class EXP oahStringSetElementAtom : public oahAtomStoringAValue
class EXP oahStringToIntMapElementAtom : public oahAtomStoringAValue
class EXP oahStringAndIntegerAtom : public oahAtomStoringAValue
class EXP oahStringAndTwoIntegersAtom : public oahAtomStoringAValue
class EXP oahLengthUnitKindAtom : public oahAtomStoringAValue
class EXP oahLengthAtom : public oahAtomStoringAValue
class EXP oahMidiTempoAtom : public oahAtomStoringAValue
class EXP oahOptionNameHelpAtom : public oahStringWithDefaultValueAtom
class EXP oahQueryOptionNameAtom : public oahPureHelpAtomExpectingAValue
class EXP oahFindStringAtom : public oahPureHelpAtomExpectingAValue
```

37.1 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:

```
class EXP oahMacroAtom : public oahAtom
{
/*
  a list of atoms
*/

  // ... ... ...

  public:

    // public services
    // ------------------------------------------------------

    void                    appendAtomToMacro (S_oahAtom atom);

    void                    applyElement (ostream& os) override;


  private:

    // private fields
    // ------------------------------------------------------

     list<S_oahAtom>        fMacroAtomsList;
};
```

Populating field `oahMacroAtom::fMacroAtomsList` is straightfoward:

```
void oahMacroAtom::appendAtomToMacro (S_oahAtom atom)
{
  // sanity check
  mfAssert (
    __FILE__, __LINE__,
    atom != nullptr,
    "atom is null");

  fMacroAtomsList.push_back (atom);
}
```

Applying the macro atom is done in method `oahMacroAtom::applyElement ()`:

```
void oahMacroAtom::applyElement (ostream& os)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
    gLogStream <<
      "==> option '" << fetchNames () << "' is a oahMacroAtom" <<
      endl;
  }
#endif

  for (
    list<S_oahAtom>::const_iterator i =
      fMacroAtomsList.begin ();
    i != fMacroAtomsList.end ();
    ++i
  ) {
    S_oahAtom atom = (*i);

    if (
      // oahAtomStoringAValue?
      S_oahAtomStoringAValue
        atomWithVariable =
          dynamic_cast<oahAtomStoringAValue*>(&(*atom))
```

```
    ) {
//        atomWithVariable-> JMI ???
//          applyAtomWithValue (theString, os);
    }
    else {
      // valueless atom
      atom->
        applyElement (os);
    }
  } // for
}
```

37.2 A OAH macro atom example

`xml2brl` has the `-auto-utf8, -au8d` option:

```
jacquesmenu@macmini > xml2brl -query auto-utf8d
--- Help for atom "auto-utf8d" in subgroup "Files"
    -auto-utf8d, -au8d
          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:

```
void brailleGenerationOahGroup::initializeMacroOptions ()
{
  S_oahSubGroup
    subGroup =
      oahSubGroup::create (
        "Macros",
        "help-braille-generation-macros", "hbgm",
R"()",
      oahElementVisibilityKind::kElementVisibilityWhole,
      this);

  appendSubGroupToGroup (subGroup);

  // create the auto utfd8 macro

  S_oahMacroAtom
    autoUTFd8MacroAtom =
      oahMacroAtom::create (
        "auto-utf8d", "au8d",
        "Combines -auto-output-file-name, -utf8d and -use-encoding-in-file-name");

  subGroup->
    appendAtomToSubGroup (
      autoUTFd8MacroAtom);

  // populate it
  autoUTFd8MacroAtom->
    appendAtomToMacro (
      gGlobalOutputFileOahGroup->getAutoOutputFileNameAtom ());

  fBrailleOutputKindAtom->
    applyAtomWithValue (
      "utf8d",
      gLogStream);
  autoUTFd8MacroAtom->
    appendAtomToMacro (
      fBrailleOutputKindAtom);

  autoUTFd8MacroAtom->
```

```
40      appendAtomToMacro (
41        fUseEncodingInFileNameAtom);
42  }
```

37.3 LilyPond octave entry

Pass `lpsr2lilypond` has three options to choose this, all controlling one and the same variable:

```
jacquesmenu@macmini > xml2ly -query absolute
--- Help for atom "absolute" in subgroup "Notes"
    -abs, -absolute
          Use absolute octave entry in the generated LilyPond code.
```

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

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

This is done in src/formatsgeneration/lilypondGeneration/lpsr2lilypondOah.h/.cpp using a single instance of class `msrOctaveEntryVariable`:

```
class EXP msrOctaveEntryVariable : public smartable
{
  // ... ... ...

  private:

    // private fields
    // ------------------------------------------------------

    string                fVariableName;
    msrOctaveEntryKind    fOctaveEntryKind;

    Bool                  fSetByUser;
};
```

The three classes:

- `lilypondAbsoluteOctaveEntryAtom`
- `lilypondRelativeOctaveEntryAtom`
- `lilypondFixedOctaveEntryAtom`

all contain an alias for an class `msrOctaveEntryVariable` variable:

```
    // private fields
    // ------------------------------------------------------

    msrOctaveEntryVariable&
                          fOctaveEntryKindVariable;
```

The `fOctaveEntryVariable` filed of class `lpsr2lilypondOahGroup` shared be all three options atoms is:

```
    // notes
    // ---------------------------------------

    msrOctaveEntryVariable
                          fOctaveEntryVariable;
```

Chapter 38

Measures handling

Measures are presented at section 19.30, [Measures], page 174.

38.1 Voices contents

Class `msrVoice` contain a list of the first elements and a last segment:

```
    list<S_msrVoiceElement>
                          fVoiceInitialElementsList;

    // fVoiceLastSegment contains the music
    // not yet stored in fVoiceInitialElementsList,
    // it is thus logically the end of the latter,
    // and is created implicitly for every voice.
    // It is needed 'outside' of the 'list<S_msrElement>'
    // because it is not a mere S_msrElement, but a S_msrSegment
    S_msrSegment          fVoiceLastSegment;
```

38.2 Voice elements

The class `msrVoiceElement` subclasses instances in `fVoiceInitialElementsList` can be of types:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep 'public msrVoiceElement' formats/
    msr/*.h
formats/msr/msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
formats/msr/msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
formats/msr/msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
formats/msr/msrFullMeasureRests.h:class EXP msrFullMeasureRests : public msrVoiceElement
formats/msr/msrSegments.h:class EXP msrSegment : public msrVoiceElement
```

Class `msrSegment` contains a list of measures:

```
    // the measures in the segment contain the mmusic
    list<S_msrMeasure>    fSegmentMeasuresList;
```

Class contains a list of mesure elements:

```
    // elements

    list<S_msrMeasureElement>
                          fMeasureElementsList;
```

38.3 Measure elements

The class `msrMeasureElements` subclasses instances in can be of types:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'public
    msrMeasureElement' *.h
msrBarChecks.h:class EXP msrBarCheck : public msrMeasureElement
msrBarLines.h:class EXP msrBarLine : public msrMeasureElement
msrBarNumberChecks.h:class EXP msrBarNumberCheck : public msrMeasureElement
msrBreaks.h:class EXP msrLineBreak : public msrMeasureElement
msrBreaks.h:class EXP msrPageBreak : public msrMeasureElement
msrClefs.h:class EXP msrClef : public msrMeasureElement
msrCodas.h:class EXP msrCoda : public msrMeasureElement
msrDoubleTremolos.h:class EXP msrDoubleTremolo : public msrMeasureElement
msrEyeGlasses.h:class EXP msrEyeGlasses : public msrMeasureElement
msrFiguredBassElements.h:class EXP msrFiguredBassElement : public msrMeasureElement
msrHarmonies.h:class EXP msrHarmony : public msrMeasureElement
msrHiddenMeasureAndBarLines.h:class EXP msrHiddenMeasureAndBarLine : public
    msrMeasureElement
msrInstruments.h:class EXP msrScordatura : public msrMeasureElement
msrInstruments.h:class EXP msrAccordionRegistration : public msrMeasureElement
msrInstruments.h:class EXP msrHarpPedalsTuning : public msrMeasureElement
msrInstruments.h:class EXP msrPedal : public msrMeasureElement
msrInstruments.h:class EXP msrDamp : public msrMeasureElement
msrInstruments.h:class EXP msrDampAll : public msrMeasureElement
msrKeys.h:class EXP msrKey : public msrMeasureElement
msrLyrics.h:class EXP msrSyllable : public msrMeasureElement
msrMusicXMLSpecifics.h:class EXP msrPrintLayout : public msrMeasureElement
msrRehearsalMarks.h:class EXP msrRehearsalMark : public msrMeasureElement
msrSegnos.h:class EXP msrSegno : public msrMeasureElement
msrSegnos.h:class EXP msrDalSegno : public msrMeasureElement
msrStavesDetails.h:class EXP msrStaffDetails : public msrMeasureElement
msrTempos.h:class EXP msrTempo : public msrMeasureElement
msrTimeSignatures.h:class EXP msrTimeSignature : public msrMeasureElement
msrTranspositions.h:class EXP msrOctaveShift : public msrMeasureElement
msrTranspositions.h:class EXP msrTranspose : public msrMeasureElement
msrTupletElements.h:class EXP msrTupletElement : public msrMeasureElement
msrVoiceStaffChanges.h:class EXP msrVoiceStaffChange : public msrMeasureElement
```

38.4 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:

```
void msrMeasure::appendElementToMeasure (S_msrMeasureElement elem)
{
  int inputLineNumber =
    elem->getInputLineNumber ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceMeasures ()) {
    gLogStream <<
      "Appending element " <<
      elem->asShortString () <<
      " to measure " <<
      asShortString () <<
      " in voice \"" <<
      fetchMeasureVoiceUpLink ()->
        getVoiceName () <<
      "\", currentMeasureWholeNotesDuration = " <<
      fCurrentMeasureWholeNotesDuration <<
      ", line " << inputLineNumber <<
      endl;
```

```
  }
#endif

  // set elem's measure number
  elem->
    setMeasureElementMeasureNumber (
      fMeasureElementMeasureNumber);

  // set elem's position in measure
  elem->
    setMeasureElementPositionInMeasure (
      fCurrentMeasureWholeNotesDuration,
      "appendElementToMeasure()");

  fMeasureElementsList.push_back (elem);

  // take elem's sounding whole notes into account JMI ???
if (false) // JMI CAFE
  incrementCurrentMeasureWholeNotesDuration (
    inputLineNumber,
    elem->
      getMeasureElementSoundingWholeNotes ());
}
```

Here is how a harmony instance is appended to a measure:

```
void msrMeasure::appendHarmonyToMeasure (S_msrHarmony harmony)
{
  int inputLineNumber =
    harmony->getInputLineNumber ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
    gLogStream <<
      "Appending harmony " << harmony->asString () <<
      " to measure " <<
      this->asShortString () <<
      " in segment '" <<
      fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
      "' in voice \"" <<
      fMeasureSegmentUpLink->
        getSegmentVoiceUpLink ()->
          getVoiceName () <<
      "\", currentMeasureWholeNotesDuration = " <<
      fCurrentMeasureWholeNotesDuration <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  // set harmony's measure number
  harmony->
    setMeasureElementMeasureNumber (
      fMeasureElementMeasureNumber);

  // append the harmony to the measure elements list
  // DON'T call 'appendElementToMeasure (harmony)':
  // that would override harmony's position in measure,
  // which already has the correct value, thus:
  fMeasureElementsList.push_back (harmony);

  // get harmony sounding whole notes
  rational
    harmonySoundingWholeNotes =
      harmony->
        getHarmonySoundingWholeNotes ();
```

```
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 (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       fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
14       "' in voice clone \"" <<
15       fMeasureSegmentUpLink->
16         getSegmentVoiceUpLink ()->
17           getVoiceName () <<
18       "\", currentMeasureWholeNotesDuration = " <<
19       fCurrentMeasureWholeNotesDuration <<
20       ", line " << inputLineNumber <<
21       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         getHarmonySoundingWholeNotes ();
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 }
```

38.5 Appending measures to a segment

Measures are appended to a segment by method `msrSegment::appendMeasureToSegment ()` in src/formats/msr/msrSegments.h/.cpp:

```
void msrSegment::appendMeasureToSegment (S_msrMeasure measure)
{
  int inputLineNumber =
    measure->getInputLineNumber ();

  string measureNumber =
    measure->getMeasureElementMeasureNumber ();

  unsigned int segmentMeasuresListSize =
    fSegmentMeasuresList.size ();

  string currentMeasureNumber =
    segmentMeasuresListSize == 0
      ? ""
      : fSegmentMeasuresList.back ()->getMeasureElementMeasureNumber ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceMeasures ()) {
    gLogStream <<
      "Appending measure '" << measureNumber <<
      "' to segment " << asString ();

    if (fSegmentMeasuresList.size () == 0)
      gLogStream <<
        ", as first measure";
    else
      gLogStream <<
      ", after measure number '" << currentMeasureNumber << "'";

    gLogStream <<
      "' in voice \"" <<
      fSegmentVoiceUpLink->getVoiceName () <<
      "\"" <<
      ", line " << measure->getInputLineNumber () <<
      endl;
  }
#endif

  if (measureNumber == currentMeasureNumber) {
    stringstream s;

    s <<
      "appending measure number '" << measureNumber <<
      "' occurs twice in a row in voice \"" <<
      fSegmentVoiceUpLink->getVoiceName () <<
      "\"";

//    msrInternalWarning ( // JMI
    msrInternalError (
      gGlobalServiceRunData->getInputSourceName (),
      inputLineNumber,
      __FILE__, __LINE__,
      s.str ());
  }

  // is measure the first one this segment?
  if (segmentMeasuresListSize == 0) {
    measure->
      setMeasureFirstInSegmentKind (
        msrMeasure::kMeasureFirstInSegmentKindYes);
  }
  else {
    measure->
      setMeasureFirstInSegmentKind (
        msrMeasure::kMeasureFirstInSegmentKindNo);
  }
```

```
67
68    // is measure the first one it the voice?
69    // this is necessary for voice clones,
70    // which don't go down the part-staff-voice-segment hierarchy
71    if (! fSegmentVoiceUpLink->getVoiceFirstMeasure ()) {
72      // yes, register it as such
73      fSegmentVoiceUpLink->
74        setVoiceFirstMeasure (measure);
75
76      measure->
77        setMeasureFirstInVoice ();
78    }
79
80    // append measure to the segment
81    fSegmentMeasuresList.push_back (measure);
82  }
```

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 `msrFullMeasureRests::appendMeasureCloneToFullMeasureRests ()` 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::createFullMeasureRestsInVoice ()`

38.6 Appending measures to a voice

Method `msrVoice::appendMeasureCloneToVoiceClone ()` does the job in src/formats/msr/msrVoices.h/.cpp.

```
S_msrMeasure msrVoice::createAMeasureAndAppendItToVoice (
  int     inputLineNumber,
  string measureNumber,
  msrMeasureImplicitKind
          measureImplicitKind)
{
  fVoiceCurrentMeasureNumber = measureNumber;

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceMeasures ()) {
    gLogStream <<
      "Creating measure '" <<
```

```
      measureNumber <<
      "' and appending it to voice \"" << getVoiceName () << "\"" <<
      "', line " << inputLineNumber <<
      endl;
  }
#endif

  fCallsCounter++;

  if (
//     true
//       ||
    false
      &&
    (
      fCallsCounter == 2 && getVoiceName ()
        ==
      "Part_POne_HARMONIES_Staff_Voice_Eleven_HARMONIES"
    )
  ) { // POUSSE JMI
    gLogStream <<
      endl <<
      "++++ createAMeasureAndAppendItToVoice() POUSSE, fCallsCounter: " << fCallsCounter
    << " ++++" <<
      endl;
    this->print (gLogStream);
    gLogStream <<
      endl;
  }

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
    displayVoice (
      inputLineNumber,
      "createAMeasureAndAppendItToVoice() 1");
  }
#endif

  ++gIndenter;

  // create the voice last segment if needed
  if (! fVoiceLastSegment) {
    createNewLastSegmentForVoice (
      inputLineNumber,
      "createAMeasureAndAppendItToVoice() 2");
  }

  // append a new measure with given number to voice last segment
  S_msrMeasure
    result =
      fVoiceLastSegment->
        createAMeasureAndAppendItToSegment (
          inputLineNumber,
          measureNumber,
          measureImplicitKind);

  // result is the new voice last appended measure
  fVoiceLastAppendedMeasure = result;

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
    displayVoice (
      inputLineNumber,
      "createAMeasureAndAppendItToVoice() 3");
  }
#endif

```

```
79    --gIndenter;
80
81    return result;
82  }
```

38.7 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   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       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     }
```

```
  }

  // append a new measure to the current part
  fCurrentPart->
    createAMeasureAndAppendItToPart (
      inputLineNumber,
      fCurrentMeasureNumber,
      measureImplicitKind);

  // ... ... ...
}
```

Upon the second visit of `S_measure`, the last appended measure appended to the current part is finalized:

```
void mxsr2msrTranslator::visitEnd (S_measure& elt)
{
  // ... ... ...

  // finalize current measure in the part,
  // to add skips if necessary and set measure kind
  fCurrentPart->
    finalizeLastAppendedMeasureInPart (
      inputLineNumber);

  // ... ... ...
}
```

38.8 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`:

```
//________________________________________________________________________
void mxsr2msrTranslator::visitStart (S_measure& elt)
{
  // ... ... ...

  // append a new measure to the current part
  fCurrentPart->
    createAMeasureAndAppendItToPart (
      inputLineNumber,
      fCurrentMeasureNumber,
      measureImplicitKind);

  // ... ... ...
}
```

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:

```
void msrPart::createAMeasureAndAppendItToPart (
  int    inputLineNumber,
  string measureNumber,
  msrMeasureImplicitKind
         measureImplicitKind)
{
  // ... ... ...

  // set part current measure number
```

```
  fPartCurrentMeasureNumber = measureNumber;

  // create and append measure in all the staves
  for (S_msrStaff staff : fPartAllStavesList) {
    staff->
      createAMeasureAndAppendItToStaff (
        inputLineNumber,
        measureNumber,
        measureImplicitKind);
  } // for

  // ... ... ...
```

38.9 Translating from MSR to MSR

This is done in src/passes/msr2msr/.

38.10 Translating from MSR to LPSR

This is done in src/passes/msr2lpsr/.

38.11 Translating from LPSR to LilyPond

This is done in src/passes/lpsr2lilypond/.

Chapter 39

Positions in measures

Positions in measures is stored in field `msrMeasureElement::fMeasureElementPositionInMeasure` in class `msrMeasureElement`, defined in src/formats/msr/msrMeasureElement.h/.cpp:

```
class EXP msrMeasureElement : public msrElement
{
  // ... ... ...

  protected:

    // protected fields
    // ------------------------------------------------------

    /*
      The measure uplink is declared in the sub-classes,
      to allow for separate *.h files, C++ constraint
    */

    rational               fMeasureElementSoundingWholeNotes;

    string                 fMeasureElementMeasureNumber;

    rational               fMeasureElementPositionInMeasure;
    rational               fMeasureElementPositionInVoice;

    msrMoment              fMeasureElementMomentInMeasure;
    msrMoment              fMeasureElementMomentInVoice;
};
```

39.1 Determining positions in measures

Chapter 40

Finalizations

40.1 Clones vs non-clones finalization

Finalizing clones may be simpler that 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:

```
void msrMeasure::finalizeMeasure (
  int                           inputLineNumber,
  msrMeasureRepeatContextKind measureRepeatContextKind,
  string                        context)
{
  // ... ... ...

    S_msrVoice
      voice =
        fMeasureSegmentUpLink->
          getSegmentVoiceUpLink ();

  // ... ... ...

    // delegate to voice kind specific methods
    switch (voice->getVoiceKind ()) {
      case msrVoiceKind::kVoiceKindRegular:
        finalizeRegularMeasure (
          inputLineNumber,
          measureRepeatContextKind,
          context);
        break;

      case msrVoiceKind::kVoiceKindDynamics:
        break;

      case msrVoiceKind::kVoiceKindHarmonies:
        finalizeHarmoniesMeasure (
          inputLineNumber,
          measureRepeatContextKind,
          context);
        break;

      case msrVoiceKind::kVoiceKindFiguredBass:
        finalizeFiguredBassMeasure (
          inputLineNumber,
          measureRepeatContextKind,
          context);
        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.

