

MusicFormats maintainance guide

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

v0.9.70 – September 20, 2023

Jacques Menu

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

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

```
1 void msr2msrTranslator::visitStart (S_msrClef& elt)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gMsrOahGroup->getTraceMsrVisitors ()) {
5             std::stringstream ss;
6
7             ss <<
8                 "--> Start visiting msrClef" <<
9                 ", line " << elt->getInputStartLineNumber ();
10
11             gWaeHandler->waeTraceWithoutInputLocation (
12                 __FILE__, __LINE__,
13                 ss.str ());
14         }
15     #endif // MF_TRACE_IS_ENABLED
16
17     fCurrentVoiceClone->
18         appendClefToVoice (elt);
19 }
```

List of Figures

1.1	The MusicFormats architecture	19
17.1	The OAH classes hierarchy	136
20.1	The solo rests problem.	186
20.2	Humdrum-Scot keys	197
20.3	The MSR classes hierarchy.	199
20.4	Three segments in a voice	212
31.1	Zoltán Kodály's Mikrokosmos III Wandering	251
31.2	The LilyPondIssue34 score.	252
68.1	Braille for HelloWorld.xml with interpretation	403
68.2	Un Petit Air, par Jean Abou-Samra	406

Listings

15.1	<code>countnotes.cpp</code>	104
15.2	<code>Visiting <scaling /></code>	105
15.3	<code>msrDoubleTremolo::browseData (basevisitor* v)</code>	106

Contents

I	Overview of MusicFormats	15
1	Introduction	16
1.1	Acknowledgements	16
1.2	Prerequisites	16
1.3	Chronology	17
1.4	Zsh vs Bash.	18
1.5	The GitHub repository	18
2	Documentation.	20
2.1	L ^A T _E X macros	22
2.2	About this document	24
2.3	The MusicFormats architecture	24
2.4	User guide	24
2.5	API guide	24
2.6	Maintainance guide	24
3	Building MusicFormats	25
3.1	Cloning the repository.	25
3.2	One-shot partial <code>make</code> runs	25
3.3	<code>cmake</code> configuration	26
3.4	Library <code>Makefile</code>	26
3.5	CLI amplies <code>Makefile</code>	26
3.6	Building MusicFormats in practise	26
4	Code base structure	28
4.1	The <code>libmusicxml</code> folder	28
4.1.1	Embedding <code>libmusicxml</code> in MusicFormats	29
4.1.2	Upgrading the supported MusicXML version	29
4.2	The <code>doc</code> folder	31
4.3	The <code>schemas</code> folder	32
4.4	The <code>src</code> folder	32
4.5	The <code>validation</code> folder	36
5	Components	37
5.1	Components terminology	37
5.2	Formats	39
5.3	Representations	39
5.4	Passes	40
5.5	Generators	41
5.6	Converters	42
5.7	Running a service	42
6	Command line samples	46
7	Warning and errors (WAE)	47
8	The trace facility.	48
8.1	Activating the trace	48
8.2	Trace categories	49
8.3	Using traces in practise	49
8.4	Debugging traces handling.	49

9	Multi-lingual support	50
10	Textual input and output	51
10.1	Indented output streams	51
10.2	Creating indented output streams	52
10.3	Indenting the output	53
10.4	Printing descriptions	54
11	Binary data output	57
12	CPU measurements	58
 II Programming style and conventions		61
13	Programming style and conventions	62
13.1	Files naming conventions	62
13.2	Adding C++ files	64
13.3	Renaming C++ files	65
13.4	Source code layout	65
13.5	Defensive programming	66
13.6	Sanity checks	67
13.7	JMI comments	67
13.8	Exported symbols	68
13.9	Smart pointers	68
13.10	Files contents layout	70
13.11	<code>#define DEBUG*</code> code sections	71
13.12	Identifiers choice conventions	71
13.13	Exceptions and warnings/errors reporting	73
13.14	Exporting symbols for Windows DLLs	75
13.15	Dynamic type checking	76
13.16	Input line numbers	76
13.17	Static declarations	77
13.18	Avoiding MusicFormats multiple initializations	78
13.19	Enumeration types	78
13.20	yes/no enumerations types	80
13.21	Boolean values anyway	81
13.22	On/off values	81
13.23	Iterating over numeration types	82
13.24	<code>mfRational</code> numbers	84
13.25	Don't mix pure and typed scalar types	85
13.26	Default values	85
13.27	<code>create*</code> methods	86
13.28	<code>get*()</code> , <code>set*()</code> and <code>fetch*()</code> methods	87
13.29	<code>initialize*()</code> and <code>finalize*()</code> methods	88
13.30	<code>*asString()</code> and <code>*fromString()</code> functions	88
13.31	<code>translate*()</code> methods and <code>convert*()</code> functions	89
13.32	<code>context</code> arguments	91
13.33	Sorting and <code>compare*()</code> methods	92
13.34	Mutually dependent classes	93
13.34.1	Pre-declarations	93
13.34.2	Simple mutual dependency using separate header files	94
13.34.3	More complex mutual dependencies	96
13.35	Templates and functional programming usage	97