40.2 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::finalizeHarmoniesMeasure ()`, 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 cascadingfinalization is method `msrPart::finalizePartAndAllItsMeasures ()`:

```
void msrPart::finalizePartAndAllItsMeasures (
  int inputLineNumber)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceParts ()) {
    gLogStream <<
      "Finalizing part clone " <<
      getPartCombinedName () <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceVoices ()) {
    gLogStream <<
      "Finalizing all the measures of part \"" <<
      getPartCombinedName () <<
      "\", line " << inputLineNumber <<
      endl;
  }
#endif

  for (
    list<S_msrVoice>::const_iterator i = fPartAllVoicesList.begin ();
    i != fPartAllVoicesList.end ();
    ++i
  ) {
    S_msrVoice voice = (*i);

    voice->
      finalizeVoiceAndAllItsMeasures (
        inputLineNumber);
  } // for

  // collect the part measures slices from the staves
  collectPartMeasuresSlices (
    inputLineNumber);
}
```

40.3 Finalizing parts

Method `msrPart::finalizePart ()` warns if there are no staves in the part, and otherwise handles them, including cascadingto method `msrStaff::finalizeStaff ()`:

```
void msrPart::finalizePart (
  int inputLineNumber)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceParts ()) {
    gLogStream <<
      "Finalizing part " <<
      getPartCombinedName () <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  ++gIndenter;

  if (! getPartStaveNumbersToStavesMap.size ()) {
    stringstream s;

    s <<
      "Part " <<
      getPartCombinedName () <<
      " appears in the part list, but doesn't contain any stave";

    musicxmlWarning (
      gGlobalServiceRunData->getInputSourceName (),
      inputLineNumber,
      s.str ());
  }

  else {
    // sort the staves to have harmonies above and figured bass below the part
    fPartAllStavesList.sort (
      compareStavesToHaveFiguredBassElementsBelowCorrespondingPart);

    // finalize the staves
    for (
      map<int, S_msrStaff>::const_iterator i =
      getPartStaveNumbersToStavesMap.begin ();
      i != getPartStaveNumbersToStavesMap.end ();
      ++i
    ) {
      S_msrStaff staff = (*i).second;

      staff->
        finalizeStaff (
          inputLineNumber);
    } // for
  }

  // set score instrument names max lengthes if relevant
  setPartInstrumentNamesMaxLengthes ();

  // collect the part measures slices from the staves
  collectPartMeasuresSlices (
    inputLineNumber);

  --gIndenter;
}
```

40.4 Finalizing staves

Method `msrStaff::finalizeStaff ()` cascade s to method `msrVoice::finalizeVoice ()` and collects the staff measusres slices:

```
void msrStaff::finalizeStaff (int inputLineNumber)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceStaves ()) {
    gLogStream <<
      "Finalizing staff \"" <<
      getStaffName () << "\"" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  ++gIndenter;

  // finalize the voices
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceVoices ()) {
    gLogStream <<
      "Finalizing the voices in staff \"" <<
      getStaffName () << "\"" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  for (
    map<int, S_msrVoice>::const_iterator i =
      fStaffVoiceNumbersToAllVoicesMap.begin ();
    i != fStaffVoiceNumbersToAllVoicesMap.end ();
    ++i
  ) {
    S_msrVoice
      voice = (*i).second;

    voice->
      finalizeVoice (
        inputLineNumber);
  } // for

  // collect the staff measures slices from the voices
  collectStaffMeasuresSlices (
    inputLineNumber);

  --gIndenter;
}
```

40.5 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.

```
void msrVoice::finalizeVoice (
  int inputLineNumber)
{
  // ... ... ...

  if (fVoiceHasBeenFinalized) {
```

```
    stringstream s;

    s <<
      "Attempting to finalize  voice \"" <<
      asShortString () <<
      "\" more than once";

    msrInternalError (
      gGlobalServiceRunData->getInputSourceName (),
      fInputLineNumber,
      __FILE__, __LINE__,
      s.str ());
  }

  // set part shortest note duration if relevant
  S_msrPart
    voicePart =
      fetchVoicePartUpLink ();

  rational
    partShortestNoteDuration =
      voicePart->
        getPartShortestNoteDuration ();

  // ... ... ...

  if (fVoiceShortestNoteDuration < partShortestNoteDuration) {
    // set the voice part shortest note duration
    voicePart->
      setPartShortestNoteDuration (
        fVoiceShortestNoteDuration);

    // set the voice part shortest note tuplet factor // JMI
    voicePart->
      setPartShortestNoteTupletFactor (
        fVoiceShortestNoteTupletFactor);
  }

  // is this voice totally empty? this should be rare...
  if (
    fVoiceInitialElementsList.size () == 0
      &&
    fVoiceLastSegment->getSegmentMeasuresList ().size () == 0
  ) {
    stringstream s;

    s <<
      "Voice \"" <<
      getVoiceName () <<
      "\" is totally empty, no contents ever specified for it" <<
      endl;

    musicxmlWarning (
      gGlobalServiceRunData->getInputSourceName (),
      inputLineNumber,
      s.str ());
  }

  // are there pending repeats in the voice repeats stack???
  unsigned int voicePendingRepeatDescrsStackSize =
    fVoicePendingRepeatDescrsStack.size ();

  // ... ... ...

  // collect the voice measures into the flat list
  collectVoiceMeasuresIntoFlatList (
    inputLineNumber);
```

```
74
75    fVoiceHasBeenFinalized = true;
76
77    // ... ... ...
78  }
```

40.6 Finalizing repeats

40.7 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    string                        context)
5  {
6    if (fMeasureHasBeenFinalized) {
7      stringstream s;
8
9      s <<
10       "Attempting to finalize measure " <<
11       this->asShortString () <<
12       " more than once in segment '" <<
13       fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
14       "', context: " << context <<
15       "', measureFinalizationContext: " << fMeasureFinalizationContext <<
16       " in voice \"" <<
17       fMeasureSegmentUpLink->
18         getSegmentVoiceUpLink ()->
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         fMeasureSegmentUpLink->
35           getSegmentVoiceUpLink ();
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:
```

```
        break;

      case msrVoiceKind::kVoiceKindHarmonies:
        finalizeHarmoniesMeasure (
          inputLineNumber,
          measureRepeatContextKind,
          context);
        break;
      case msrVoiceKind::kVoiceKindFiguredBass:
        finalizeFiguredBassMeasure (
          inputLineNumber,
          measureRepeatContextKind,
          context);
        break;
    } // switch

    // position in voice
    rational
      positionInVoice =
        fetchMeasureVoiceUpLink ()->
          getCurrentPositionInVoice ();

    // assign measure' elements position in measure
    for (
      list<S_msrMeasureElement>::const_iterator i = fMeasureElementsList.begin ();
      i != fMeasureElementsList.end ();
      ++i
    ) {
      S_msrMeasureElement measureElement = (*i);

      measureElement->
        assignMeasureElementPositionInVoice (
          positionInVoice,
          "finalizeMeasure()");
    } // for

    // register finalization
    fMeasureHasBeenFinalized = true;
    fMeasureFinalizationContext = context;
  }
}
```

40.7.1 Finalizing regular measures

```
void msrMeasure::finalizeRegularMeasure (
  int                          inputLineNumber,
  msrMeasureRepeatContextKind measureRepeatContextKind,
  string                       context)
{
  // fetch the regular voice
  S_msrVoice
    voice =
      fMeasureSegmentUpLink->
        getSegmentVoiceUpLink ();

  // fetch the regular voice's part
  S_msrPart
    regularPart =
      voice->
        fetchVoicePartUpLink ();

  mfAssert (
    __FILE__, __LINE__,
    regularPart != nullptr,
```

```
    "regularPart is null");

  if (false) { // JMI
    gLogStream <<
      "---> regularPart: " <<
      endl;

    ++gIndenter;
    gLogStream <<
      regularPart <<
      endl;
    --gIndenter;
    gLogStream << endl;
  }

  rational
    measureWholeNotesDurationFromPartMeasuresVector =
      regularPart->
        getPartMeasuresWholeNotesDurationsVector () [
           fMeasureOrdinalNumberInVoice - 1 ];

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceMeasures ()) {
    gLogStream <<
      "Finalizing regular measure " <<
      this->asShortString () <<
      " in segment '" <<
      fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
      "' in regular voice \"" <<
      voice->getVoiceName () <<
      "\" (" << context << ")" <<
      ", measureWholeNotesDurationFromPartMeasuresVector: " <<
      measureWholeNotesDurationFromPartMeasuresVector <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  ++gIndenter;

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
    displayMeasure (
      inputLineNumber,
      "finalizeRegularMeasure() 1");
  }
#endif

  padUpToPositionAtTheEndOfTheMeasure (
    inputLineNumber,
    measureWholeNotesDurationFromPartMeasuresVector);

  // register this measures's length in the part
  S_msrPart
    part =
      this->fetchMeasurePartUpLink ();

  part->
    registerOrdinalMeasureNumberWholeNotesDuration (
      inputLineNumber,
      fMeasureOrdinalNumberInVoice,
      fCurrentMeasureWholeNotesDuration);

  // determine the measure kind and purist number
  determineMeasureKindAndPuristNumber (
    inputLineNumber,
    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-> getNoteSoundingWholeNotes ()
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           fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
134           "' in voice \"" <<
135           voice->getVoiceName () <<
136           "\", line " << inputLineNumber <<
137           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;
```

```
}
```

40.7.2 Finalizing harmonies measures

```
void msrMeasure::finalizeHarmoniesMeasure (
  int                           inputLineNumber,
  msrMeasureRepeatContextKind measureRepeatContextKind,
  string                        context)
{
  // fetch the harmonies voice
  S_msrVoice
    harmoniesVoice =
      fMeasureSegmentUpLink->
        getSegmentVoiceUpLink ();

  // fetch the harmonies part
  S_msrPart
    harmoniesPart =
      harmoniesVoice->
        fetchVoicePartUpLink ();

  mfAssert (
    __FILE__, __LINE__,
    harmoniesPart != nullptr,
    "harmoniesPart is null");

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
    gLogStream <<
      "Finalizing harmonies measure " <<
      this->asShortString () <<
      " in segment '" <<
      fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
      "' in harmonies voice \"" <<
      harmoniesVoice->getVoiceName () <<
      "\" (" << context << ")" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  ++gIndenter;

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
    displayMeasure (
      inputLineNumber,
      "finalizeHarmoniesMeasure() 1");
  }
#endif

#ifdef TRACING_IS_ENABLED
  // get the harmoniesPart number of measures
  int
    harmoniesPartNumberOfMeasures =
      harmoniesPart->
        getPartNumberOfMeasures ();

  if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
    gLogStream <<
      "fMeasureOrdinalNumberInVoice = " <<
      fMeasureOrdinalNumberInVoice <<
      ", harmoniesPartNumberOfMeasures = " <<
      harmoniesPartNumberOfMeasures <<
```

```
      endl;
  }
#endif

  // the measureWholeNotesDuration has to be computed
  rational
    measureWholeNotesDurationFromPartMeasuresVector =
      harmoniesPart->
        getPartMeasuresWholeNotesDurationsVector () [
           fMeasureOrdinalNumberInVoice - 1 ];

  // handle the harmonies in this measure
  finalizeHarmoniesInHarmoniesMeasure (
    inputLineNumber,
    context);

  // pad the measure up to measureWholeNotesDurationFromPartMeasuresVector
  padUpToPositionAtTheEndOfTheMeasure (
    inputLineNumber,
    measureWholeNotesDurationFromPartMeasuresVector);

  // determine the measure kind and purist number
  determineMeasureKindAndPuristNumber (
    inputLineNumber,
    measureRepeatContextKind);

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
    displayMeasure (
      inputLineNumber,
      "finalizeHarmoniesMeasure() 2");
  }
#endif

  --gIndenter;
}
```

40.7.3 Finalizing figured bass measures

```
void msrMeasure::finalizeFiguredBassMeasure (
  int                          inputLineNumber,
  msrMeasureRepeatContextKind measureRepeatContextKind,
  string                       context)
{
  // fetch the figured bass voice
  S_msrVoice
    figuredBassVoice =
      fMeasureSegmentUpLink->
        getSegmentVoiceUpLink ();

  // fetch the figured bass part
  S_msrPart
    figuredBassPart =
      figuredBassVoice->
        fetchVoicePartUpLink ();

  mfAssert (
    __FILE__, __LINE__,
    figuredBassPart != nullptr,
    "figuredBassPart is null");

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceFiguredBass ()) {
    gLogStream <<
```

```
      "Finalizing figured bass measure " <<
      this->asShortString () <<
      " in segment '" <<
      fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
      "' in figured bass voice \"" <<
      figuredBassVoice->getVoiceName () <<
      "\" (" << context << ")" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  ++gIndenter;

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceFiguredBassDetails ()) {
    displayMeasure (
      inputLineNumber,
      "finalizeFiguredBassMeasure() 1");
  }
#endif

#ifdef TRACING_IS_ENABLED
  // get the figuredBassPart number of measures
  int
    figuredBassPartNumberOfMeasures =
      figuredBassPart->
        getPartNumberOfMeasures ();

  if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
    gLogStream <<
      "fMeasureOrdinalNumberInVoice = " <<
      fMeasureOrdinalNumberInVoice <<
      ", figuredBassPartNumberOfMeasures = " <<
      figuredBassPartNumberOfMeasures <<
      endl;
  }
#endif

  // the measureWholeNotesDuration has to be computed
  rational
    measureWholeNotesDuration =
      figuredBassPart->
        getPartMeasuresWholeNotesDurationsVector () [
           fMeasureOrdinalNumberInVoice - 1 ];

  // handle the figured bass elements in this measure
  finalizeFiguredBassElementsInFiguredBassMeasure (
    inputLineNumber,
    context);

  // pad the measure up to fFullMeasureWholeNotesDuration
  padUpToPositionAtTheEndOfTheMeasure (
    inputLineNumber,
    measureWholeNotesDuration);

  // determine the measure kind and purist number
  determineMeasureKindAndPuristNumber (
    inputLineNumber,
    measureRepeatContextKind);

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceFiguredBassDetails ()) {
    displayMeasure (
      inputLineNumber,
      "finalizeFiguredBassMeasure() 2");
  }
```

```
93 #endif
94
95   --gIndenter;
96 }
```

40.8 Determining positions in measures

Chapter 41

Tempos handling

Tempos are presented at section 19.18, [Tempos], page 162.

41.1 Translating from MXSR to MSR (src/passes/mxsr2msr/)

41.2 Translating from MXSR to MSR (src/passes/mxsr2msr/)

41.3 Translating from MSR to MSR (src/passes/msr2msr/)

41.4 Translating from MSR to LPSR (src/passes/msr2lpsr/)

41.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)

Chapter 42

Positions in measures

Positions in measures is stored in field `msrMeasureElement::fMeasureElementPositionInMeasure` in class `msrMeasureElement`, defined in src/formats/msr/msrMeasureElement.h/.cpp:

```
class EXP msrMeasureElement : public msrElement
{
  // ... ... ...

  protected:

    // protected fields
    // ------------------------------------------------------

    /*
      The measure uplink is declared in the sub-classes,
      to allow for separate *.h files, C++ constraint
    */

    rational               fMeasureElementSoundingWholeNotes;

    string                 fMeasureElementMeasureNumber;

    rational               fMeasureElementPositionInMeasure;
    rational               fMeasureElementPositionInVoice;

    msrMoment              fMeasureElementMomentInMeasure;
    msrMoment              fMeasureElementMomentInVoice;
};
```

42.1 Determining positions in measures

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 180.

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:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'msrSegment::create (' *.cpp
msrSegments.cpp:S_msrSegment msrSegment::create (
msrSegments.cpp:      msrSegment::create (
msrSegments.cpp:      msrSegment::create (
msrVoices.cpp:        msrSegment::create (
msrVoices.cpp:    msrSegment::create (
msrVoices.cpp:    msrSegment::create (
msrVoices.cpp:      msrSegment::create (
```

Calls to method `msrSegment::createSegmentNewbornClone ()` occurs only when visiting class `msrSegment` instances in passes:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/passes > grep createSegmentNewbornClone */*
msr2bsr/msr2bsrTranslator.cpp:      elt->createSegmentNewbornClone (
msr2lpsr/msr2lpsrTranslator.cpp:    elt->createSegmentNewbornClone (
msr2msr/msr2msrTranslator.cpp:    elt->createSegmentNewbornClone (
```

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 :

```
void msrVoice::initializeVoice (
  msrVoiceCreateInitialLastSegmentKind
    voiceCreateInitialLastSegmentKind)
{
  // ... ... ...

    // create the initial last segment if requested
  switch (voiceCreateInitialLastSegmentKind) {
    case msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes:
      // sanity check // JMI LAST
      mfAssert (
        __FILE__, __LINE__,
        fVoiceLastSegment == nullptr,
        "fVoiceLastSegment is null");

      // create the last segment
      fVoiceLastSegment =
        msrSegment::create (
          fInputLineNumber,
          this);

      if (! fVoiceFirstSegment) {
        fVoiceFirstSegment = fVoiceLastSegment;
      }
      break;
    case msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentNo:
      break;
  } // switch

  // ... ... ...
};
```

Method `msrVoice::createMeasureRepeatFromItsFirstMeasures ()` is presented in section 46, [Measure repeats handling], page 278, 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:

```
void msrVoice::createNewLastSegmentForVoice (
  int    inputLineNumber,
  const string& context)
{
  // ... ... ...

  // create the last segment
  fVoiceLastSegment =
    msrSegment::create (
      inputLineNumber,
      this);

  if (! fVoiceFirstSegment) {
    fVoiceFirstSegment = fVoiceLastSegment;
  }

  // ... ... ...
}
```

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::handleFullMeasureRestsStartInVoiceClone ()`
- 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:

```
void msrVoice::createNewLastSegmentFromItsFirstMeasureForVoice (
  int           inputLineNumber,
  S_msrMeasure  firstMeasure,
  string        context)
{
  // create the last segment
  fVoiceLastSegment =
    msrSegment::create (
      inputLineNumber,
      this);

  if (! fVoiceFirstSegment) {
    fVoiceFirstSegment = fVoiceLastSegment;
  }

  // ... ... ...

  // append firstMeasure to fVoiceLastSegment
  fVoiceLastSegment->
    appendMeasureToSegment (firstMeasure);
```

```
21
22    // firstMeasure is the new voice last appended measure
23    fVoiceLastAppendedMeasure = firstMeasure;
24
25    // is firstMeasure the first one it 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 280), measure repeats (section 46, [Measure repeats handling], page 278) and full measure rests(section 65, [Full measure rests handling], page 331).

44.2 Appending measures to a segment

Method `msrSegment::assertSegmentMeasuresListIsNotEmpty ()` is called as a sanity check by many methods in src/formats/msr/msrSegments.cpp:

```
1  void msrSegment::assertSegmentMeasuresListIsNotEmpty (
2    int inputLineNumber) const
3  {
4    if (! fSegmentMeasuresList.size ()) {
5  #ifdef TRACING_IS_ENABLED
6    if (
7      gGlobalTracingOahGroup->getTraceMeasuresDetails ()
8        ||
9      gGlobalTracingOahGroup->getTraceSegmentsDetails ()
10       ||
11     gGlobalTracingOahGroup->getTraceRepeatsDetails ()
12   ) {
13     fSegmentVoiceUpLink->
14       displayVoiceRepeatsStackFullMeasureRestsMeasureRepeatAndVoice (
15         inputLineNumber,
16         "assertSegmentMeasuresListIsNotEmpty()");
17   }
18 #endif
19
20     gLogStream <<
21       "assertSegmentMeasuresListIsNotEmpty()" <<
22       ", fSegmentMeasuresList is empty" <<
23       ", segment: " <<
24       this->asString () <<
25       ", in voice \"" <<
26       fSegmentVoiceUpLink->getVoiceName () <<
27       "\"" <<
28       "', line " << inputLineNumber <<
29       endl;
30
31     mfAssert (
32       __FILE__, __LINE__,
33       false,
34       ", fSegmentMeasuresList is empty");
35   }
36 }
```

One such call is:

```
void msrSegment::appendKeyToSegment (S_msrKey key)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceKeys ()) {
    gLogStream <<
      "Appending key " << key->asString () <<
      " to segment " << asString () <<
    ", in voice \"" <<
    fSegmentVoiceUpLink->getVoiceName () <<
    "\"" <<
      endl;
  }
#endif

  // sanity check
  assertSegmentMeasuresListIsNotEmpty (
    key->getInputLineNumber ());

  ++gIndenter;

  // register key in segments's current measure
  fSegmentMeasuresList.back ()->
    appendKeyToMeasure (key);

  --gIndenter;
}
```

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 176.

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 177.

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

Full measure rests handling

Full measure rests are presented at section 19.34, [Full measure rests], page 178.

47.1 Translating from MXSR to MSR (src/passes/mxsr2msr/)

47.2 Translating from MXSR to MSR (src/passes/mxsr2msr/)

47.3 Translating from MSR to MSR (src/passes/msr2msr/)

47.4 Translating from MSR to LPSR (src/passes/msr2lpsr/)

47.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)

Chapter 48

Repeats handling

Repeats are presented at section 19.36, [Repeats], page 179.

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:

```
    // location
    enum msrBarLineLocationKind {
      kBarLineLocationNone,

      kBarLineLocationLeft,
      kBarLineLocationMiddle,
      kBarLineLocationRight // by default
    };
```

```
    // style
    enum msrBarLineStyleKind {
      kBarLineStyleNone,

      kBarLineStyleRegular,  // by default

      kBarLineStyleDotted, kBarLineStyleDashed, kBarLineStyleHeavy,
      kBarLineStyleLightLight, kBarLineStyleLightHeavy,
      kBarLineStyleHeavyLight, kBarLineStyleHeavyHeavy,
      kBarLineStyleTick, kBarLineStyleShort
    };
```

```
    // repeat direction
    enum msrBarLineRepeatDirectionKind {
      kBarLineRepeatDirectionNone,
      kBarLineRepeatDirectionForward, kBarLineRepeatDirectionBackward
    };
```

```
    // ending type
    enum msrBarLineEndingTypeKind {
      kBarLineEndingNone,

      kBarLineEndingTypeStart,
      kBarLineEndingTypeStop,
      kBarLineEndingTypeDiscontinue
    };
```

```
    // category
    enum msrBarLineCategoryKind {
      k_NoBarLineCategory,

      kBarLineCategoryStandalone,

      kBarLineCategoryRepeatStart, kBarLineCategoryRepeatEnd,

      kBarLineCategoryHookedEndingStart, kBarLineCategoryHookedEndingEnd,
      kBarLineCategoryHooklessEndingStart, kBarLineCategoryHooklessEndingEnd
    };
```

```
    // segno
    enum msrBarLineHasSegnoKind {
      kBarLineHasSegnoYes, kBarLineHasSegnoNo
    };
```

```
    // coda
    enum msrBarLineHasCodaKind {
      kBarLineHasCodaYes, kBarLineHasCodaNo
    };
```

```
    // repeat winged
    enum msrBarLineRepeatWingedKind {
      kBarLineRepeatWingedNone,

      kBarLineRepeatWingedStraight, kBarLineRepeatWingedCurved,
      kBarLineRepeatWingedDoubleStraight, kBarLineRepeatWingedDoubleCurved
    };
```

The attributes of `<barline/>` are deciphered upon the first visit of `S_barline` in src/passes/mxsr2msr/mxsr2msrTranslator.cpp:

```
void mxsr2msrTranslator::visitStart ( S_barline& elt )
{
  // ... ... ...

  // location

  {
    string
      location =
        elt->getAttributeValue ("location");

    fCurrentBarLineLocationKind =
      msrBarLine::kBarLineLocationRight; // by default

    if       (location == "left") {
      fCurrentBarLineLocationKind = msrBarLine::kBarLineLocationLeft;
    }
    else  if (location == "middle") {
      fCurrentBarLineLocationKind = msrBarLine::kBarLineLocationMiddle;
    }
    else if  (location == "right") {
      fCurrentBarLineLocationKind = msrBarLine::kBarLineLocationRight;
```

```
    }
    else {
      stringstream s;

      s <<
        "barLine location \"" << location <<
        "\" is unknown, using 'right' by default";

   // JMI   musicxmlError (
      musicxmlWarning (
        gGlobalServiceRunData->getInputSourceName (),
        inputLineNumber,
   //     __FILE__, __LINE__,
        s.str ());
    }
  }

  fOnGoingBarLine = true;
}
```

Then the class `msrBarLine` instance is created upon the second visit of `S_barline`:

```
void mxsr2msrTranslator::visitEnd ( S_barline& elt )
{
  // ... ... ...

  // create the barLine
  S_msrBarLine
    barLine =
      msrBarLine::create (
        inputLineNumber,
        fCurrentBarLineLocationKind,
        fCurrentBarLineStyleKind,
        fCurrentBarLineRepeatDirectionKind,
        fCurrentBarLineEndingTypeKind,
        fCurrentBarLineEndingNumber,
        fCurrentBarLineTimes,
        msrBarLine::k_NoBarLineCategory, // will be set afterwards
        fCurrentBarLineHasSegnoKind,
        fCurrentBarLineHasCodaKind,
        fCurrentBarLineRepeatWingedKind);

  // ... ... ...

  // wait until its category is defined
  // to append the barLine to the current segment

  // handle the barLine according to: JMI
  // http://www.musicxml.com/tutorial/the-midi-compatible-part/repeats/

  Bool barLineHasBeenHandled = false;

  switch (fCurrentBarLineLocationKind) {
    case msrBarLine::kBarLineLocationNone:
      // should not occur
      break;

    case msrBarLine::kBarLineLocationLeft:
      if (
        fCurrentBarLineEndingTypeKind
          ==
        msrBarLine::kBarLineEndingTypeStart
      ) {
        // ending start, don't know yet whether it's hooked or hookless
        // ------------------------------------------------------
        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          msrBarLine::kBarLineRepeatDirectionForward
67        ) {
68          // repeat start
69          // ------------------------------------------------------
70          // set the barLine category
71          barLine->
72            setBarLineCategory (
73              msrBarLine::kBarLineCategoryRepeatStart);
74
75          // handle the repeat start
76          handleRepeatStart (barLine);
77
78          barLineHasBeenHandled = true;
79        }
80        break;
81
82      case msrBarLine::kBarLineLocationMiddle:
83        // JMI ???
84        break;
85
86      case msrBarLine::kBarLineLocationRight:
87        {
88          if (
89            fCurrentBarLineEndingTypeKind == msrBarLine::kBarLineEndingTypeStop
90              &&
91            fCurrentBarLineEndingNumber.size () != 0
92          ) {
93            // hooked ending end
94            // ------------------------------------------------------
95            // set current barLine ending start category
96            fCurrentRepeatEndingStartBarLine->
97              setBarLineCategory (
98                msrBarLine::kBarLineCategoryHookedEndingStart);
99
100           // set this barLine's category
101           barLine->
102             setBarLineCategory (
103               msrBarLine::kBarLineCategoryHookedEndingEnd);
104
105           // handle the repeat hooked ending end
106           handleRepeatHookedEndingEnd (barLine);
107
108           barLineHasBeenHandled = true;
109         }
110
111         else if (
```

```
          fCurrentBarLineRepeatDirectionKind
            ==
          msrBarLine::kBarLineRepeatDirectionBackward
        ) {
          // repeat end
          // ------------------------------------------------------

          // set this barLine's category
          barLine->
            setBarLineCategory (
              msrBarLine::kBarLineCategoryRepeatEnd);

          // handle the repeat end
          handleRepeatEnd (barLine);

          barLineHasBeenHandled = true;
        }

        else if (
          fCurrentBarLineEndingTypeKind == msrBarLine::kBarLineEndingTypeDiscontinue
            &&
          fCurrentBarLineEndingNumber.size () != 0
        ) {
          // hookless ending end
          // ------------------------------------------------------
          // set current barLine ending start category
          fCurrentRepeatEndingStartBarLine->
            setBarLineCategory (
              msrBarLine::kBarLineCategoryHooklessEndingStart);

          // set this barLine's category
          barLine->
            setBarLineCategory (
              msrBarLine::kBarLineCategoryHooklessEndingEnd);

          // handle the repeat hookless ending end
          handleRepeatHooklessEndingEnd (barLine);

          barLineHasBeenHandled = true;
        }

        // forget about current repeat ending start barLine
        fCurrentRepeatEndingStartBarLine = nullptr;
      }
      break;
    } // switch

    // set the barLine category to stand alone if not yet handled
    if (! barLineHasBeenHandled) {
      switch (fCurrentBarLineStyleKind) {
        case msrBarLine::kBarLineStyleRegular:
        case msrBarLine::kBarLineStyleDotted:
        case msrBarLine::kBarLineStyleDashed:
        case msrBarLine::kBarLineStyleHeavy:
        case msrBarLine::kBarLineStyleLightLight:
        case msrBarLine::kBarLineStyleLightHeavy:
        case msrBarLine::kBarLineStyleHeavyLight:
        case msrBarLine::kBarLineStyleHeavyHeavy:
        case msrBarLine::kBarLineStyleTick:
        case msrBarLine::kBarLineStyleShort:
          barLine->
            setBarLineCategory (
              msrBarLine::kBarLineCategoryStandalone);

          // append the bar line to the current part
          // ... ... ...

```

```
        fCurrentPart->
          appendBarLineToPart (barLine);

        barLineHasBeenHandled = true;
        break;

      case msrBarLine::kBarLineStyleNone:
        stringstream s;

        s <<
          "barLine " <<
          barLine->asString () <<
          " has no barLine style";

        musicxmlWarning (
          gGlobalServiceRunData->getInputSourceName (),
          inputLineNumber,
    //      __FILE__, __LINE__,
          s.str ());
        break;
    } // switch
  }

  // has this barLine been handled?
  if (! barLineHasBeenHandled) {
    stringstream s;

    s << left <<
      "cannot handle a barLine containing: " <<
      barLine->asString ();

    msrInternalWarning (
      gGlobalServiceRunData->getInputSourceName (),
      inputLineNumber,
      s.str ());
  }

  fOnGoingBarLine = false;
}
```

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 173.

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 171.

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 171.

51.1 Parts browsing

Method `msrPart::browseData ()` defined in `src/formats/msr/msrParts.h/.cpp` is pecular in that it imposes a *partial order* on the part staves browsing:

```
void msrPart::browseData (basevisitor* v)
{
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "% ==> msrPart::browseData ()" <<
      endl;
  }

#ifdef TRACING_IS_ENABLED // JMI
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) { // JMI TEMP
    gLogStream <<
      "++++++++ fPartAllStavesList.size(): " <<
      fPartAllStavesList.size () <<
      endl;

    if (fPartAllStavesList.size ()) {
      for (S_msrStaff staff : fPartAllStavesList) {
        gLogStream <<
          endl <<
          "+++++++++ staff: ++++++++" <<
          " \"" << staff->getStaffName () << "\"" <<
          endl;
      } // for
    }

    gLogStream <<
      "++++++++ fPartNonHarmoniesNorFiguredBassStavesList.size(): " <<
      fPartNonHarmoniesNorFiguredBassStavesList.size () <<
      endl;

    if (fPartNonHarmoniesNorFiguredBassStavesList.size ()) {
      for (S_msrStaff staff : fPartNonHarmoniesNorFiguredBassStavesList) {
        gLogStream <<
          endl <<
          "+++++++++ staff: ++++++++" <<
          " \"" << staff->getStaffName () << "\"" <<
          endl;
      } // for
    }
```

```
  }
#endif

  /* don't enforce any order here, leave it to the client thru sorting JMI */

  // browse the part harmonies staff if any right now, JMI
  // to place it before the corresponding part
  if (fPartHarmoniesStaff) {
    msrBrowser<msrStaff> browser (v);
    browser.browse (*fPartHarmoniesStaff);
  }

  // browse all non harmonies and non figured bass staves
  for (S_msrStaff staff : fPartNonHarmoniesNorFiguredBassStavesList) {
    // browse the staff
    msrBrowser<msrStaff> browser (v);
    browser.browse (*staff);
  } // for

  // browse the part figured bass staff if any only now, JMI
  // to place it after the corresponding part
  if (fPartFiguredBassStaff) {
    msrBrowser<msrStaff> browser (v);
    browser.browse (*fPartFiguredBassStaff);
  }

//   // browse all the part staves JMI
//   for (S_msrStaff staff : fPartAllStavesList) {
//     if (staff != fPartHarmoniesStaff && staff != fPartFiguredBassStaff) {
//       // browse the staff
//       msrBrowser<msrStaff> browser (v);
//       browser.browse (*staff);
//     }
//   } // for
}
```

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

Scores handling

Scores are presented at section 19.24, [Scores], page 170.

52.1 Translating from MXSR to MSR (src/passes/mxsr2msr/)

52.2 Translating from MXSR to MSR (src/passes/mxsr2msr/)

52.3 Translating from MSR to MSR (src/passes/msr2msr/)

52.4 Translating from MSR to LPSR (src/passes/msr2lpsr/)

52.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)

Chapter 53

Books handling

Books are presented at section 19.23, [Books], page 170.

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

Ornaments handling

Ornaments are presented at section 19.40, [Ornaments], page 182.

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

Ties handling

Ties are presented at section 19.41, [Ties], page 182.

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

Dynamics handling

Dynamics are presented at section 19.42, [Dynamics], page 182.

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

Beams handling

Beams are presented at section 19.43, [Beams], page 182.

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

Slurs handling

Slurs are presented at section 19.44, [Slurs], page 182.

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

Grace notes groups handling

Grace notes groups are presented at section 19.45, [Grace notes groups], page 182.

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

Chords handling

Chords are presented at section 19.46, [Chords], page 182.

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

Tuplets handling

Tuplets are presented at section 19.47, [Tuplets], page 182.

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

Harmonies handling

Harmonies are presented at section 19.49, [Harmonies], page 185.

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:

```
    // harmonies
    void                    appendHarmonyToNoteHarmoniesList (
                              S_msrHarmony harmony);

    const list<S_msrHarmony>&
                            getNoteHarmoniesList () const
                                { return fNoteHarmoniesList; }

    // ... ... ...

    // harmonies
    // ------------------------------------------------------

    list<S_msrHarmony>    fNoteHarmoniesList;

    // ... ... ...
```

62.1 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:

```
  public:

    // constants
    // ------------------------------------------------------

    #define msrPart::K_PART_HARMONIES_STAFF_NUMBER       10
    #define msrPart::K_PART_HARMONIES_VOICE_NUMBER       11
```

In class `msrStaff`, there is:

```
    void                  registerHarmoniesVoiceByItsNumber (
                            int         inputLineNumber,
                            S_msrVoice  voice);
```

Class `msrPart` also contains:

```
   // harmonies

    S_msrVoice            createPartHarmoniesVoice (
                            int    inputLineNumber,
                            string currentMeasureNumber);

    void                  appendHarmonyToPart (
                            S_msrVoice  harmonySupplierVoice,
                            S_msrHarmony harmony);

    void                  appendHarmonyToPartClone (
                            S_msrVoice   harmonySupplierVoice,
                            S_msrHarmony harmony);
```

```
    // harmonies

    S_msrStaff            fPartHarmoniesStaff;
    S_msrVoice            fPartHarmoniesVoice;
```

62.2 Harmonies staves creation

This is done in src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.cpp.h/.cpp:

```
S_msrVoice mxsr2msrSkeletonBuilder::createPartHarmoniesVoiceIfNotYetDone (
  int         inputLineNumber,
  S_msrPart   part)
{
  // is the harmonies voice already present in part?
  S_msrVoice
    partHarmoniesVoice =
      part->
        getPartHarmoniesVoice ();

  if (! partHarmoniesVoice) {
    // create the harmonies voice and append it to the part
    partHarmoniesVoice =
      part->
        createPartHarmoniesVoice (
          inputLineNumber,
          fCurrentMeasureNumber);
```

```
  }

  return partHarmoniesVoice;
}
```

Method msrPartcreatePartHarmoniesVoice creates the part harmonies staff and the part harmonies voice, and then registers the latter in the former:

```
S_msrVoice msrPart::createPartHarmoniesVoice (
  int    inputLineNumber,
  string currentMeasureNumber)
{
  // ... ... ...

  // create the part harmonies staff
  int partHarmoniesStaffNumber =
    msrPart::K_PART_HARMONIES_STAFF_NUMBER;

  // ... ... ...

  fPartHarmoniesStaff =
    addHarmoniesStaffToPart (
      inputLineNumber);

  // ... ... ...

  // create the part harmonies voice
  int partHarmoniesVoiceNumber =
    msrPart::K_PART_HARMONIES_VOICE_NUMBER;

  // ... ... ...

  fPartHarmoniesVoice =
    msrVoice::create (
      inputLineNumber,
      msrVoiceKind::kVoiceKindHarmonies,
      partHarmoniesVoiceNumber,
      msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes,
      fPartHarmoniesStaff);

  // register the part harmonies voice in part harmonies staff
  fPartHarmoniesStaff->
    registerVoiceInStaff (
      inputLineNumber,
      fPartHarmoniesVoice);

  // ... ... ...

  return fPartHarmoniesVoice;
}
```

62.3 Harmonies creation

There are several methods for harmonies creation:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep create formats/msr/msrHarmonies.h
    static SMARTP<msrHarmonyDegree> create (
    static SMARTP<msrHarmony> createWithoutVoiceUplink (
    SMARTP<msrHarmony> createHarmonyNewbornClone (
//     SMARTP<msrHarmony> createHarmonyDeepClone ( // JMI ???
    static SMARTP<msrHarmony> createWithVoiceUplink (
```

62.4 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 harmoniy 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::fThereAreHarmoniesToBeAttachedToCurrentNot it used to account for that:

```
void mxsr2msrSkeletonBuilder::visitStart ( S_harmony& elt )
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
    gLogStream <<
      "--> Start visiting S_harmony" <<
      ", harmoniesVoicesCounter = " << fHarmoniesVoicesCounter <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  /* JMI ???
    several harmonies can be attached to a given note,
    leading to as many harmonies voices in the current part
  */

  // take harmonies voice into account
  ++fHarmoniesVoicesCounter; // UNUSED JMI

  fThereAreHarmoniesToBeAttachedToCurrentNote = true;
}
```

Upon the second visit of class msrNote, the part harmonies voice is created if harmonies are not to be ignored due to option option -ignore-harmonies, -oharms and it has not been created yet:

```
void mxsr2msrSkeletonBuilder::visitEnd ( S_note& elt )
{
  // ... ... ...

  // are there harmonies attached to the current note?
  if (fThereAreHarmoniesToBeAttachedToCurrentNote) {
    if (gGlobalMxsr2msrOahGroup->getIgnoreHarmonies ()) {
#ifdef TRACING_IS_ENABLED
      if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
        gLogStream <<
          "Ignoring the harmonies" <<
          ", line " <<
          inputLineNumber <<
          endl;
      }
#endif
    }
    else {
      // create the part harmonies voice if not yet done
      S_msrVoice
        partHarmoniesVoice =
          createPartHarmoniesVoiceIfNotYetDone (
            inputLineNumber,
            fCurrentPart);
    }

    fThereAreHarmoniesToBeAttachedToCurrentNote = false;
```

```
  }

  // ... ... ...
}
```

Creating the part harmonies voice is delegated to the part:

```
S_msrVoice mxsr2msrSkeletonBuilder::createPartHarmoniesVoiceIfNotYetDone (
  int        inputLineNumber,
  S_msrPart  part)
{
  // is the harmonies voice already present in part?
  S_msrVoice
    partHarmoniesVoice =
      part->
        getPartHarmoniesVoice ();

  if (! partHarmoniesVoice) {
    // create the harmonies voice and append it to the part
    partHarmoniesVoice =
      part->
        createPartHarmoniesVoice (
          inputLineNumber,
          fCurrentMeasureNumber);
  }

  return partHarmoniesVoice;
}
```

62.5 Translating harmonies from MXSR to MSR

This is done in src/passes/mxsr2msr/.

This is where the class `msrHarmony` instances are created.

62.5.1 First `S_harmony` visit

The first visit of `S_harmony` initializes the fields storing values to be gathered visiting subelements:

```
void mxsr2msrTranslator::visitStart ( S_harmony& elt )
{
  int inputLineNumber =
    elt->getInputLineNumber ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
    gLogStream <<
      "--> Start visiting S_harmony" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  ++fHarmoniesVoicesCounter;

  fCurrentHarmonyInputLineNumber       = inputLineNumber;

  fCurrentHarmonyRootDiatonicPitchKind = msrDiatonicPitchKind::k_NoDiatonicPitch;
  fCurrentHarmonyRootAlterationKind    = msrAlterationKind::kAlterationNatural;
  fCurrentHarmonyKind                  = msrHarmonyKind::k_NoHarmony;
  fCurrentHarmonyKindText              = "";
```

```
  fCurrentHarmonyInversion              = K_HARMONY_NO_INVERSION;
  fCurrentHarmonyBassDiatonicPitchKind = msrDiatonicPitchKind::k_NoDiatonicPitch;
  fCurrentHarmonyBassAlterationKind     = msrAlterationKind::kAlterationNatural;
  fCurrentHarmonyDegreeValue            = -1;
  fCurrentHarmonyDegreeAlterationKind   = msrAlterationKind::kAlterationNatural;

  fCurrentHarmonyWholeNotesOffset = rational (0, 1);

  fOnGoingHarmony = true;
}
```

62.5.2 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::createWithoutVoiceUplink ()`:

```
void mxsr2msrTranslator::visitEnd ( S_harmony& elt )
{
  // ... ... ...

  if (gGlobalMxsr2msrOahGroup->getIgnoreHarmonies ()) {
 #ifdef TRACING_IS_ENABLED
    if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
      gLogStream <<
        "Ignoring harmony" <<
        ", line " <<
        inputLineNumber <<
        endl;
    }
#endif
  }
  else {
   // create the harmony
#ifdef TRACING_IS_ENABLED
    if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
      gLogStream <<
        "Creating a harmony" <<
        ", line " << inputLineNumber << ":" <<
        endl;

      // ... ... ...
    }
#endif

    S_msrHarmony
      harmony =
        msrHarmony::createWithoutVoiceUplink (
          fCurrentHarmonyInputLineNumber,
          // no harmoniesVoiceUpLink yet

          fCurrentHarmonyRootQuarterTonesPitchKind,

          fCurrentHarmonyKind,
          fCurrentHarmonyKindText,

          fCurrentHarmonyInversion,

          fCurrentHarmonyBassQuarterTonesPitchKind,

          rational (1, 1),            // harmonySoundingWholeNotes,
                                      // will be set upon next note handling
          rational (1, 1),            // harmonyDisplayWholeNotes,
                                      // will be set upon next note handling
```

```
          fCurrentHarmoniesStaffNumber,
          msrTupletFactor (1, 1),     // will be set upon next note handling
          fCurrentHarmonyWholeNotesOffset);

    // append pending harmony degrees if any to the harmony
    if (! fCurrentHarmonyDegreesList.size ()) {
#ifdef TRACING_IS_ENABLED
      if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
        musicxmlWarning (
          gGlobalServiceRunData->getInputSourceName (),
          inputLineNumber,
          "harmony has no degrees contents");
      }
#endif
    }

    else {
      // handle harmony degrees if any
      while (fCurrentHarmonyDegreesList.size ()) {
        S_msrHarmonyDegree
          harmonyDegree =
            fCurrentHarmonyDegreesList.front ();

        // ... ... ...