III	Maintenance tools	98
14	Maintenance tools	99
14.1	Building the library	99
14.2	Building the documentation	100
14.3	Releases creation	100
14.4	Checking source files contents	100
14.5	Handly ZSH functions and aliases	100
IV	The two-phase visitors pattern	101
15	The two-phase visitors pattern	102
15.1	Basic mechanism	102
15.2	Browser template classes	103
15.3	A first example: counting notes in MusicXML data	104
15.4	A more complex example	104
15.5	Data browsing order	106
15.6	Selectively inhibiting data browsing	106
15.6.1	Inhibiting data browsing in the code	106
15.6.2	Inhibiting data browsing by options	109
15.7	Adapting visitors to data browsing order with booleans	110
15.8	Adapting visitors to data browsing order with stacks	112
15.9	Avoiding the visiting pattern by cascading	112
V	MusicFormats components	114
16	MusicFormats components (MFC)	115
16.1	Versions numbers	115
16.2	Versions descriptions	116
16.3	Versions histories	116
16.4	Components descriptions	117
16.5	Multi-components	118
16.6	Versions history creation	120
16.6.1	Representations and passes components creation	120
16.6.2	Generators and converters components creation	121
16.6.3	MusicFormats library component creation	123
16.6.4	Version and history options handling	125
16.7	Accessing versions in regular handlers	126
16.8	Getting current version numbers	127
16.8.1	Current version numbers in options	127
16.8.2	Current version numbers in formats	129
16.8.3	Current version numbers in passes	131
VI	Options and help	133
17	Options and help (OAH)	134
17.1	OAH basics	134
17.2	Features	135
17.3	OAH classes inheritance	135
17.4	Value-fitted atoms	137
17.4.1	The <code>oahAtomStoringAValue</code> class	139
17.4.2	The <code>oahBooleanAtom</code> special case	140
17.4.3	Checking whether an option has been selected	141
17.4.4	The <code>oahAtomStoringAValue</code> sub-classes	141
17.5	Pure help atoms	143
17.5.1	Pure help atoms without a value	143
17.5.2	Pure help value-fitted atoms	143

17.6	Options implicitly storing a value	144
17.7	Options and help handling.	145
17.8	Basic OAH types	145
17.9	Prefixes handling	146
17.10	argc/argv versus oahOptionsVector	146
17.11	Applying options	147
17.12	A OAH atoms collection.	147
17.13	An option and help example	148
17.14	Options and help introspection	149
17.15	Insider versus regular handlers.	150
17.16	Deciphering the options and arguments	151
17.16.1	Options and arguments multi-pass analysis	151
17.16.2	Pure help runs	152
17.16.3	Applying options.	152
17.16.4	Early handling of some options	155
17.17	Implementing the -find option	156
17.18	Checking options consistency	159
17.19	Adding new options	159
17.19.1	'Representations' vs. 'passes' options	159
17.19.2	Using an existing OAH atom class	160
17.19.3	Creating a new OAH atom class without a value	161
17.19.4	Creating a new OAH value-fitted atom class.	164
17.20	Extra options	167
17.21	man pages generation	168
17.22	Specific global OAH groups	168
17.23	Visiting OAH groups	169

VII Representations 170

18	Representations general principles.	171
18.1	Trees vs graphs, denormalization.	171
18.2	Denormalization	171
18.2.1	Descriptions sharing	171
18.2.2	Multiple voices.	172
18.3	Newborn clones	172
18.4	Deep clones	174
18.5	Inheritance	175
18.5.1	Single inheritance	175
18.5.2	Single inheritance for smart pointers	176
18.5.3	Multiple inheritance for visitors.	177
18.5.4	Multiple inheritance in other classes	178
18.5.5	Reversibility	179
19	Displaying formats	180
19.1	Display categories	180
19.2	Displaying in practise	180