        // append it to harmony's degrees list
        harmony->
          appendHarmonyDegreeToHarmony (
            harmonyDegree);

        // remove it from the list
        fCurrentHarmonyDegreesList.pop_front ();
      } // while
    }

    // attach the current frame if any to the harmony
    if (fCurrentFrame) {
      harmony->setHarmonyFrame (fCurrentFrame);
    }

    // append the harmony to the pending harmonies list
    fPendingHarmoniesList.push_back (harmony);
  }

  fOnGoingHarmony = false;
}
```

62.5.3 Attaching msrHarmony instances to notes

msrHarmony

The contents of mxsr2msrTranslatorfPendingHarmoniesList is attached to the class msrNote instance in method mxsr2msrTranslator::populateNote ():

```
void mxsr2msrTranslator::populateNote (
  int        inputLineNumber,
  S_msrNote newNote)
{
  // ... ... ...

  // handle the pending harmonies if any
  if (fPendingHarmoniesList.size ()) {
    // get voice to insert harmonies into
    S_msrVoice
```

```
      voiceToInsertHarmoniesInto =
        fCurrentPart->
          getPartHarmoniesVoice ();

    // ... ... ...

    handlePendingHarmonies (
      newNote,
      voiceToInsertHarmoniesInto);

    // reset harmony counter
    fHarmoniesVoicesCounter = 0;
  }
}
```

62.5.4 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::appendHarmonyToNoteHarmoniesList ()`:

```
void mxsr2msrTranslator::handlePendingHarmonies (
  S_msrNote  newNote,
  S_msrVoice voiceToInsertInto)
{
  // ... ... ...

  rational
    newNoteSoundingWholeNotes =
      newNote->
        getNoteSoundingWholeNotes (),
    newNoteDisplayWholeNotes =
      newNote->
        getNoteDisplayWholeNotes ();

  while (fPendingHarmoniesList.size ()) { // recompute at each iteration
    S_msrHarmony
      harmony =
        fPendingHarmoniesList.front ();

    /*
       MusicXML harmonies don't have a duration,
       and MSR could follow this line, but LilyPond needs one...
       So:
         - we register all harmonies with the duration of the next note
         - they will be sorted by position in the measure in finalizeMeasure(),
           at which time their duration may be shortened
           so that the offsets values are enforced
           and they don't overflow the measure
      It is VITAL that harmonies measures be finalized
      AFTER the corresponding measure in the regular voice,
      since the current sounding whole notes of the latter is needed for that
    */

    // set the harmony's sounding whole notes
    harmony->
      setHarmonySoundingWholeNotes (
        newNoteSoundingWholeNotes);

    // set the harmony's display whole notes JMI useless???
    harmony->
      setHarmonyDisplayWholeNotes (
        newNoteDisplayWholeNotes);
```

```
43
44      // set the harmony's tuplet factor
45      harmony->
46        setHarmonyTupletFactor (
47          msrTupletFactor (
48            fCurrentNoteActualNotes,
49            fCurrentNoteNormalNotes));
50
51      // attach the harmony to newNote
52      newNote->
53        appendHarmonyToNoteHarmoniesList (
54          harmony);
55
56      // get the part harmonies voice
57      S_msrVoice
58        partHarmoniesVoice =
59          fCurrentPart->
60            getPartHarmoniesVoice ();
61
62      // sanity check
63      mfAssert (
64        __FILE__, __LINE__,
65        partHarmoniesVoice != nullptr,
66        "partHarmoniesVoice is null");
67
68      // set the harmony's voice upLink
69      // only now that we know which harmonies voice will contain it
70      harmony->
71        setHarmoniesVoiceUpLink (
72          partHarmoniesVoice);
73
74  /* JMI CAFE
75      // append the harmony to the part harmonies voice
76      partHarmoniesVoice->
77        appendHarmonyToVoice (
78          harmony);
79  */
80      // don't append the harmony to the part harmonies voice // BLARK
81      // before the note itself has been appended to the voice
82
83      // remove the harmony from the list
84      fPendingHarmoniesList.pop_front ();
85    } // while
86  }
```

62.5.5 First S_harmony visit

msrHarmony

Method msrNote::appendHarmonyToNoteHarmoniesList () is where the harmony's note uplink is set:

```
1  void msrNote::appendHarmonyToNoteHarmoniesList (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       endl;
12   }
13 #endif
```

```
14
15    // update the harmony whole notes if it belongs to a tuplet ??? utf8.xml JMI
16
17    fNoteHarmoniesList.push_back (harmony);
18
19    // register this note as the harmony note upLink
20    harmony->
21      setHarmonyNoteUpLink (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 position in measure when setting the chord's one.

```
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 list<S_msrHarmony>&
7      noteHarmoniesList =
8        note->getNoteHarmoniesList ();
9
10   if (noteHarmoniesList.size ()) {
11     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           endl;
24       }
25 #endif
26
27       chord->
28         appendHarmonyToChord (harmony);
29
30     } // for
31   }
32 }
```

62.5.6 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 (S_msrNote note)
2  {
3    // ... ... ...
4
5    // are there harmonies attached to this note? // BLARK
6    const list<S_msrHarmony>&
7      noteHarmoniesList =
8        note->
9          getNoteHarmoniesList ();
10
```

```
  if (noteHarmoniesList.size ()) {
    // get the current part's harmonies voice
    S_msrVoice
      partHarmoniesVoice =
        part->
          getPartHarmoniesVoice ();

    for (S_msrHarmony harmony : noteHarmoniesList) {
      // append the harmony to the part harmonies voice
      partHarmoniesVoice->
        appendHarmonyToVoice (
          harmony);
    } // for
  }

  // ... ... ...
};
```

62.6 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:

```
void msr2msrTranslator::visitStart (S_msrHarmony& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrHarmony '" <<
      elt->asString () <<
      ", fOnGoingNonGraceNote: " << fOnGoingNonGraceNote <<
      ", fOnGoingChord: " << fOnGoingChord <<
      ", fOnGoingHarmoniesVoice: " << fOnGoingHarmoniesVoice <<
      ", fOnGoingHarmony: " << fOnGoingHarmony <<
      "', line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  // create a harmony new born clone
  fCurrentHarmonyClone =
    elt->
      createHarmonyNewbornClone (
        fCurrentVoiceClone);

  if (fOnGoingNonGraceNote) {
    // register the harmony in the current non-grace note clone
    fCurrentNonGraceNoteClone->
      appendHarmonyToNoteHarmoniesList (
        fCurrentHarmonyClone);

    // don't append the harmony to the part harmony,
    // this has been done in pass2b // JMI ???
  }

  else if (fOnGoingChord) {
    // register the harmony in the current chord clone
    fCurrentChordClone->
      appendHarmonyToChord (fCurrentHarmonyClone); // JMI
  }
```

```
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
     JMI
55              elt->                getHarmonyDisplayWholeNotes (),
56              0, // JMI elt->                getHarmonyDotsNumber (),
57              fCurrentVoiceClone-> getRegularVoiceStaffSequentialNumber (), // JMI
58              fCurrentVoiceClone-> getVoiceNumber ());
59
60        // append it to the current voice clone
61        // to 'push' the harmony aside
62        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      stringstream s;
75
76      s <<
77        "harmony is out of context, cannot be handled: '" <<
78        elt->asShortString () <<
79        "'";
80
81      msrInternalError (
82        gGlobalServiceRunData->getInputSourceName (),
83        elt->getInputLineNumber (),
84        __FILE__, __LINE__,
85        s.str ());
86    }
87
88    fOnGoingHarmony = true;
89  }
```

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       endl;
```

```
  }
#endif

  fCurrentHarmonyClone = nullptr;
  fOnGoingHarmony = false;
}
```

62.7 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: :

```
void msr2lpsrTranslator::visitStart (S_msrHarmony& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrHarmony '" <<
      elt->asString () <<
      ", onGoingNonGraceNote: " << fOnGoingNonGraceNote <<
      ", onGoingChord: " << fOnGoingChord <<
      ", onGoingHarmoniesVoice: " << fOnGoingHarmoniesVoice <<
      ", onGoingHarmony: " << fOnGoingHarmony <<
      "', line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  // create a harmony new born clone
  fCurrentHarmonyClone =
    elt->
      createHarmonyNewbornClone (
        fCurrentVoiceClone);

  if (fOnGoingNonGraceNote) {
    // register the harmony in the current non-grace note clone
    fCurrentNonGraceNoteClone->
      appendHarmonyToNoteHarmoniesList (
        fCurrentHarmonyClone);

    // don't append the harmony to the part harmony,
    // this has been done in pass2b // JMI ???
  }

  else if (fOnGoingChord) {
    // register the harmony in the current chord clone
    fCurrentChordClone->
      appendHarmonyToChord (fCurrentHarmonyClone); // JMI
  }

  else if (fOnGoingHarmoniesVoice) {
  /* JMI
    // get the harmony whole notes offset
    rational
      harmonyWholeNotesOffset =
        elt->getHarmonyWholeNotesOffset ();

    // is harmonyWholeNotesOffset not equal to 0?
    if (harmonyWholeNotesOffset.getNumerator () != 0) {
      // create skip with duration harmonyWholeNotesOffset
```

```
      S_msrNote
        skip =
          msrNote::createSkipNote (
            elt->                getInputLineNumber (),
            "666", // JMI elt->                getHarmoniesMeasureNumber (),
            elt->                getHarmonyDisplayWholeNotes (), // would be 0/1 otherwise
    JMI
            elt->                getHarmonyDisplayWholeNotes (),
            0, // JMI elt->                getHarmonyDotsNumber (),
            fCurrentVoiceClone-> getRegularVoiceStaffSequentialNumber (), // JMI
            fCurrentVoiceClone-> getVoiceNumber ());

      // append it to the current voice clone
      // to 'push' the harmony aside
      fCurrentVoiceClone->
        appendNoteToVoice (skip);
    }
*/

    // append the harmony to the current voice clone
    fCurrentVoiceClone->
      appendHarmonyToVoiceClone (
        fCurrentHarmonyClone);
  }

  else {
    stringstream s;

    s <<
      "harmony is out of context, cannot be handled: '" <<
      elt->asShortString () <<
      "'";

    msrInternalError (
      gGlobalServiceRunData->getInputSourceName (),
      elt->getInputLineNumber (),
      __FILE__, __LINE__,
      s.str ());
  }

  fOnGoingHarmony = true;
}
```

Here too, there are only fields updates upon the second visit of `S_msrHarmony` instances:

```
void msr2lpsrTranslator::visitEnd (S_msrHarmony& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> End visiting msrHarmony '" <<
      elt->asString () <<
      "'" <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  fCurrentHarmonyClone = nullptr;
  fOnGoingHarmony = false;
}
```

62.8 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:

```
void lpsr2lilypondTranslator::visitStart (S_msrHarmony& elt)
{
#ifdef TRACING_IS_ENABLED
  {
    Bool
      traceMsrVisitors =
        gGlobalMsrOahGroup->
          getTraceMsrVisitors (),
      generateMsrVisitingInformation =
        gGlobalLpsr2lilypondOahGroup->
          getGenerateMsrVisitingInformation ();

    if (traceMsrVisitors || generateMsrVisitingInformation) {
      stringstream s;

      s <<
        "% --> Start visiting msrHarmony '" <<
        elt->asString () <<
        "'" <<
        ", fOnGoingNotesStack.size () = " <<
        fOnGoingNotesStack.size () <<
        ", fOnGoingChord = " <<
        fOnGoingChord <<
        ", fOnGoingHarmoniesVoice = " <<
        fOnGoingHarmoniesVoice <<
        ", line " << elt->getInputLineNumber () <<
        endl;

      if (traceMsrVisitors) {
        gLogStream << s.str ();
      }

      if (generateMsrVisitingInformation) {
        fLilypondCodeStream << s.str ();
      }
    }
```

```
  }
#endif

  if (fOnGoingNotesStack.size () > 0) {
  /* JMI
#ifdef TRACING_IS_ENABLED
    if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
      fLilypondCodeStream <<
        "%{ fOnGoingNotesStack.size () S_msrHarmony JMI " <<
        elt->asString () <<
        " %}" <<
        endl;
    }
#endif
*/
  }

  else if (fOnGoingChord) { // JMI
  }

  else if (fOnGoingHarmoniesVoice) {
    // actual LilyPond code generation
    fLilypondCodeStream <<
      harmonyAsLilypondString (elt) <<
      ' ';

    // generate the input line number as comment if relevant
    if (
      gGlobalLpsr2lilypondOahGroup->getInputLineNumbers ()
        ||
      gGlobalLpsr2lilypondOahGroup->getGeneratePositionsInMeasures ()
    ) {
      generateInputLineNumberAndOrPositionInMeasureAsAComment (
        elt);
    }
  }
}
```

Chapter 63

Figured bass elements handling

Figured bass elements are presented at section 19.50, [Figured bass elements], page 185.

The useful options here are:

- option `-trace-figured-bass, -tfigbass`
- 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-2, -dmsr2`
- option `-display-msr-2-short, -dmsr2s` and option `-display-msr-2-details, -dmsr2d`

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:

```
    // figured bass
    void                   appendFiguredBassToNoteFiguredBassElementsList (
                             S_msrFiguredBassElement figuredBass);

    const list<S_msrFiguredBassElement>&
                           getNoteFiguredBassElementsList () const
                               { return fNoteFiguredBassElementsList; }

    // ... ... ...

    // figured bass
    // ------------------------------------------------------

    list<S_msrFiguredBassElement>
                           fNoteFiguredBassElementsList;
```

63.1 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:

```
  public:

    // constants
    // ------------------------------------------------------

    // ... ... ...

    #define msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER  20
    #define msrPart::K_PART_FIGURED_BASS_VOICE_NUMBER  21
```

In class `msrStaff`, there is:

```
     void                    registerFiguredBassVoiceByItsNumber (
                              int         inputLineNumber,
                              S_msrVoice  voice);
```

Class `msrPart` also contains:

```
    // figured bass

    S_msrVoice              createPartFiguredBassVoice (
                              int    inputLineNumber,
                              string currentMeasureNumber);

    void                    appendFiguredBassToPart (
                              S_msrVoice              figuredBassSupplierVoice,
                             S_msrFiguredBassElement figuredBass);

    void                    appendFiguredBassToPartClone (
                              S_msrVoice              figuredBassSupplierVoice,
                              S_msrFiguredBassElement figuredBass);
```

```
    // figured bass

    S_msrStaff              fPartFiguredBassStaff;
    S_msrVoice              fPartFiguredBassVoice;
```

63.2 Figured bass staves creation

This is done in src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.cpp.h/.cpp:

```
S_msrVoice mxsr2msrSkeletonBuilder::createPartFiguredBassVoiceIfNotYetDone (
  int         inputLineNumber,
  S_msrPart   part)
{
  // is the figured bass voice already present in part?
  S_msrVoice
    partFiguredBassVoice =
      part->
        getPartFiguredBassVoice ();

  if (! partFiguredBassVoice) {
    // create the figured bass voice and append it to the part
    partFiguredBassVoice =
      part->
        createPartFiguredBassVoice (
```

```
          inputLineNumber,
          fCurrentMeasureNumber);
  }

  return partFiguredBassVoice;
}
```

Method `msrPart::createPartFiguredBassVoice ()` creates the part figured bass staff and the part figured bass voice, and then registers the latter in the former:

```
S_msrVoice msrPart::createPartFiguredBassVoice (
  int    inputLineNumber,
  string currentMeasureNumber)
{
  // ... ... ...

  // create the part figured bass staff
  int partFiguredBassStaffNumber =
    msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER;

  // ... ... ...

  fPartFiguredBassStaff =
    addHFiguredBassStaffToPart (
      inputLineNumber);

  // ... ... ...

  // create the figured bass voice
  int partFiguredBassVoiceNumber =
    msrPart::K_PART_FIGURED_BASS_VOICE_NUMBER;

  // ... ... ...

  fPartFiguredBassVoice =
    msrVoice::create (
      inputLineNumber,
      msrVoiceKind::kVoiceKindFiguredBass,
      partFiguredBassVoiceNumber,
      msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes,
      fPartFiguredBassStaff);

  // register the figured bass voice in the part figured bass staff
  fPartFiguredBassStaff->
    registerVoiceInStaff (
      inputLineNumber,
      fPartFiguredBassVoice);

  // ... ... ...

  return fPartFiguredBassVoice;
}
```

63.3 Figured bass elements creation

There several methods for Figured bass elements creation:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep create formats/msr/
    msrFiguredBassElements.h
    static SMARTP<msrBassFigure> create (
    SMARTP<msrBassFigure> createFigureNewbornClone (
//      SMARTP<msrBassFigure> createFigureDeepClone ( // JMI ???
    static SMARTP<msrFiguredBassElement> create (
```

```
    static SMARTP<msrFiguredBassElement> create (
    SMARTP<msrFiguredBassElement> createFiguredBassElementNewbornClone (
//      SMARTP<msrFiguredBassElement> createFiguredBassElementDeepClone ();
```

63.4 Translating figured bass elements from MXSR to MSR

This is done in src/passes/mxsr2msr/, and this is where the class `msrFiguredBassElement` instances are created.

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.
FieldmxsrmsrSkeletonBuilderfThereAreFiguredBassToBeAttachedToCurrentNote it used when visiting an `S_FiguredBassElement` element to account for that:

```
void mxsr2msrSkeletonBuilder::visitStart ( S_figured_bass& elt )
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
    gLogStream <<
      "--> Start visiting S_figured_bass" <<
      ", figuredBassVoicesCounter = " << fFiguredBassVoicesCounter <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  /* JMI
    several figured bass elements can be attached to a given note,
    leading to as many figured bass voices in the current part JMI TRUE???
  */

  // take figured bass voice into account
  ++fFiguredBassVoicesCounter;

  fThereAreFiguredBassToBeAttachedToCurrentNote = true;
}
```

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 option `-ignore-figured-bass, -ofigbass` and it has not been created yet:

```
void mxsr2msrSkeletonBuilder::visitEnd ( S_note& elt )
{
  // ... ... ...

  // are there figured bass attached to the current note?
  if (fThereAreFiguredBassToBeAttachedToCurrentNote) {
    if (gGlobalMxsr2msrOahGroup->getIgnoreFiguredBassElements ()) {
#ifdef TRACING_IS_ENABLED
      if (gGlobalTracingOahGroup->getTraceFiguredBass ()) {
        gLogStream <<
          "Ignoring the figured bass elements" <<
          ", line " <<
          inputLineNumber <<
          endl;
      }
#endif
    }
```

```
    else {
      // create the part figured bass voice if not yet done
      S_msrVoice
        partFiguredBassVoice =
          createPartFiguredBassVoiceIfNotYetDone (
            inputLineNumber,
            fCurrentPart);
    }

    fThereAreFiguredBassToBeAttachedToCurrentNote = false;

  // ... ... ...
}
```

Creating the part figured bass voice is delegated to the part:

```
S_msrVoice mxsr2msrSkeletonBuilder::createPartFiguredBassVoiceIfNotYetDone (
  int         inputLineNumber,
  S_msrPart   part)
{
  // is the figured bass voice already present in part?
  S_msrVoice
    partFiguredBassVoice =
      part->
        getPartFiguredBassVoice ();

  if (! partFiguredBassVoice) {
    // create the figured bass voice and append it to the part
    partFiguredBassVoice =
      part->
        createPartFiguredBassVoice (
          inputLineNumber,
          fCurrentMeasureNumber);
  }

  return partFiguredBassVoice;
}
```

63.4.1 First S_figured_bass visit

The first visit of `S_figured_bass` initializes the fields storing values to be gathered visiting subelements:

```
void mxsr2msrTranslator::visitStart ( S_figured_bass& elt )
{
  int inputLineNumber =
    elt->getInputLineNumber ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
    gLogStream <<
      "--> Start visiting S_figured_bass" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  ++fFiguredBassVoicesCounter;

  string parentheses = elt->getAttributeValue ("parentheses");

  fCurrentFiguredBassParenthesesKind =
    msrFiguredBassElement::kFiguredBassElementParenthesesNo; // default value

  if (parentheses.size ()) {
```

```
    if (parentheses == "yes")
      fCurrentFiguredBassParenthesesKind =
        msrFiguredBassElement::kFiguredBassElementParenthesesYes;

    else if (parentheses == "no")
     fCurrentFiguredBassParenthesesKind =
        msrFiguredBassElement::kFiguredBassElementParenthesesNo;

    else {
      stringstream s;

      s <<
        "parentheses value " << parentheses <<
        " should be 'yes' or 'no'";

      musicxmlError (
        gGlobalServiceRunData->getInputSourceName (),
        inputLineNumber,
        __FILE__, __LINE__,
        s.str ());
    }
  }

  fCurrentFiguredBassInputLineNumber   = -1;

  fCurrentFigureNumber = -1;

  fCurrentFigurePrefixKind = msrBassFigure::k_NoFigurePrefix;
  fCurrentFigureSuffixKind = msrBassFigure::k_NoFigureSuffix;

  fCurrentFiguredBassSoundingWholeNotes = rational (0, 1);
  fCurrentFiguredBassDisplayWholeNotes  = rational (0, 1);

  fOnGoingFiguredBass = true;
}
```

63.4.2 Second `S_figured_bass` visit

Upon the second visit of `S_figured_bass` , the class `msrFiguredBassElement` instance is created, populated and appended to `mxsr2msrTranslator`fPendingFiguredBassElementsList:

```
void mxsr2msrTranslator::visitEnd ( S_figured_bass& elt )
{
  int inputLineNumber =
    elt->getInputLineNumber ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
    gLogStream <<
      "--> End visiting S_figured_bass" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  // create the figured bass element
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceFiguredBass ()) {
    gLogStream <<
      "Creating a figured bass" <<
      ", line " << inputLineNumber << ":" <<
      endl;
  }
#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_msrFiguredBassElement
29      figuredBassElement =
30        msrFiguredBassElement::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        figuredBassElement->
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      fPendingFiguredBassElementsList.push_back (figuredBassElement);
57    }
58
59    fOnGoingFiguredBass = false;
60  }
```

63.4.3 Attaching `msrFiguredBassElement` instances to notes

`msrFiguredBassElement`

The contents of `mxsr2msrTranslatorfPendingFiguredBassElementsList` is attached to the class `msrNote` instance in method
method `mxsr2msrTranslator::populateNote ()`:

```
1  void mxsr2msrTranslator::populateNote (
2    int        inputLineNumber,
3    S_msrNote newNote)
4  {
5    // ... ... ...
6
7    // handle the pending figured bass elements if any
8    if (fPendingFiguredBassElementsList.size ()) {
9      // get voice to insert figured bass elements into
10     S_msrVoice
11       voiceToInsertFiguredBassElementsInto =
12         fCurrentPart->
13           getPartFiguredBassVoice ();
14
15     // ... ... ...
16
17     handlePendingFiguredBassElements (
18       newNote,
```

```
      voiceToInsertFiguredBassElementsInto);

    // reset figured bass counter
    fFiguredBassVoicesCounter = 0;
  }
}
```

63.4.4 Populating `msrFiguredBassElement` instances

`msrFiguredBassElement`

In src/formats/msr/mxsr2msrTranslator.cpp, the class `msrFiguredBassElement` instances are populated further and attached to the note by method `mxsr2msrTranslator::handlePendingFiguredBassElements ()`:

```
void mxsr2msrTranslator::handlePendingFiguredBassElements (
  S_msrNote  newNote,
  S_msrVoice voiceToInsertInto)
{
  // ... ... ...

  rational
    newNoteSoundingWholeNotes =
      newNote->
        getNoteSoundingWholeNotes (),
    newNoteDisplayWholeNotes =
      newNote->
        getNoteDisplayWholeNotes ();

  while (fPendingFiguredBassElementsList.size ()) { // recompute at each iteration
    S_msrFiguredBassElement
      figuredBassElement =
        fPendingFiguredBassElementsList.front ();

    /*
      Figured bass elements take their position from the first
      regular note (not a grace note or chord note) that follows
      in score order. The optional duration element is used to
      indicate changes of figures under a note.
    */

    // set the figured bass element's sounding whole notes
    figuredBassElement->
      setMeasureElementSoundingWholeNotes (
        newNoteSoundingWholeNotes,
        "handlePendingFiguredBassElements()");

    // set the figured bass element's display whole notes JMI useless???
    figuredBassElement->
      setFiguredBassDisplayWholeNotes (
        newNoteDisplayWholeNotes);

    // set the figured bass element's tuplet factor
    figuredBassElement->
      setFiguredBassTupletFactor (
        msrTupletFactor (
          fCurrentNoteActualNotes,
          fCurrentNoteNormalNotes));

    // append the figured bass to newNote
    newNote->
      appendFiguredBassElementToNoteFiguredBassElementsList (
        figuredBassElement);

/* JMI
```

```
    // get the figured bass voice for the current voice
    S_msrVoice
      voiceFiguredBassVoice =
        voiceToInsertInto->
          getRegularVoiceFiguredBassVoiceForwardLink ();

    // sanity check
    mfAssert (
      __FILE__, __LINE__,
      voiceFiguredBassVoice != nullptr,
      "voiceFiguredBassVoice is null");

    // set the figuredBassElement's voice upLink
    // only now that we know which figured bass voice will contain it
    figuredBassElement->
      setFiguredBassVoiceUpLink (
        voiceFiguredBassVoice);

    // append the figured bass to the figured bass voice for the current voice
    voiceFiguredBassVoice->
      appendFiguredBassElementToVoice (
        figuredBassElement);
*/

    // don't append the figured bass to the part figured bass voice
    // before the note itself has been appended to the voice

    // remove the figured bass from the list
    fPendingFiguredBassElementsList.pop_front ();
  } // while
}
```

```
%void mxsr2msrTranslator::copyNoteHarmoniesToChord (
%  S_msrNote note, S_msrChord chord)
%{
%  // copy note's harmony if any from the first note to chord
%
%  const list<S_msrHarmony>&
%    noteHarmoniesList =
%      note->getNoteHarmoniesList ();
%
%  if (noteHarmoniesList.size ()) {
%    list<S_msrHarmony>::const_iterator i;
%    for (i=noteHarmoniesList.begin (); i!=noteHarmoniesList.end (); ++i) {
%      S_msrHarmony harmony = (*i);
%
%#ifdef TRACING_IS_ENABLED
%      if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
%        gLogStream <<
%          "Copying harmony '" <<
%          harmony->asString () <<
%          "' from note " << note->asString () <<
%          " to chord '" << chord->asString () <<
%          "'" <<
%          endl;
%      }
%#endif
%
%      chord->
%        appendHarmonyToChord (harmony);
%
%    } // for
%  }
%}
%
```

63.4.5 Inserting S_msrFiguredBassElement 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:

```
void msrVoice::appendNoteToVoice (S_msrNote note)
{
  // ... ... ...

  // are there figured bass elements attached to this note?
  const list<S_msrFiguredBassElement>&
    noteFiguredBassElementsList =
      note->
        getNoteFiguredBassElementsList ();

  if (noteFiguredBassElementsList.size ()) {
    // get the current part's figured bass voice
    S_msrVoice
      partFiguredBassVoice =
        part->
          getPartFiguredBassVoice ();

    for (S_msrFiguredBassElement figuredBassElement : noteFiguredBassElementsList) {
      // append the figured bass element to the part figured bass voice
      partFiguredBassVoice->
        appendFiguredBassElementToVoice (
          figuredBassElement);
    } // for
  }
};
```

63.5 Translating figured bass elements from MSR to MSR

This is done in src/passes/msr2msr/.

In src/passes/msr2msr/msr2msrTranslator.cpp, a newborn clone of the figured bass element is created upon the first visit, stored in msr2msrTranslatorfCurrentFiguredBassElementClone, 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:

```
void msr2msrTranslator::visitStart (S_msrFiguredBassElement& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrFiguredBassElement '" <<
      elt->asString () <<
      "'" <<
      ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  // create a figured bass element new born clone
  fCurrentFiguredBassElementClone =
    elt->
      createFiguredBassElementNewbornClone (
        fCurrentVoiceClone);

  if (fOnGoingNonGraceNote) {
    // append the figured bass to the current non-grace note clone
```

```
    fCurrentNonGraceNoteClone->
      appendFiguredBassElementToNoteFiguredBassElementsList (
        fCurrentFiguredBassElementClone);

    // don't append the figured bass to the part figured bass,  JMI ???
    // this will be done below
  }

  /* JMI
  else if (fOnGoingChord) {
    // register the figured bass in the current chord clone
    fCurrentChordClone->
      setChordFiguredBass (fCurrentFiguredBassElementClone); // JMI
  }
  */

  else if (fOnGoingFiguredBassVoice) { // JMI
    /*
    // register the figured bass in the part clone figured bass
    fCurrentPartClone->
      appendFiguredBassElementToPartClone (
        fCurrentVoiceClone,
        fCurrentFiguredBassElementClone);
        */
    // append the figured bass to the current voice clone
    fCurrentVoiceClone->
      appendFiguredBassElementToVoiceClone (
        fCurrentFiguredBassElementClone);
  }

  else {
    stringstream s;

    s <<
      "figured bass is out of context, cannot be handled:'" <<
      elt->asShortString () <<
      "'";

    msrInternalError (
      gGlobalServiceRunData->getInputSourceName (),
      elt->getInputLineNumber (),
      __FILE__, __LINE__,
      s.str ());
  }
}
```

There are only fields updates upon the second visit:

```
void msr2msrTranslator::visitEnd (S_msrFiguredBassElement& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> End visiting msrFiguredBassElement '" <<
      elt->asString () <<
      "'" <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  fCurrentFiguredBassElementClone = nullptr;
}
```

63.6 Translating figured bass elements 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 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:

```
void msr2lpsrTranslator::visitStart (S_msrFiguredBassElement& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrFiguredBassElement '" <<
      elt->asString () <<
      "'" <<
      ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  // create a figured bass new born clone
  fCurrentFiguredBassElementClone =
    elt->
      createFiguredBassElementNewbornClone (
        fCurrentVoiceClone);

  if (fOnGoingNonGraceNote) {
    // append the figured bass to the current non-grace note clone
    fCurrentNonGraceNoteClone->
      appendFiguredBassElementToNoteFiguredBassElementsList (
        fCurrentFiguredBassElementClone);

    // don't append the figured bass to the part figured bass,  JMI ???
    // this will be done below
  }

  /* JMI
  else if (fOnGoingChord) {
    // register the figured bass in the current chord clone
    fCurrentChordClone->
      setChordFiguredBass (fCurrentFiguredBassElementClone); // JMI
  }
  */

  else if (fOnGoingFiguredBassVoice) { // JMI
    /*
    // register the figured bass in the part clone figured bass
    fCurrentPartClone->
      appendFiguredBassElementToPartClone (
        fCurrentVoiceClone,
        fCurrentFiguredBassElementClone);
        */
    // append the figured bass to the current voice clone
    fCurrentVoiceClone->
      appendFiguredBassElementToVoiceClone (
        fCurrentFiguredBassElementClone);
  }

  else {
    stringstream s;

    s <<
      "figured bass is out of context, cannot be handled:'" <<
      elt->asShortString () <<
```

```
      "'";

    msrInternalError (
      gGlobalServiceRunData->getInputSourceName (),
      elt->getInputLineNumber (),
      __FILE__, __LINE__,
      s.str ());
  }
}
```

Here too, there are only fields updates upon the second visit of `S_msrFiguredBassElement` instances:

```
void msr2lpsrTranslator::visitEnd (S_msrFiguredBassElement& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> End visiting msrFiguredBassElement '" <<
      elt->asString () <<
      "'" <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  fCurrentFiguredBassElementClone = nullptr;
}
```

63.7 Translating figured bass elements from LPSR to LilyPond

This is done in src/passes/lpsr2lilypond/.

There is only one visit of class `msrFiguredBassElement` 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:

```
void msr2lpsrTranslator::visitStart (S_msrFiguredBassElement& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrFiguredBassElement '" <<
      elt->asString () <<
      "'" <<
      ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  // create a figured bass new born clone
  fCurrentFiguredBassElementClone =
    elt->
      createFiguredBassElementNewbornClone (
        fCurrentVoiceClone);

  if (fOnGoingNonGraceNote) {
    // append the figured bass to the current non-grace note clone
    fCurrentNonGraceNoteClone->
      appendFiguredBassElementToNoteFiguredBassElementsList (
        fCurrentFiguredBassElementClone);
```

```
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 (fCurrentFiguredBassElementClone); // JMI
36    }
37    */
38
39    else if (fOnGoingFiguredBassVoice) { // JMI
40      /*
41      // register the figured bass in the part clone figured bass
42      fCurrentPartClone->
43        appendFiguredBassElementToPartClone (
44          fCurrentVoiceClone,
45          fCurrentFiguredBassElementClone);
46          */
47      // append the figured bass to the current voice clone
48      fCurrentVoiceClone->
49        appendFiguredBassElementToVoiceClone (
50          fCurrentFiguredBassElementClone);
51    }
52
53    else {
54      stringstream s;
55
56      s <<
57        "figured bass is out of context, cannot be handled:'" <<
58        elt->asShortString () <<
59        "'";
60
61      msrInternalError (
62        gGlobalServiceRunData->getInputSourceName (),
63        elt->getInputLineNumber (),
64        __FILE__, __LINE__,
65        s.str ());
66    }
67  }
```

Chapter 64

Lyrics handling

Lyrics are presented at section 19.51, [Lyrics], page 185.

64.1 Translating from MXSR to MSR (src/passes/mxsr2msr/)

64.2 Translating from MXSR to MSR (src/passes/mxsr2msr/)

64.3 Translating from MSR to MSR (src/passes/msr2msr/)

64.4 Translating from MSR to LPSR (src/passes/msr2lpsr/)

64.5 Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)

Chapter 65

Full measure rests handling

Full measure rests are presented at section 19.34, [Full measure rests], page 178.

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 19.52, [MIDI], page 186.

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 XII

Music Scores Description Language (MSDL)

Chapter 67

MSDL

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.

67.1 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:

67.2 MSDL basic types

Some types used thoughout MSR are defined in src/formats/msdl/msdlBasicTypes.h/.cpp:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msdl >  egrep -rIn  '^// '
    msdlBasicTypes.h
msdlBasicTypes.h:28:// user languages
msdlBasicTypes.h:52:// comments types
msdlBasicTypes.h:74:// initialization
```

67.3 What the MSDL converter does

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl > msdl -about
What msdlConverter does:

    This multi-pass converter performs various passes depending on the output generated,
    which should be specified a '-lilypond', '-braille', '-musicxml' or '-guido' option.

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

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

The output format is selected via options.
```

67.3.1 LilyPond generation

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl > msdl -lilypond -about
What msdlConverter does:

    This multi-pass converter basically performs 3 passes when generating LilyPond output
    output:

        Pass 1:  converts the MSDL input into a first MSR
        Pass 2:  converts the first MSR into a second MSR;
        Pass 3:  converts the second MSR into a
                 LilyPond Score Representation (LPSR);
        Pass 4:  converts the LPSR to LilyPond code
                 and writes it to standard output.

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

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

67.3.2 Braille generation

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl > msdl -braille -about
What msdlConverter does:

    This multi-pass converter basically performs 4 passes when generating braille output
    output:

        Pass 1:  converts the MSDL input into a first MSR
        Pass 2:  converts the first MSR into a second MSR;
        Pass 3a: converts the second MSR into a
                 Braille Score Representation (BSR)
                 containing one Braille page per MusicXML page;
        Pass 3b: converts the BSRinto another BSR
                 with as many Braille pages as needed
                 to fit the line and page lengthes;
        Pass 4:  converts the BSR to Braille text
                 and writes it to standard output.)

    In this preliminary version, pass 2b merely clones the BSR it receives.

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

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

67.3.3 MusicXML generation

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl > msdl -musicxml -about
What msdlConverter does:

    This multi-pass converter basically performs 4 passes when generating MusicXML output
    output:

        Pass 1:  converts the MSDL input into a first MSR
        Pass 2:  converts the first MSR into a second MSR;
        Pass 3:  converts the second MSR into an MusicXML tree;
        Pass 4:  converts the MusicXML tree to MusicXML code
                 and writes it to standard output.

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

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

67.3.4 Guido generation

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl > msdl -guido -about
What msdlConverter does:

    This multi-pass converter basically performs 4 passes when generating Guido output
    output:

        Pass 1:  converts the MSDL input into a first MSR
        Pass 2:  converts the first MSR into a second MSR;
        Pass 3:  converts the second MSR into an MusicXML tree;
        Pass 4:  converts the MusicXML tree to Guido code
                 and writes it to standard output.

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

    The activity log and warning/error messages go to standard error.
jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl >
```

67.4 A first example

`HelloWorld.msdl` is a minimal example:

```
%{
  The unavoidable HelloWorld score
%}


% the language used for the pitches
% ------------------------------------------------

pitches english            % default is english


% is there an anacrusis?
% ------------------------------------------------

anacrusis                  % measure numbers start at 0


% the structure
```

```
% ------------------------------------------------

% score helloWorld = "Hello World in MSPL" {

  music {
    |1 c2.. d''8
    |||                    % final bar
  }

% } % helloWorld
```

67.5 First example output from the MSDL converter

Compiling `HelloWorld.msdl` to LilyPond, we get the output below.

67.5.1 LilyPond output

```
\version "2.22.0"

% Comment or adapt next line as needed (default is 20)
#(set-global-staff-size 20 )

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

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

\header {
  title                 = ""
  encodingDate          = "Sunday 2021-05-30 @ 12:11:50 CEST"
  software              = "MSDL converter 1.0"
}

\paper {
}

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

Part_Part_One_Staff_One_Voice_One = \absolute {
  \language "nederlands"
c2.. d''8 }

\book {
  \score {
    <<

      \new Staff = "Part_Part_One_Staff_One"
      \with {
      }
      <<
```

```
        \context Voice = "Part_Part_One_Staff_One_Voice_One" <<
          \Part_Part_One_Staff_One_Voice_One
        >>
      >>

    >>

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

    \midi {
      \tempo 4 = 90
    }
  }

}
```

67.5.2 Braille output

With:

```
msdl -braille HelloWorld.msdl -use-encoding-in-file-name -braille-output-kind utf8d
```

we get in file `HelloWorld.msdl_UTF8Debug.brf` Braille 6-dots cells, which can be displayed in a suitable editor as:

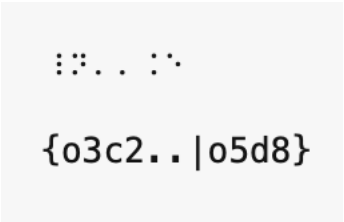


Figure 67.1: Braille for HelloWorld.xml with interpretation

The interpretation shows a textual view of the contents of the previous line. `o*` indicates the octave number.