20	Music Scores Representation (MSR).	181
20.1	MSR basic types	182
20.2	Data matching across formats	183
20.3	Lengths.	184
20.4	Sounding and displayed durations	184
20.5	Measure positions and moments	185
20.6	Rests and skips	185
20.7	Solo notes and rests	186
20.8	Linear versus time-oriented representation	186
20.9	Spanners	187
20.10	Uplinks, direct uplinks and sidelinks	187
20.11	Printing descriptions	188
20.12	Pitches	189
20.13	Octaves.	191
20.14	NotesDurations	192
20.15	Alterations	192
20.16	Accidentals	192
20.17	Tempos.	193
20.17.1	Tempos notes	193
20.17.2	Tempos triplets	194
20.17.3	Tempos description	195
20.18	Clefs	196
20.19	Keys	197
20.20	Time signatures	198
20.21	MSR classes inheritance	200
20.22	Books	201
20.23	Scores	202
20.24	Part groups	202
20.25	Parts	202
20.26	Staves	203
20.27	Voice elements	203
20.28	Voices	204
20.29	Measures	205
20.30	Repeats patterns and replicas	206
20.31	Beat repeats	207
20.32	Measure repeats.	208
20.33	Multi-measure rests	209
20.34	Barlines	210
20.35	Repeats.	210
20.36	Segments	211
20.37	Notes and rests	212
20.38	Grace notes groups	213
20.39	Chords	213
20.40	Tuplets	213
20.41	Harmonies and figured bass similarities.	214
20.42	Harmonies	215
20.43	Figured bass	215
20.44	Lyrics	216
20.45	MIDI.	217
21	MSR time-oriented representation.	218
21.1	Note events	218
21.2	Simultaneous notes chunks	219
21.3	Measures slices	219
21.4	Measures slices sequences	219
21.5	Building the measures slices	220
21.5.1	Part measures slices	220
21.5.2	Staff measures slices	221

21.6	Solo notes and rests	221
21.7	A measures slices example	222
22	Path to voice.	223
23	LilyPond Scores Representation (LPSR).	224
23.1	LPSR basic types	224
23.2	Adapting LilyPond code generation to the target version number	225
24	Braille Scores Representation (BSR)	226
24.1	BSR basic types.	226
24.2	Representing cells	226
25	MusicXML Scores Representation (MXSR)	229
25.1	MusicXML elements and attributes	229
25.2	The <code>xmlelement</code> and <code>xmlattribute</code> types	230
25.3	Enumeration types for <code>xmlelement</code> 's <code>fType</code>	231
25.4	Classes for the <code>xmlements</code>	232
25.5	<code>xmlelement</code> trees	233
25.5.1	Creating <code>xmlelement</code> trees from textual data	234
25.5.2	Printing <code>xmlelement</code> trees	234
25.6	The <code>SXMLFile</code> type	234
VIII	Passes	236
26	The passes	237
26.1	Translating MusicXML data to an MXSR format	237
26.1.1	MusicXML coverage	237
26.1.2	Backup and forward handling.	237
26.2	Translating an MXSR to an MSR	237
26.3	Translating an MSR to an MXSR	237
26.4	Translating an MSR to another MSR	237
26.5	Translating an MSR to an LPSR.	238
26.6	Translating an LPSR to LilyPond code.	238
26.7	Translating an MSR to an BSR	238
26.8	Translating a BSR to another BSR	238
26.9	Translating an MXSR to Guido	238
27	LilyPond code generation	239
27.1	Basic principle	239
27.2	Generating Scheme functions in the LilyPond output	240
28	Braille generation	241
28.1	Basic principle	241
28.2	Output files name and contents options	242
28.3	Braille generators	242
28.4	Writing braille cells	243
29	MusicXML generation	245
29.1	Basic principle	245
29.2	Creating an <code>xmlelement</code>	245
29.3	Creating an <code>xmlelement</code> tree	246
29.4	Browsing the visited MSR score	247
29.5	Ancillary functions to create MXSR data.	248
30	Guido code generation	249
30.1	Basic principle	249
IX	Generators	250
31	The generators	251
31.1	<code>MusicAndHarmonies</code>	251
31.2	<code>Mikrokosmos3Wandering</code>	251
31.3	<code>LilyPondIssue34</code>	252

X	Converters	253
32	The converters	254
32.1	xml2ly	254
32.2	xml2brl	254
32.3	xml2xml	255
32.4	xml2gmn	255
32.5	msdlconverter	255
XI	Interfaces	256
33	Library interfaces	257
34	Representations interfaces	258
34.1	MSR interfaces	258
34.2	LPSR interfaces	258
34.3	MSDL interfaces	258
35	Passes interfaces	259
35.1	Translating MusicXML data to an MXSR	260
35.2	Translating an MXSR to an MSR	260
35.3	Translating an MSR to an MXSR	260
35.4	Translating an MSR to another MSR	260
35.5	Translating an MSR to an LPSR	260
35.6	Translating an LPSR to LilyPond code	260
35.7	Translating an MSR to an BSR	260
35.8	Translating a BSR to another BSR	260
35.9	Translating an MXSR to Guido	260
36	Converters interfaces	261
XII	Releases	263
36.1	The GitHub repository	264
36.1.1	Creating the releases	265
36.1.2	Security issue in recent MacOS™ versions	267
36.2	GitHub actions	268
37	MusicFormats releases	272
XIII	Selected topics	277
38	Initializations	278
38.1	Options and help initializations	278
38.2	Representations initializations	279
38.2.1	MSR initialization	279
38.2.2	LPSR initialization	280
38.2.3	BSR initialization	280
38.3	Passes initializations	280
38.4	Converters initializations	280
39	The OAH atoms collection	283
39.1	OAH macro atoms	283
39.2	A OAH macro atom example	285
39.3	LilyPond octave entry	286