67.5.3 MusicXML output

Compiling `HelloWorld.msdl` to MusicXML, we get:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
      "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.1">
    <!-- ===== Created by msdl 0.02 on Sunday 2021-05-30 @ 12:15:44 CEST from HelloWorld.
    msdl ===== -->
    <work>
        <work-number/>
        <work-title/>
```

```
    </work>
    <movement-number/>
    <movement-title/>
    <identification>
        <encoding>
            <software>msdl 0.02, https://github.com/jacques-menu/musicformats</software>
            <encoding-date>2021-05-30</encoding-date>
        </encoding>
        <miscellaneous>
            <miscellaneous-field name="description"/>
        </miscellaneous>
    </identification>
    <part-list>
        <score-part id="Part_One">
            <part-name/>
            <score-instrument id="Part_OneI1">
                <instrument-name/>
            </score-instrument>
        </score-part>
    </part-list>
    <part id="Part_One">
        <measure number="1">
            <attributes>
                <divisions>2</divisions>
            </attributes>
            <note>
                <pitch>
                    <step>C</step>
                    <octave>3</octave>
                </pitch>
                <duration>7</duration>
                <voice>1</voice>
                <type>half</type>
                <dot/>
                <dot/>
                <staff>1</staff>
            </note>
            <note>
                <pitch>
                    <step>D</step>
                    <octave>5</octave>
                </pitch>
                <duration>1</duration>
                <voice>1</voice>
                <type>eighth</type>
                <staff>1</staff>
            </note>
        </measure>
    </part>
</score-partwise>
```

67.5.4 Guido output

Compiling `HelloWorld.msdl` to Guido, we get:

```
{[ \staff<1> \set<autoHideTiedAccidentals="on"> \title<""> \barFormat<style= "system",
    range="1"> \bar<hidden="true"> \beamsOff c0/2.. \beamsOff d2/8 ]
  }
```

67.6 A more realistic example

Thanks to Jean Abou-Samra for providing `UnPetitAir.msdl`:

```
%{
  An explicit and implicit voices piano score
%}


% l'identification
% ------------------------------------------------

titre       "Un petit air"
compositeur "Jean Abou Samra"


% la langue pour les hauteurs de notes
% ------------------------------------------------

hauteurs francais          % par défaut: english


% la partition
% ------------------------------------------------

musique unPetitAir =
{
  |1  clef treble
      key c
      time 9/8
      r4. a,4-> <e g bf>8~ <e g bf>4.~

  |2  <e g bf>4. r2.

  %  Maintenant, je reviens en arrière pour la voix supérieure.
  |2  fs''16 gs'' fs''8 cs'' ds'' e'' b' d'' a' e'

  %  La voix inférieure s'éteint.
  |3  c''8 gs' d' c' fs' a' b' gs' b
  |4  a'8 e' a g as gs' d'( a ds)
  |5  e8( b g d' a' e'' b'' c''' b''
  |6  e'''4.) % Rien à la fin.

  %  Je décide d'ajouter une tenue de la basse.
  |5  e2.~ e4.

  %  J'ajoute encore une voix. Au passage, je change la métrique.
  |6  time 6/8
  |6  r8 e'( f') e' c'' d''

  %  Et encore un changement de métrique.
  |7  time 4/4
  |7  e''1~

  %  Je finis la phrase.
  |7  e''4 e' d''8 c'' b' a'
  |8  b'1

  %  Je retourne sur mes pas pour introduire l'ostinato.
  |7  r8 e8 f e f e c' a
  |8  r8 e8 f e c' a e f
  |9  r8 ds e ds e ds b fs

  % etc.
}
```

Jean also provided the output created by hand with LilyPond, see figure 67.2, [Un Petit Air, par Jean Abou-Samra], page 341:

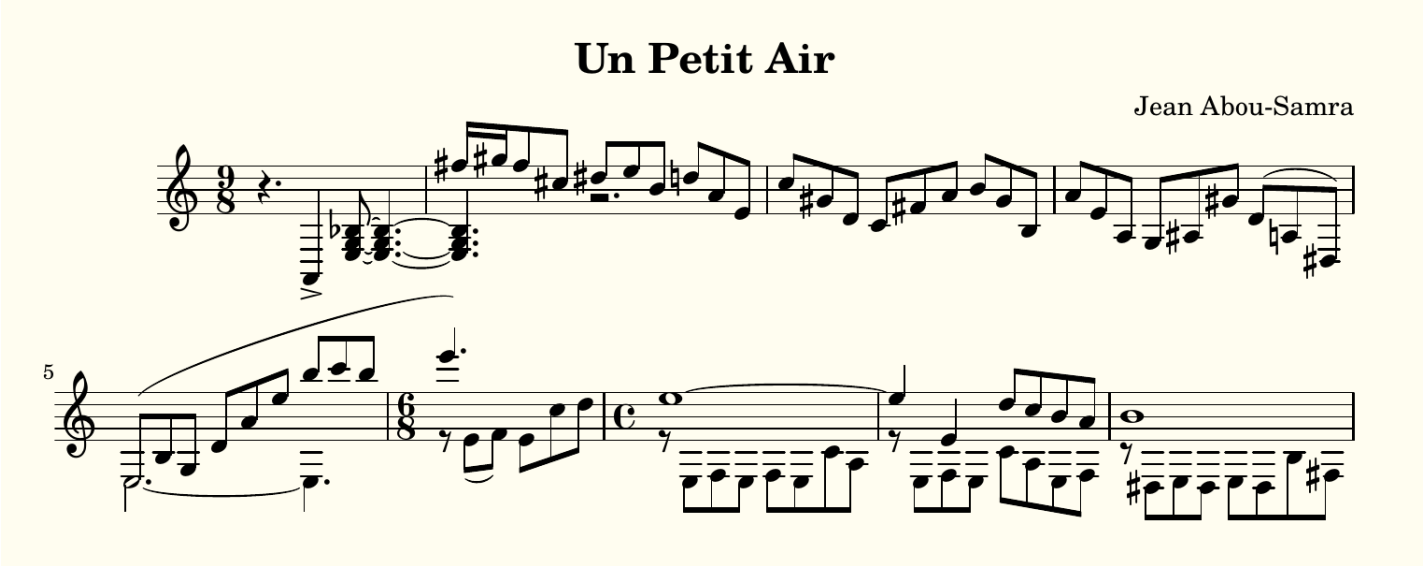


Figure 67.2: Un Petit Air, par Jean Abou-Samra

67.7 Multi-language support

67.7.1 Multi-language messages handling

67.7.2 Multi-language keywords handling

67.8 Lexical analysis

67.9 Music Scores Descriptions Representation (MSDR)

67.10 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.

67.10.1 Error recovery

The MSDL 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 wether to consume a token or not. The corresponding term is it shift when building the analysis tables in LR technology.

Part XIII

Debugging

Chapter 68

Debugging

Debugging MusicFormats can be quite time-consuming. The trace options available have designed to provide fine-grained tracing information to help locate issues.

function `catchSignals ()` in `main ()` functions

File src/wae/enableAbortToDebugErrors.h contains:

```
/*
  MusicFormats Library
  Copyright (C) Jacques Menu 2016-2022

  This Source Code Form is subject to the terms of the Mozilla Public
  License, v. 2.0. If a copy of the MPL was not distributed with this
  file, You can obtain one at http://mozilla.org/MPL/2.0/.

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

#ifndef ___enableAbortToDebugErrorsIfDesired___
#define ___enableAbortToDebugErrorsIfDesired___


// comment the following definition if abort on internal errors is desired
// CAUTION: DON'T USE THIS IN PRODUCTION CODE,
// since that could kill a session on a \Web\ server, for example

#ifndef ABORT_TO_DEBUG_ERRORS
  #define ABORT_TO_DEBUG_ERRORS
#endif


#endif
```

68.1 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

Part XIV

Indexes

Files index

Symbols

.cpp 74
.h 44
<regex> 23
*.cpp 41
Interface. 35
1.0.0 31
CommonLaTeXFiles/ 19
Interface 32
IntroductionToMusicxml/ 19
LilyPondIssue34 ..... 29, 221
Makefile 24, 26, 31
Mikrokosmos3Wandering 29, 105
MusicFormatsLaTeXSettings.tex 27
MusicFormatsVersionDate.h 20
MusicFormatsVersionDate.txt 20
MusicFormatsVersionNumber.h 20
MusicFormatsVersionNumber.txt 20
oah 66
braille 66
brailleGeneration 30
bsr 30, 66
bsr2braille 30
bsr2bsr 30
build 26
build/bin 24
clean 31
clisamples 28
common 27
components 29
converters 28
countnotes 26
doc 26
documentation/ 19
elements.bash 27
files 26, 31
formats 30
formatsgeneration 30
generators 29
graphics/ 19
guidoGeneration 30
javascript 26
libmusicxml 26
libmusicxml/build/bin 26
libmusicxml/src 198
lilypond 31, 66
lilypondGeneration 30
lpsr 31, 66
lpsr2lilypond 30
mfutilities 29
msdl 31, 66
msdl2braille 28
msdl2guido 28
msdl2lilypond 28
msdl2musicxml 28
msdlconverter 28
msdr 31
msr 31, 66
msr2braille 28
msr2bsr 30
msr2guido 28
msr2lilypond 28
msr2lpsr 30
msr2msr 30
msr2musicxml 28
msr2mxsr 30
msr2mxsrOah 132
msrGeneration 30
msrapi 31
multiGeneration 30
musicformatsversion.txt 31
musicxml2braille 28, 100
musicxml2guido 28
musicxml2lilypond 28
musicxml2musicxml 28
mxsr 31, 66, 218
mxsr2guido 30
mxsr2msr 30
mxsr2msrOah 132
mxsr2musicxml 30
mxsrGeneration 30
oah 29
ostream 212
packages 26
passes 28, 30
presentation 27
presentation/ 19
samples 26
schemas 26
src 26, 28
validation 26
wae 31
win32 26
xml2brl 66, 113
xml2guido 26
xml2lbr 113
xml2ly 43, 66, 113
xml2xml 113

musicxml2ly 221

B

build
CMakeLists.txt 69
CMakesList.txt 23
lib 20
Makefile 24

C

clisamples 121
displayMusicformatsHistory.cpp 103
displayMusicformatsVersion.cpp 103
libMultipleInitsTest.cpp 121
LilyPondIssue34.cpp 37, 121
Makefile 24
Mikrokosmos3Wandering.cpp 17, 36, 121, 126
msdl.cpp 121, 126
xml2Any.cpp 121
xml2ly.cpp 39
tt *Component.h/.cpp 99

D

documentation
MusicFormatsAPIGuide/MusicFormatsAPIGuide.pdf 22
MusicFormatsLaTeXSettings.tex 20
MusicFormatsMaintainanceGuide/MusicFormatsMaintainanceGuide.pdf 22
MusicFormatsUserGuide/MusicFormatsUserGuide.pdf 22

I

InsiderHandler.cpp 44

L

libmusicxml/samples
. 78
countnotes.cpp 57
elements/factory.cpp 201
libmusicxml/src/files
xmlfile.h 202
xmlreader.h/.cpp 203

M

main () 28, 41, 120, 228

S

src
..... 69
clisamples 41
formats 34
passes 35
representations 34
src/clisamples 121
displayMusicformatsHistory.cpp 103
displayMusicformatsVersion.cpp 103
libMultipleInitsTest.cpp 121
LilyPondIssue34.cpp 37, 121
Makefile 24
Mikrokosmos3Wandering.cpp 17, 36, 121, 126
msdl.cpp 121, 126
xml2Any.cpp 121
xml2ly.cpp 39
src/components
. 94
mfcComponents.h 33
src/formats/bsr
bsr.cpp 80
bsrBasicTypes.h 195
bsrBasicTypes.h/.cpp 195
src/formats/lpsr
lpsr2lilypondTranslator.cpp 116
lpsrBasicTypes.h/.cpp 193
lpsrElements.h/.cpp 66
lpsrOah.cpp 139
lpsrScores.cpp 108
lpsrScores.h/.cpp 39
src/formats/msdl
msdlBasicTypes.h/.cpp 334
msdlScanner.cpp 146
msdlScanner.h 146
msdlTokens.cpp 84
src/formats/msr 174
msr.h/.cpp 234
msr2msrTranslator.h 147
msrBarLines.h 68, 280
msrBarlines.h 68
msrBasicTypes.cpp 70, 84
msrBasicTypes.h ... 83, 136, 158, 160, 161, 173, 181
msrBasicTypes.h/.cpp ... 88, 136, 152, 183, 234
msrBeatRepeats.h/.cpp 176
msrChords.h/.cpp 255
msrElements.h 56
msrFullMeasureRests.h/.cpp 178
msrHistory.h/.cpp 99
msrIdentification.h 153
msrKeys.h/.cpp 87
msrMeasureElement.h/.cpp 253, 270
msrMeasureRepeat.h/.cpp 177
msrMeasureRepeats.h/.cpp 91
msrMeasures.h/.cpp 244, 255
msrMeasuresSlice.h 155
msrMeasuresSlices.h/.cpp 187
msrMeausre.h/.cpp 174
msrNotes.cpp 60, 309, 325
msrOah.cpp 112
msrOah.h 112
msrParts.h 301, 317
msrParts.h/.cpp 255--257, 288
msrPathToVoice.h.h/.cpp 192
msrRepeats.h/.cpp 179
msrSegment.cpp 88
msrSegments.cpp 275
msrSegments.h/.cpp 174, 246, 255
msrStaves.h/.cpp 255--257
msrTempos.h 162
msrTempos.h/.cpp 86, 154
msrTies.h 87
msrTimeSignature.cpp 145

msrTimeSignature.h 145
msrTuplets.h/.cpp 255
msrVoiceElements.h/.cpp 172
msrVoices.cpp 272--274
msrVoices.h 91, 151, 152, 173
msrVoices.h/.cpp 145, 248, 255--257
mxsr2msrTranslator.cpp 307, 323
src/formats/mxsr
mxsr.h/.cpp 198, 203, 217
mxsrOah.h/.cpp 132
src/mflibrary
mfMusicformatsError.h 121
mfServiceRunData.h/.cpp 38, 233
src/mfutilities 66
mfcBasicTypes.h 134
mfEnumAll.h 71, 83, 84
mfIndentedTextOutput.h 47, 48
mfStringsHandling.cpp 74
src/oah
tracingOah.h/.cpp 66
enableHarmoniesExtraOahIfDesired.h 140
enableTracingIfDesired.h 43
generalOah.h/.cpp 140
harmoniesExtraOah.h/.cpp 139, 140
mfIndentedTextOutput.cpp 74
musicxmlOah.h/.cpp 66
oahAtomCollection.h/.cpp 134
oahAtomsCollection 113
oahAtomsCollection.cpp 115, 130, 137
oahAtomsCollection.h 136
oahAtomsCollection.h/.cpp ..106, 122, 124, 131, 238
oahBasicTypes.cpp 122, 128
oahBasicTypes.h 121
oahBasicTypes.h/.cpp ..113, 114, 117, 118, 120--122, 128
oahEarlyOptions.cpp 74, 131
oahEarlyOptions.h 130
oahElements.cpp 133
oahElements.h 127
oahElements.h/.cpp 122
oahOah.cpp 135
oahOah.h/.cpp 132
tracingOah.h/.cpp 43
src/utilities
mfBool.h/.cpp 83
mfcLibraryComponent.h/.cpp 101
mfIndentedTextOutput.cpp.h/.cpp 148
mfIndentedTextOutput.h/.cpp 46, 66
mfTiming.h/.cpp 52
src/wae
enableAbortToDebugErrors.h 42, 343
wae.h.h/.cpp 76
waeExceptions 42
waeExceptions.h/.cpp 76
waeHandlers.cpp 73
waeHandlers.h 73

Types index

Symbols

, ... 200
: 201
Bool 83
Enum* 85
EnumTrueHarmonies 85
Mikrokosmos3WanderingInsiderHandler 105
SMARTP 86
SXMLFile 203
S_FiguredBassElement 319
S_figured_bass 320, 321
S_harmony 303--305
S_msrBarLine 144
S_msrFiguredBassElement 325, 328
S_msrHarmony 314
S_msrVoice 144
TXMLFile 203
..... 136, 200, 234
bsr2brailleTranslator 210
bsrBrailleGenerator 211
bsrCellKind 195, 196
c 88
cubase 113
lpsr2lilypondTranslator 59, 60, 208
lpsrComment 108
lpsrPitchesLanguageAtom 113
lpsrScore 39, 108, 193, 209
mfEnumAll 71
mfMultiGenerationOutputKindAtom 119
mfOptionsAndArguments 126
mfServiceRunData 37
mfTimingItemsList 52
mfcComponenKind 96
mfcComponentDescr 96
mfcConverterComponent 97
mfcGeneratorComponent 97
mfcLibraryComponent 97
mfcMultiComponent 97, 103, 134
mfcMultiComponentEntropicityKind 97
mfcMultiComponentUsedFromTheCLIKind 98
mfcPassComponent 97
mfcRepresentationComponent 97
mfcVersionDescr 95
mfcVersionNumber 94
mfcVersionsHistory 95
msr*Element 169
msr2mxsrTranslator 62, 216
msrAccordionRegistration 169
msrBarCheck 169
msrBarLine 169, 280, 282
msrBarNumberCheck 169
msrBeatRepeat 176
msrBeatRepeatPattern 176
msrBeatRepeatReplicas 176
msrBook 170, 192
msrChord 143, 167, 174, 300, 316
msrClef 165, 169
msrClefKind 165
msrCoda 170
msrDalSegno 170
msrDamp 169
msrDampAll 169
msrDottedDuration 162
msrDurationKind 160, 161
msrElement 169
msrEyeGlasses 170
msrFiguredBassElement 184, 319, 321--323, 328
msrFullMeasureRests 59, 178
msrFullMeasureRestsContents 178
msrGraceNotesGroup 169
msrHarmony 184, 304--309, 314
msrHarmonyKind 83
msrHarpPedalsTuning 169
msrHiddenMeasureAndBarLine 169
msrHumdrumScotKeyItem 166
msrIdentification 153
msrKey 87, 166, 169
msrLength 136, 137
msrLineBreak 169
msrMeasure 59, 151, 250, 251
msrMeasureElement 156, 253, 270
msrMeasureElements 174, 244
msrMeasureRepeat 177
msrMeasureRepeatPattern 177
msrMeasureRepeatReplicas 177
msrMeasuresSlice 155, 188
msrMeasuresSlicesSequence 155, 188
msrMeasusre 174
msrMeausre 174
msrMoment 154
msrNote 64, 156, 181, 184, 186, 300, 303, 306, 316, 319, 322
msrNoteEvent 155
msrNoteEventKind 155, 187
msrNoteKind 181
msrOctaveEntryVariable 241, 242
msrOctaveShift 169
msrPageBreak 169

msrPart 169, 184, 257, 300, 301, 316, 317
msrPartGroup 171
msrPartGroupElement 169
msrPathToVoice 192
msrPedal 169
msrPrintLayout 170
msrRehearsalMark 170
msrRepeat 173, 174, 179
msrRepeatCommonPart 179
msrRepeatEnding 179, 180
msrRepeatEndingKind 180
msrScaling 57
msrScordatura 169
msrScore 59, 110, 170, 192
msrSegment 169, 173, 178, 243, 272
msrSegno 170
msrSlur 151
msrSpannerTypeKind 156
msrStaff 167, 301, 317
msrStaffDetails 169
msrStaffLevelElement 170
msrStanza 186
msrSyllable 169, 170, 185, 186
msrSyllableKind 185
msrTempo 86, 154, 164, 169
msrTempoKind 162
msrTempoNote 162
msrTempoTuplet 163
msrTime 169
msrTimeSignature 75, 167
msrTimeSignatureSymbolKind 166
msrTranspose 170
msrTuplet 63, 169, 300, 316
msrTupletElement 170
msrTupletFactor 183
msrTupletInKind 182
msrVoice 145, 173, 186, 243, 257, 301, 317
msrVoiceCreateInitialLastSegmentKind 173
msrVoiceElement .169, 172, 173, 179, 180, 243
msrVoiceKind 173
msrVoiceStaffChange 169
mxmlelement 198
mxsr2msrSkeletonBuilder 35
mxsr2msrTranslator 206
oahAtom 112, 113
oahAtomExpectingAValue 113, 114
oahAtomImplicitlyStoringAValue 119
oahAtomStoringAValue 113--116
oahBooleanAtom 113, 115, 132
oahCombinedBooleansAtom 113
oahContactAtom 113
oahElement 112
oahElementHelpOnlyKind 127
oahElementUse 120
oahFloatAtom 113
oahGroup 112, 113
oahHandler 103, 104, 112
oahHistoryAtom 134
oahIntegerAtom 113
oahLengthAtom 113, 136
oahLengthUnitKindAtom 115
oahMacroAtom 238
oahMultiplexBooleansAtom 113
oahOahGroup 132
oahOptionsVector 113, 119
oahPureHelpAtomExpectingAValue 118
oahPureHelpAtomWithoutAValue 117
oahRGBColorAtom 115
oahRationalAtom 113
oahSubGroup 112
oahThreeBooleansAtom 113
oahVersionAtom 106, 134
s 70, 73, 79, 81, 86, 280
xml2lyInsiderHandler 235
xmlattribute 199
xmlelement 199, 200

S

SXMLFile 203

Methods and fields index

Symbols

*DeepClone () 145
*IsNeeded 209
*NewbornClone () 144
fBooleanVariable 115
fInsiderHandler 125
fReverseNamesDisplayOrder 132
fSetByUser 115
fVersionsList 95
msr2msrTranslator 310, 325
mxsr2msrTranslator 305, 306, 321, 322
RegularHandler
  createOahRegularGroup () 134
acceptIn () 74
acceptOut () 74
applyAtomWithDefaultValue () 120
applyAtomWithValue () 120
applyElement () 120
asString() () 89
asStringShort() () 89
asString () 49, 74, 157
browseData () 59, 74
browsedata () 55, 59
compare* () 92
countnotes
  visitStart 57
create* () 74, 75, 86
createGlobalMxsr2msrOahGroup () 44
finalize*() () 88
finalize* () 255
generateCodeForBrailleCell () 212
get*() () 87
get* () 74
getSetByUser () 116
initialize*() () 88
initializeHandlerMultiComponent () .. 104, 105
lpsr2lilypondTranslator
  generateLilypondVersion () 116
lpsrOahGroup
  initializeLpsrPaperOptions () 139
lpsrScore
  lpsrScore () 108
mfIndentedStreamBuf
  sync () 46
mfMultiGenerationOutputKindAtom
  mfMultiGenerationOutputKindAtom () 119
mfcComponentDescr
  printOwnHistory () 97
mfcMultiComponent
  printHistory () 98
  print () 98
msr2bsrTranslator
  finalizeCurrentMeasureClone () 255
msr2lpsrTranslator
  visitStart (S_msrMeasure& elt) 248
msr2msrTranslator
  visitStart (S_msrMeasure& elt) 248
msr2mxsrTranslator
  translateMsrToMxsr () 216
msrChord
  finalizeChord () 255
msrClef
  getClefStaffNumber () 216
msrFullMeasureRests
  appendMeasureCloneToFullMeasureRests () 248
msrHarmony
  createWithoutVoiceUplink () 305
msrMeasureElement
  fMeasureElementPositionInMeasure . 253, 270
msrMeasureRepeat
  displayMeasureRepeat () 91
msrMeasuresSlicesSequence
  identifySoloNotesAndRests () 191
msrMeasure
  appendElementToMeasure () 244
  finalizeFiguredBassMeasure () 255
  finalizeHarmoniesMeasure () 255
  finalizeMeasureClone () 255
  finalizeMeasure () 254, 255, 261
  finalizeRegularMeasure () 255
msrNote
  appendHarmonyToNoteHarmoniesList () .. 307, 308
msrPart
  appendStaffDetailsToPart () 64
  browseData () 288
  collectPartMeasuresSlices () 189
  createAMeasureAndAppendItToPart () 251
  createPartFiguredBassVoice () 318
  finalizeLastAppendedMeasureInPart () .. 257
  finalizePartAndAllItsMeasures () 257
  finalizePartClone () 255
  finalizePart () 255, 258
  finalizeRepeatEndInPart () 256
msrSegment
  appendMeasureToSegment () 246, 248
  assertSegmentMeasuresListIsNotEmpty () 275

createAMeasureAndAppendItToSegment () 174, 248
createSegmentDeepClone () 272
createSegmentNewbornClone () 272
create () 272
finalizeAllTheMeasuresOfSegment () 255
msrStaff
collectStaffMeasuresSlices () 190, 191
finalizeLastAppendedMeasureInStaff () .257
finalizeRepeatEndInStaff () 256
finalizeStaff () 255, 258, 259
msrTempo
createTempoPerMinute () 80
fTempoBeatUnit 165
fTempoEquivalentBeatUnit 165
fTempoKind 165
fTempoNotesRelationshipKind 165
fTempoNotesRelationshipLeftElements ...165
fTempoNotesRelationshipRightElements ..165
fTempoParenthesizedKind 165
fTempoPerMinute 165
fTempoPlacementKind 165
fTempoWordsList 165
kTempoNotesRelationshipEquals 165
msrTemp
tempoKindAsString () 80
msrTie
create () 80
msrTuplets
finalizeTuplet () 255
msrVoice
addGraceNotesGroupBeforeAheadOfVoiceIfNeeded () 274
appendBarLineToVoice () 274
appendCodaToVoice () 274
appendDampAllToVoice () 274
appendDampToVoice () 274
appendEyeGlassesToVoice () 274
appendMeasureCloneToVoiceClone () 248
appendNoteToVoice () 309, 325
appendPedalToVoice () 274
appendPendingMeasureRepeatToVoice () ..248
appendSegnoToVoice () 274
appendStaffDetailsToVoice () 274
createAMeasureAndAppendItToVoice () ..248, 273
createFullMeasureRestsInVoice () 248
createMeasureRepeatFromItsFirstMeasures () 248, 273, 278
createNewLastSegmentForVoice () 273
createNewLastSegmentFromItsFirstMeasureForVoice () 248, 274
fVoiceLastSegment 173
finalizeLastAppendedMeasureInVoice () .257
finalizeRepeatEndInVoice () 256
finalizeVoiceAndAllItsMeasures () 257
finalizeVoice () 255, 259
handleFullMeasureRestsStartInVoiceClone () 274
handleHooklessRepeatEndingEndInVoice () 274
handleVoiceLevelContainingRepeatEndWithoutStart () 274
handleVoiceLevelRepeatEndWithStart () .274
handleVoiceLevelRepeatEndWithoutStart () 274
handleVoiceLevelRepeatEndingStartWithExplicitStart 248, 274
handleVoiceLevelRepeatEndingStartWithoutExplicitSta 248, 274
handleVoiceLevelRepeatStart () ...248, 274
initializeVoice () 272
mxsr2msrSkeletonBuilder
fThereAreHarmoniesToBeAttachedToCurrentNote 303
mxsr2msrTranslator
finalizeCurrentChord () 255
finalizeTupletAndPopItFromTupletsStack () 255
handlePendingFiguredBassElements () ...323
populateNote () 306, 322
oahAtomStoringAValue
fSetByUser 115
oahEarlyOptions
applyEarlyOptionIfRelevant () 131
oahElement
fetchNames () 133
oahHandler
handleOptionNameCommon () 120
applyOptionsFromElementUsesList () 128
checkMissingPendingArgvAtomExpectingAValueValue () 122
fHandlerArgumentsVector 120
fHandlerMultiComponent 103--105
fPendingArgvAtomExpectingAValue 120
handleKnownArgvAtom () 120
handleOptionNameCommon () 120
handleOptionsAndArgumentsFromArgcArgv () 120
oahHistoryAtom
printHistory () 135
oahIntegerAtom
setIntegerVariable () 115
oahLengthAtom
applyAtomWithValue () 137
printAtomWithVariableOptionsValues () .139
oahMacroAtom
applyElement () 239
fMacroAtomsList 239
oahOahGroup
initializeOahBasicHelpOptions () .132, 135
printOahOahValues () 133
oahVersionAtom
applyElement () 107, 130
printVersion () 107
printHistory () 97, 107, 134, 135
printSlices () 64
printVersion () 97, 106, 107, 134, 135
print () 49, 74, 157
set*() () 87
set*Variable () 115
set* () 74

translate*() () 90
visit* () 55, 147
visitEnd () 55, 147
visitStart () 55, 57, 147
xml2lyInsiderHandler
createTheXml2lyOptionGroups () 235
xml2lyRegularHandler
createInformationsRegularGroup () 136
xml2xmlInsiderHandler
xml2xmlInsiderHandler () 104
xmlelement
fType 200
() 135

Constants, functions and variables index

Symbols

AllFirst 84
AllLast 84
K_NO_STANZA_NUMBER 80
TrueHarmoniesFirst 85
TrueHarmoniesLast 85
gGlobalOStreamIndenter 80
gGlobalOahOahGroup 132
gGlobalServiceRunData 233
gGlobalTimingItemsList 80
gIndenter 47
i 74
kComponentUsedFromTheCLIYes 98
kPlacementAbove 165
kSlurTypeRegularStart 81
pPrivateThisMethodHasBeenRun 80
*AsString() () 88
catchSignals () 343
convert*() () 90
convertArgcArgvToOptionsAndArguments () . 121, 126
create*OahGroup () 233
create*PassComponent () 99
create*RepresentationComponent () 99
createGlobalHarmoniesExtraOahGroup () 139
createLibraryComponent () 101, 103
createMsrRepresentationComponent () 99
createMusicxml2brailleConverterComponent () 100
createMxmlScorePartWiseElement () 217
fromString () 88
getopt* () 112
initialize* () 234
initializeBSR () 234
initializeLPSR () 234
initializeMSRBasicTypes () 234
initializeMSR () 234
initializeMsrGenerationAPI () 80
main () 343
mfAssert () 70
musicxmlfile2lilypond () 112
printMxsr () 203
translateLpsrToLilypondWithHandler () 47
translateMsrToLpsrScore () 53

A

argc/argv 120, 121

Options

Symbols

--hist, --history 97
--v, --version 97
-auto-output-file-name, -aofn 208, 218
-auto-utf8, -au8d 240
-cpu 52
-display* 150, 343
-display-msr-1, -dmsr1 .....300, 316
-display-msr-1-details, -dmsr1d 300, 316
-display-msr-1-short, -dmsr1s 300, 316
-display-msr-2, -dmsr2 316
-display-msr-2-details, -dmsr2d 300, 316
-display-msr-2-short, -dmsr2s 316
-display-msr-2-short, -msr2s 300
-display-msr-2msr2, -dmsr2 300
-display-msr-skeleton, -dmsrskel 300, 316
-display-options-values 139
-display-options-values, -dov 113
-files 211
-history, -hist 98, 99, 103, 106
-ignore-figured-bass, -ofigbass 319
-ignore-harmonies, -oharms 303
-input-line-numbers, -iln 343
-insider 43, 113, 130
-lilypond-generation-infos, -lpgi 108
-lilypond-version, -lpv 116
-output-file-name, -o 208, 218
-quiet, -q 130
-reverse-names-display-order, -rndo .132, 133
-trace-components, -tcomps 130
-trace-figured-bass, -tfigbass 316
-trace-harmonies, -tharms 300
-trace-oah, -toah 130
-trace-oah-details, -toahd 130
-trace-passes, -tpasses 343
-version, -v 98, 99, 103, 106, 130, 136

MusicXML index

Symbols

!DOCTYPE </> 198, 204
?xml </> 198, 204
backup </> 132
barLine </> 280
barline </> 281
defaults </> 61
direction </> 153
direction-type </> 153
divisions </> 152
forward </> 132
metronome </> 153
millimeter </> 57
part </> 303, 319
print </> 61
repeat </> 81
scaling </> 57, 58
system-layout </> 61
tenth </> 57
words </> 153

Main index

Symbols

. ..... 78, 94
.cpp 74
.h 44
..... 126
<!DOCTYPE /> 198, 204
<?xml /> 198, 204
<backup /> 132
<barLine /> 280
<barline /> 281
<defaults /> 61
<direction /> 153
<direction-type /> 153
<divisions /> 152
<forward /> 132
<metronome /> 153
<millimeter /> 57
<part /> 303, 319
<print /> 61
<regex> 23
<repeat /> 81
<scaling /> 57, 58
<system-layout /> 61
<tenth /> 57
<words /> 153
*.cpp 41
Both 21
Interface. 35
Name 21
*Repr 21
, ... 200
--hist, --history 97
--v, --version 97
-auto-output-file-name, -aofn 208, 218
-auto-utf8, -au8d 240
-cpu 52
-display* 150, 343
-display-msr-1, -dmsr1 ..... 300, 316
-display-msr-1-details, -dmsr1d 300, 316
-display-msr-1-short, -dmsr1s 300, 316
-display-msr-2, -dmsr2 316
-display-msr-2-details, -dmsr2d 300, 316
-display-msr-2-short, -dmsr2s 316
-display-msr-2-short, -msr2s 300
-display-msr-2msr2, -dmsr2 300
-display-msr-skeleton, -dmsrskel 300, 316
-display-options-values 139
-display-options-values, -dov 113
-files 211
-history, -hist 98, 99, 103, 106
-ignore-figured-bass, -ofigbass 319
-ignore-harmonies, -oharms 303
-input-line-numbers, -iln 343
-insider 43, 113, 130
-lilypond-generation-infos, -lpgi 108
-lilypond-version, -lpv 116
-output-file-name, -o 208, 218
-quiet, -q 130
-reverse-names-display-order, -rndo . 132, 133
-trace-components, -tcomps 130
-trace-figured-bass, -tfigbass 316
-trace-harmonies, -tharms 300
-trace-oah, -toah 130
-trace-oah-details, -toahd 130
-trace-passes, -tpasses 343
-version, -v 98, 99, 103, 106, 130, 136
1.0.0 31
2.22.0 116
: 201
AllFirst 84
AllLast 84
Bool 83
CommonLaTeXFiles 19
EXP 78
Enum* 85
EnumTrueHarmonies 85
Interface 32
IntroductionToMusicxml 19
K_NO_STANZA_NUMBER 80
LilyPondIssue34 ..... 29, 221
Makefile 24, 26, 31
Mikrokosmos3Wandering 29, 105
MikrokosmosWanderingInsiderHandler 105
MusicAndHarmonies.cpp 220
MusicFormatsLaTeXSettings.tex 27
MusicFormatsVersionDate.h 20
MusicFormatsVersionDate.txt 20
MusicFormatsVersionNumber.h 20
MusicFormatsVersionNumber.txt 20
SMARTP 86
SXMLFile 203
S_FiguredBassElement 319
S_figured_bass 320, 321
S_harmony 303--305
S_msrBarLine 144
S_msrFiguredBassElement 325, 328
S_msrHarmony 314
S_msrVoice 144

TRACING_IS_ENABLED 43
TXMLFile 203
TrueHarmoniesFirst 85
TrueHarmoniesLast 85
..... 136, 200, 234
oah 66
__declspec(dllexport) 78
bool 83
braille 66
brailleGeneration 30
bsr 30, 66
bsr2braille 30
bsr2brailleTranslator 210
bsr2bsr 30
bsrBrailleGenerator 211
bsrCellKind 195, 196
build 26
build/bin 24
c 88
clean 31
clisamples 28
cmake 23, 68
common 27
components 29
converters 28
countnotes 26
cubase 113
doc 26
documentation 19
dynamic_cast 79
elements.bash 27
fBooleanVariable 115
fHandlerArgumentsVector 120
fHandlerMultiComponent 103--105
fInsiderHandler 125
fMacroAtomsList 239
fMeasureElementPositionInMeasure 253, 270
fPendingArgvAtomExpectingAValue 120
fReverseNamesDisplayOrder 132
fSetByUser 115
fTempoBeatUnit 165
fTempoEquivalentBeatUnit 165
fTempoKind 165
fTempoNotesRelationshipKind 165
fTempoNotesRelationshipLeftElements 165
fTempoNotesRelationshipRightElements 165
fTempoParenthesizedKind 165
fTempoPerMinute 165
fTempoPlacementKind 165
fTempoWordsList 165
fThereAreHarmoniesToBeAttachedToCurrentNote 303
fType 200
fVersionsList 95
fVoiceLastSegment 173
false 115
files 26, 31
formats 30
formatsgeneration 30
gGlobalOStreamIndenter 80
gGlobalOahOahGroup 132
gGlobalServiceRunData 233
gGlobalTimingItemsList 80
gIndenter 47
generators 29
graphics 19
guidoGeneration 30
i 74
javascript 26
kComponentUsedFromTheCLIYes 98
kPlacementAbove 165
kSlurTypeRegularStart 81
kTempoNotesRelationshipEquals 165
libmusicxml 26
libmusicxml/build/bin 26
libmusicxml/src 198
lilypond 31, 66
lilypondGeneration 30
lpsr 31, 66
lpsr2lilypond 30
lpsr2lilypondTranslator 59, 60, 208
lpsrComment 108
lpsrPitchesLanguageAtom 113
lpsrScore 39, 108, 193, 209
make 23, 26, 31, 69
mf 66
mfEnumAll 71
mfMultiGenerationOutputKindAtom 119
mfOptionsAndArguments 126
mfServiceRunData 37
mfTimingItemsList 52
mfcComponenKind 96
mfcComponentDescr 96
mfcConverterComponent 97
mfcGeneratorComponent 97
mfcLibraryComponent 97
mfcMultiComponent 97, 103, 134
mfcMultiComponentEntropicityKind 97
mfcMultiComponentUsedFromTheCLIKind 98
mfcPassComponent 97
mfcRepresentationComponent 97
mfcVersionDescr 95
mfcVersionNumber 94
mfcVersionsHistory 95
mfutilities 29
msdl 31, 66
msdl2braille 28
msdl2guido 28
msdl2lilypond 28
msdl2musicxml 28
msdlconverter 28
msdr 31
msr 31, 66
msr*Element 169
msr2braille 28
msr2bsr 30
msr2guido 28
msr2lilypond 28
msr2lpsr 30
msr2msr 30
msr2msrTranslator 310, 325
msr2musicxml 28

msr2mxsr 30
msr2mxsrOah 132
msr2mxsrTranslator 62, 216
msrAccordionRegistration 169
msrBarCheck 169
msrBarLine 169, 280, 282
msrBarNumberCheck 169
msrBeatRepeat 176
msrBeatRepeatPattern 176
msrBeatRepeatReplicas 176
msrBook 170, 192
msrChord 143, 167, 174, 300, 316
msrClef 165, 169
msrClefKind 165
msrCoda 170
msrDalSegno 170
msrDamp 169
msrDampAll 169
msrDottedDuration 162
msrDurationKind 160, 161
msrElement 169
msrEyeGlasses 170
msrFiguredBassElement 184, 319, 321--323, 328
msrFullMeasureRests 59, 178
msrFullMeasureRestsContents 178
msrGeneration 30
msrGraceNotesGroup 169
msrHarmony 184, 304--309, 314
msrHarmonyKind 83
msrHarpPedalsTuning 169
msrHiddenMeasureAndBarLine 169
msrHumdrumScotKeyItem 166
msrIdentification 153
msrKey 87, 166, 169
msrLength 136, 137
msrLineBreak 169
msrMeasure 59, 151, 250, 251
msrMeasureElement 156, 253, 270
msrMeasureElements 174, 244
msrMeasureRepeat 177
msrMeasureRepeatPattern 177
msrMeasureRepeatReplicas 177
msrMeasuresSlice 155, 188
msrMeasuresSlicesSequence 155, 188
msrMeasusre 174
msrMeausre 174
msrMoment 154
msrNote 64, 156, 181, 184, 186, 300, 303, 306, 316, 319, 322
msrNoteEvent 155
msrNoteEventKind 155, 187
msrNoteKind 181
msrOctaveEntryVariable 241, 242
msrOctaveShift 169
msrPageBreak 169
msrPart 169, 184, 257, 300, 301, 316, 317
msrPartGroup 171
msrPartGroupElement 169
msrPathToVoice 192
msrPedal 169
msrPrintLayout 170
msrRehearsalMark 170
msrRepeat 173, 174, 179
msrRepeatCommonPart 179
msrRepeatEnding 179, 180
msrRepeatEndingKind 180
msrScaling 57
msrScordatura 169
msrScore 59, 110, 170, 192
msrSegment 169, 173, 178, 243, 272
msrSegno 170
msrSlur 151
msrSpannerTypeKind 156
msrStaff 167, 301, 317
msrStaffDetails 169
msrStaffLevelElement 170
msrStanza 186
msrSyllable 169, 170, 185, 186
msrSyllableKind 185
msrTempo 86, 154, 164, 169
msrTempoKind 162
msrTempoNote 162
msrTempoTuplet 163
msrTime 169
msrTimeSignature 75, 167
msrTimeSignatureSymbolKind 166
msrTranspose 170
msrTuplet 63, 169, 300, 316
msrTupletElement 170
msrTupletFactor 183
msrTupletInKind 182
msrVoice ... 145, 173, 186, 243, 257, 301, 317
msrVoiceCreateInitialLastSegmentKind 173
msrVoiceElement .169, 172, 173, 179, 180, 243
msrVoiceKind 173
msrVoiceStaffChange 169
msrapi 31
multiGeneration 30
musicformatsversion.txt 31
musicxml2braille 28, 100
musicxml2guido 28
musicxml2lilypond 28
musicxml2musicxml 28
mxmlelement 198
mxsr 31, 66, 218
mxsr2guido 30
mxsr2msr 30
mxsr2msrOah 132
mxsr2msrSkeletonBuilder 35
mxsr2msrTranslator ... 206, 305, 306, 321, 322
mxsr2musicxml 30
mxsrGeneration 30
oah 29
oahAtom 112, 113
oahAtomExpectingAValue 113, 114
oahAtomImplicitlyStoringAValue 119
oahAtomStoringAValue 113--116
oahBooleanAtom 113, 115, 132
oahCombinedBooleansAtom 113
oahContactAtom 113
oahElement 112
oahElementHelpOnlyKind 127

oahElementUse 120
oahFloatAtom 113
oahGroup 112, 113
oahHandler 103, 104, 112
oahHistoryAtom 134
oahIntegerAtom 113
oahLengthAtom 113, 136
oahLengthUnitKindAtom 115
oahMacroAtom 238
oahMultiplexBooleansAtom 113
oahOahGroup 132
oahOptionsVector 113, 119
oahPureHelpAtomExpectingAValue 118
oahPureHelpAtomWithoutAValue 117
oahRGBColorAtom 115
oahRationalAtom 113
oahSubGroup 112
oahThreeBooleansAtom 113
oahVersionAtom 106, 134
ostream 212
pPrivateThisMethodHasBeenRun 80
packages 26
passes 28, 30
presentation 19, 27
rmcache 69
s 70, 73, 79, 81, 86, 280
samples 26
schemas 26
src 26, 28
star* 21
v* 43
v 17
validation 26
visitStart 57
visitStart (S_msrMeasure& elt) 248
wae 31
win32 26
xml2brl 66, 113
xml2guido 26
xml2lbr 113
xml2ly 43, 66, 113
xml2lyInsiderHandler 235
xml2xml 113
xmlattribute 199
xmlelement 199, 200
musicxml2ly 221
*AsString() () 88
//#define DEBUG_EARLY_OPTIONS
i 74
//#define DEBUG_INDENTER
i 74
//#define DEBUG_SPLITTING
i 74
handleOptionNameCommon () 120
acceptIn () 74
acceptOut () 74
addGraceNotesGroupBeforeAheadOfVoiceIfNeeded ()
274
appendBarLineToVoice () 274
appendCodaToVoice () 274
appendDampAllToVoice () 274
appendDampToVoice () 274
appendElementToMeasure () 244
appendEyeGlassesToVoice () 274
appendHarmonyToNoteHarmoniesList () . 307, 308
appendMeasureCloneToFullMeasureRests () .. 248
appendMeasureCloneToVoiceClone () 248
appendMeasureToSegment () 246, 248
appendNoteToVoice () 309, 325
appendPedalToVoice () 274
appendPendingMeasureRepeatToVoice () 248
appendSegnoToVoice () 274
appendStaffDetailsToPart () 64
appendStaffDetailsToVoice () 274
applyAtomWithDefaultValue () 120
applyAtomWithValue () 120, 137
applyEarlyOptionIfRelevant () 131
applyElement () 107, 120, 130, 239
applyOptionsFromElementUsesList () 128
asString() () 89
asStringShort() () 89
asString () 49, 74, 157
assertSegmentMeasuresListIsNotEmpty () ... 275
browseData () 59, 74, 288
browsedata () 55, 59
catchSignals () 343
checkMissingPendingArgvAtomExpectingAValueValue ()
122
collectPartMeasuresSlices () 189
collectStaffMeasuresSlices () 190, 191
compare* () 92
convert*() () 90
convertArgcArgvToOptionsAndArguments () . 121,
126
countnotes
visitStart 57
create*OahGroup () 233
create*PassComponent () 99
create*RepresentationComponent () 99
create* () 74, 75, 86
createAMeasureAndAppendItToPart () 251
createAMeasureAndAppendItToSegment () ... 174,
248
createAMeasureAndAppendItToVoice () . 248, 273
createFullMeasureRestsInVoice () 248
createGlobalHarmoniesExtraOahGroup () 139
createGlobalMxsr2msrOahGroup () 44
createInformationsRegularGroup () 136
createLibraryComponent () 101, 103
createMeasureRepeatFromItsFirstMeasures ()
248, 273, 278
createMsrRepresentationComponent () 99
createMusicxml2brailleConverterComponent ()
100
createMxmlScorePartWiseElement () 217
createNewLastSegmentForVoice () 273
createNewLastSegmentFromItsFirstMeasureForVoice ()
248, 274
createPartFiguredBassVoice () 318
createSegmentDeepClone () 272
createSegmentNewbornClone () 272
createTempoPerMinute () 80

createTheXml2lyOptionGroups () 235
createWithoutVoiceUplink () 305
create () 80, 272
displayMeasureRepeat () 91
fetchNames () 133
finalize*() () 88
finalize* () 255
finalizeAllTheMeasuresOfSegment () 255
finalizeChord () 255
finalizeCurrentChord () 255
finalizeCurrentMeasureClone () 255
finalizeFiguredBassMeasure () 255
finalizeHarmoniesMeasure () 255
finalizeLastAppendedMeasureInPart () 257
finalizeLastAppendedMeasureInStaff () 257
finalizeLastAppendedMeasureInVoice () 257
finalizeMeasureClone () 255
finalizeMeasure () 254, 255, 261
finalizePartAndAllItsMeasures () 257
finalizePartClone () 255
finalizePart () 255, 258
finalizeRegularMeasure () 255
finalizeRepeatEndInPart () 256
finalizeRepeatEndInStaff () 256
finalizeRepeatEndInVoice () 256
finalizeStaff () 255, 258, 259
finalizeTupletAndPopItFromTupletsStack () 255
finalizeTuplet () 255
finalizeVoiceAndAllItsMeasures () 257
finalizeVoice () 255, 259
fromString () 88
generateCodeForBrailleCell () 212
generateLilypondVersion () 116
get*() () 87
get* () 74
getClefStaffNumber () 216
getSetByUser () 116
getopt* () 112
handleFullMeasureRestsStartInVoiceClone ()
274
handleHooklessRepeatEndingEndInVoice () ..274
handleKnownArgvAtom () 120
handleOptionNameCommon () 120
handleOptionsAndArgumentsFromArgcArgv () .120
handlePendingFiguredBassElements () 323
handleVoiceLevelContainingRepeatEndWithoutStart ()
274
handleVoiceLevelRepeatEndWithStart () 274
handleVoiceLevelRepeatEndWithoutStart () .274
handleVoiceLevelRepeatEndingStartWithExplicitStart ()
248, 274
handleVoiceLevelRepeatEndingStartWithoutExplicitStart()
248, 274
handleVoiceLevelRepeatStart () 248, 274
identifySoloNotesAndRests () 191
initialize*() () 88
initialize* () 234
initializeBSR () 234
initializeHandlerMultiComponent () ..104, 105
initializeLPSR () 234
initializeLpsrPaperOptions () 139
initializeMSRBasicTypes () 234
initializeMSR () 234
initializeMsrGenerationAPI () 80
initializeOahBasicHelpOptions () 132, 135
initializeVoice () 272
lpsr2lilypondTranslator
generateLilypondVersion () 116
lpsrOahGroup
initializeLpsrPaperOptions () 139
lpsrScore
lpsrScore () 108
lpsrScore () 108
main () 343
mfAssert () 70
mfIndentedStreamBuf
sync () 46
mfMultiGenerationOutputKindAtom
mfMultiGenerationOutputKindAtom () 119
mfMultiGenerationOutputKindAtom () 119
mfcComponentDescr
printOwnHistory () 97
mfcMultiComponentUsedFromTheCLIKind
kComponentUsedFromTheCLIYes 98
mfcMultiComponent
printHistory () 98
print () 98
msdlTokenKind
AllFirst 84
AllLast 84
msr2bsrTranslator
finalizeCurrentMeasureClone () 255
msr2lpsrTranslator
visitStart (S_msrMeasure& elt) 248
msr2msrTranslator
visitStart (S_msrMeasure& elt) 248
msr2mxsrTranslator
translateMsrToMxsr () 216
msrChord
finalizeChord () 255
msrClef
getClefStaffNumber () 216
msrFullMeasureRests
appendMeasureCloneToFullMeasureRests ()
248
msrHarmonyKind
AllFirst 84
AllLast 84
TrueHarmoniesFirst 85
TrueHarmoniesLast 85
msrHarmony
createWithoutVoiceUplink () 305
msrMeasureElement
fMeasureElementPositionInMeasure .253, 270
msrMeasureRepeat
displayMeasureRepeat () 91
msrMeasuresSlicesSequence
identifySoloNotesAndRests () 191
msrMeasure
appendElementToMeasure () 244
finalizeFiguredBassMeasure () 255
finalizeHarmoniesMeasure () 255

finalizeMeasureClone () 255
finalizeMeasure () 254, 255, 261
finalizeRegularMeasure () 255
msrNote
appendHarmonyToNoteHarmoniesList () .. 307, 308
msrPart
appendStaffDetailsToPart () 64
browseData () 288
collectPartMeasuresSlices () 189
createAMeasureAndAppendItToPart () 251
createPartFiguredBassVoice () 318
finalizeLastAppendedMeasureInPart () .. 257
finalizePartAndAllItsMeasures () 257
finalizePartClone () 255
finalizePart () 255, 258
finalizeRepeatEndInPart () 256
msrPlacementKind
kPlacementAbove 165
msrSegment
appendMeasureToSegment () 246, 248
assertSegmentMeasuresListIsNotEmpty () 275
createAMeasureAndAppendItToSegment () 174, 248
createSegmentDeepClone () 272
createSegmentNewbornClone () 272
create () 272
finalizeAllTheMeasuresOfSegment () 255
msrSlurTypeKind
kSlurTypeRegularStart 81
msrStaff
collectStaffMeasuresSlices () 190, 191
finalizeLastAppendedMeasureInStaff () . 257
finalizeRepeatEndInStaff () 256
finalizeStaff () 255, 258, 259
msrStanza
K_NO_STANZA_NUMBER 80
msrTempo
createTempoPerMinute () 80
fTempoBeatUnit 165
fTempoEquivalentBeatUnit 165
fTempoKind 165
fTempoNotesRelationshipKind 165
fTempoNotesRelationshipLeftElements ... 165
fTempoNotesRelationshipRightElements .. 165
fTempoParenthesizedKind 165
fTempoPerMinute 165
fTempoPlacementKind 165
fTempoWordsList 165
kTempoNotesRelationshipEquals 165
msrTemp
tempoKindAsString () 80
msrTie
create () 80
msrTuplets
finalizeTuplet () 255
msrVoice
addGraceNotesGroupBeforeAheadOfVoiceIfNeeded () 274
appendBarLineToVoice () 274
appendCodaToVoice () 274
appendDampAllToVoice () 274
appendDampToVoice () 274
appendEyeGlassesToVoice () 274
appendMeasureCloneToVoiceClone () 248
appendNoteToVoice () 309, 325
appendPedalToVoice () 274
appendPendingMeasureRepeatToVoice () .. 248
appendSegnoToVoice () 274
appendStaffDetailsToVoice () 274
createAMeasureAndAppendItToVoice () .. 248, 273
createFullMeasureRestsInVoice () 248
createMeasureRepeatFromItsFirstMeasures () 248, 273, 278
createNewLastSegmentForVoice () 273
createNewLastSegmentFromItsFirstMeasureForVoice () 248, 274
fVoiceLastSegment 173
finalizeLastAppendedMeasureInVoice () . 257
finalizeRepeatEndInVoice () 256
finalizeVoiceAndAllItsMeasures () 257
finalizeVoice () 255, 259
handleFullMeasureRestsStartInVoiceClone () 274
handleHooklessRepeatEndingEndInVoice () 274
handleVoiceLevelContainingRepeatEndWithoutStart () 274
handleVoiceLevelRepeatEndWithStart () . 274
handleVoiceLevelRepeatEndWithoutStart () 274
handleVoiceLevelRepeatEndingStartWithExplicitStart 248, 274
handleVoiceLevelRepeatEndingStartWithoutExplicitSta 248, 274
handleVoiceLevelRepeatStart () ... 248, 274
initializeVoice () 272
musicxmlfile2lilypond () 112
mxsr2msrSkeletonBuilder
fThereAreHarmoniesToBeAttachedToCurrentNote 303
mxsr2msrTranslator
finalizeCurrentChord () 255
finalizeTupletAndPopItFromTupletsStack () 255
handlePendingFiguredBassElements () ... 323
populateNote () 306, 322
oahAtomStoringAValue
fSetByUser 115
oahEarlyOptions
applyEarlyOptionIfRelevant () 131
oahElement
fetchNames () 133
oahHandler
handleOptionNameCommon () 120
applyOptionsFromElementUsesList () 128
checkMissingPendingArgvAtomExpectingAValueValue () 122
fHandlerArgumentsVector 120
fHandlerMultiComponent 103--105
fPendingArgvAtomExpectingAValue 120

handleKnownArgvAtom () 120
handleOptionNameCommon () 120
handleOptionsAndArgumentsFromArgcArgv () 120
oahHistoryAtom
printHistory () 135
oahIntegerAtom
setIntegerVariable () 115
oahLengthAtom
applyAtomWithValue () 137
printAtomWithVariableOptionsValues () . 139
oahMacroAtom
applyElement () 239
fMacroAtomsList 239
oahOahGroup
initializeOahBasicHelpOptions () . 132, 135
printOahOahValues () 133
oahVersionAtom
applyElement () 107, 130
printVersion () 107
populateNote () 306, 322
printAtomWithVariableOptionsValues () 139
printHistory () 97, 98, 107, 134, 135
printMxsr () 203
printOahOahValues () 133
printOwnHistory () 97
printSlices () 64
printVersion () 97, 106, 107, 134, 135
print () 49, 74, 98, 157
set*() () 87
set*Variable () 115
set* () 74
setIntegerVariable () 115
sync () 46
tempoKindAsString () 80
translate*() () 90
translateLpsrToLilypondWithHandler () 47
translateMsrToLpsrScore () 53
translateMsrToMxsr () 216
visit* () 55, 147
visitEnd () 55, 147
visitStart () 55, 57, 147
xml2lyInsiderHandler
createTheXml2lyOptionGroups () 235
xml2lyRegularHandler
createInformationsRegularGroup () 136
xml2xmlInsiderHandler
xml2xmlInsiderHandler () 104
xml2xmlInsiderHandler () 104
xmlelement
fType 200
() 135
() 135

A

API ... 17, 22, 36, 37, 47, 98, 112, 113, 121, 126, 220, 228
argc/argv 120, 121
auto 23

B

basic blocs 180
BMML 27
Braille 17, 335, 338
branch 17, 23, 43
BSR 17, 32, 149, 195
bsr.cpp 80
bsrBasicTypes 195
bsrBasicTypes.h 195

C

C++11 ..... 23, 70, 81
cache 68, 69
cascade 251, 259
cascaded 64, 261
cascading 64, 257, 258
circularity 130
command line 22, 23, 28, 36, 37, 98, 112, 113, 119, 121, 126, 140
component 94, 96
tt *Component.h/.cpp 99
converter 32, 37
countnotes.cpp 57

D

data driven 55
*DeepClone 145
default 17
defensive programming 70
denormalization 143, 300, 314, 316, 328
description 17
displayMusicformatsHistory.cpp 103
displayMusicformatsVersion.cpp 103
DLL 78
Dominique Fober .. 16, 17, 19, 27, 55, 72, 198
drawing 280
drawn 154, 272, 300, 316

E

elements/factory.cpp 201
enableAbortToDebugErrors.h 42, 343
enableHarmoniesExtraOahIfDesired.h 140
enableTracingIfDesired.h 43
encapsulates 83
enumeration type 70, 73, 79, 81, 83, 86, 88, 96--98, 113, 119, 127, 136, 155, 156, 160--162, 165, 166, 173, 180--182, 185, 187, 195, 196, 200, 201, 234, 280

F

format 34
frozen 17
functions 28, 112

G

generator 36
Git 69
GUI 154
Guido 17, 334, 336, 339

H

has not been set yet 81

I

initialization 81, 233--235

insider 105, 113, 125, 126, 130, 132, 235
InsiderHandler.cpp 44
*IsNeeded 209

L
libMultipleInitsTest.cpp 121
libmusicxml2 .16, 17, 19, 26, 27, 68, 72, 79, 85, 86, 100, 198, 200, 202, 206
LilyPond 17, 37, 47, 108, 116, 154, 170, 193, 206--209, 221, 229, 314, 328, 334, 335, 337, 341
LilyPondIssue34.cpp 37, 121
lpsr2lilypondTranslator.cpp 116
lpsrOah.cpp 139
lpsrScores.cpp 108
LSPR 32, 149, 193, 227, 235

M
main () 28, 41, 120, 228
Makefile 24
master 17, 23
MEI 27
mfBool.h/.cpp 83
MFC 108
mfcBasicTypes.h 134
mfcComponents.h 33
mfcLibraryComponent.h/.cpp 101
mfEnumAll.h 71, 83, 84
mfIndentedTextOutput.cpp 74
mfIndentedTextOutput.cpp.h/.cpp 148
mfIndentedTextOutput.h 47, 48
mfIndentedTextOutput.h/.cpp 46, 66
mfMusicformatsError.h 121
mfStringsHandling.cpp 74
mfTiming.h/.cpp 52
Mikrokosmos3Wandering.cpp ...17, 36, 121, 126
MSDL 26, 334
msdl.cpp 121, 126
msdlScanner.cpp 146
msdlScanner.h 146
msdlTokens.cpp 84
MSR 32, 64, 99, 123, 145, 149, 152--154, 162, 173, 175, 206, 227, 235, 332, 334
msr2msrTranslator.h 147
msrBarLines.h 68, 280
msrBarlines.h 68
msrBasicTypes.cpp 70, 84
msrBasicTypes.h .83, 136, 158, 160, 161, 173, 181
msrElements.h 56
msrIdentification.h 153
msrMeasuresSlice.h 155
msrNotes.cpp 60, 309, 325
msrOah.cpp 112
msrOah.h 112
msrParts.h 301, 317
msrSegment.cpp 88
msrSegments.cpp 275
msrTempos.h 162
msrTies.h 87
msrTimeSignature.cpp 145
msrTimeSignature.h 145
msrVoices.cpp 272--274
msrVoices.h 91, 151, 152, 173
MuseScore 154
Music eXtended Markup Language 223
MusicFormats 1, 16, 17, 19, 20, 22, 23, 26, 28--30, 32, 33, 37, 41--43, 45, 47--49, 55, 63, 64, 68, 69, 71, 72, 74--76, 78--80, 83, 85, 86, 90, 91, 94, 96, 97, 100, 112, 121, 125, 127, 139, 140, 145, 149--152, 157, 160, 179, 185, 206, 228, 233, 238, 255, 256, 272, 301, 317, 332, 334, 343
MusicXML 16, 17, 26, 27, 35, 41, 57, 61, 81, 100, 109, 149, 152, 153, 156, 166, 179, 183, 198, 200--203, 206, 214, 216, 218, 221, 223, 251, 280, 300, 303, 316, 319, 332, 336, 338
MXSR ...17, 32, 100, 123, 149, 206, 207, 214, 217
mxsr2msrTranslator.cpp 307, 323

N
*NewbornClone 144
no error message whatsoever 147

O
OAH 132, 134, 136, 139, 140, 233, 235
oahAtomsCollection 113
oahAtomsCollection.cpp 115, 130, 137
oahAtomsCollection.h 136
oahBasicTypes.cpp 122, 128
oahBasicTypes.h 121
oahEarlyOptions.cpp 74, 131
oahEarlyOptions.h 130
oahElements.cpp 133
oahElements.h 127
oahOah.cpp 135
operating system 23, 69, 83

P
partial order 288
pass 35
PNG 19
position in measure 309
programming 17
pure help 117

R
regular 105, 113, 125, 126, 130, 132, 136
RegularHandler
  createOahRegularGroup () 134
createOahRegularGroup () 134
release notes 94, 98
representation 34
run 37

S
service 37, 126
smart pointer 189, 200, 201, 203
stopper sets 341

V
version 23, 43
very fine-grained 152

W
waeExceptions 42
waeHandlers.cpp 73
waeHandlers.h 73
Web 22, 26, 47, 121, 141, 228
Windows 78

X
xml2Any.cpp 121
xml2ly.cpp 39
xmlfile.h 202
xmlreader.h/.cpp 203