40	Measures handling	288
40.1	Voices contents	288
40.2	Voice elements	288
40.3	Measure elements	289
40.4	Appending measure elements to a measure	289
40.5	Appending measures to a segment	292
40.6	Appending measures to a voice	294
40.7	Translating from MXSR to MSR	295
40.8	Translating from MXSR to MSR	296
40.9	Translating from MSR to MSR	297
40.10	Translating from MSR to LPSR	297
40.11	Translating from LPSR to LilyPond	297
41	Finalizations	298
41.1	Clones vs non-clones finalization	298
41.2	The finalization methods	299
41.3	Finalizing parts	302
41.4	Finalizing staves	303
41.5	Finalizing voices	304
41.6	Finalizing repeats	305
41.7	Finalizing measures	305
41.7.1	Finalizing regular measures	307
41.7.2	Finalizing harmonies measures	309
41.7.3	Finalizing figured bass measures	311
41.8	Determining measure positionss	312
42	Tempos handling	313
42.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	313
42.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	313
42.3	Translating from MSR to MSR (src/passes/msr2msr/)	313
42.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	313
42.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	313
43	Notes handling	314
43.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	314
43.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	314
43.3	Translating from MSR to MSR (src/passes/msr2msr/)	314
43.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	314
43.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	314
44	Segments handling	315
44.1	Segments creation	315
44.1.1	Creating a new last segment for a voice	316
44.1.2	Creating a new last segment for a voice from its first measure	317
44.2	Appending measures to a segment	318
44.3	Translating from MXSR to MSR	319
44.4	Translating from MXSR to MSR	319
44.5	Translating from MSR to MSR	319
44.6	Translating from MSR to LPSR	319
44.7	Translating from LPSR to LilyPond	320
45	Beat repeats handling	321
45.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	321
45.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	321
45.3	Translating from MSR to MSR (src/passes/msr2msr/)	321
45.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	321
45.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	321

46	Measure repeats handling	322
46.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	322
46.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	322
46.3	Translating from MSR to MSR (src/passes/msr2msr/)	322
46.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	322
46.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	322
47	Multi-measure rests handling	323
48	Repeats handling	324
48.1	Translating repeats from MXSR to MSR	324
48.2	Translating repeats from MXSR to MSR	329
48.3	Translating repeats from MSR to MSR	329
48.4	Translating repeats from MSR to LPSR	329
48.5	Translating repeats from LPSR to LilyPond	330
49	Voices handling	331
49.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	331
49.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	331
49.3	Translating from MSR to MSR (src/passes/msr2msr/)	331
49.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	331
49.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	331
50	Staves handling	332
50.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	332
50.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	332
50.3	Translating from MSR to MSR (src/passes/msr2msr/)	332
50.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	332
50.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	332
51	Parts handling	333
51.1	Parts browsing	333
51.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	334
51.3	Translating from MXSR to MSR (src/passes/mxsr2msr/)	334
51.4	Translating from MSR to MSR (src/passes/msr2msr/)	334
51.5	Translating from MSR to LPSR (src/passes/msr2lpsr/)	334
51.6	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	334
52	Part groups handling	335
53	Scores handling	337
53.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	337
53.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	337
53.3	Translating from MSR to MSR (src/passes/msr2msr/)	337
53.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	337
53.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	337
54	Books handling	338
54.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	338
54.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	338
54.3	Translating from MSR to MSR (src/passes/msr2msr/)	338
54.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	338
54.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	338
55	Ornaments handling	339
55.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	339
55.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	339
55.3	Translating from MSR to MSR (src/passes/msr2msr/)	339
55.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	339
55.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	339
56	Ties handling	340
56.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	340
56.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	340
56.3	Translating from MSR to MSR (src/passes/msr2msr/)	340
56.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	340
56.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	340

57	Dynamics handling	341
57.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	341
57.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	341
57.3	Translating from MSR to MSR (src/passes/msr2msr/)	341
57.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	341
57.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	341
58	Beams handling	342
58.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	342
58.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	342
58.3	Translating from MSR to MSR (src/passes/msr2msr/)	342
58.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	342
58.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	342
59	Slurs handling	343
59.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	343
59.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	343
59.3	Translating from MSR to MSR (src/passes/msr2msr/)	343
59.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	343
59.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	343
60	Grace notes groups handling	344
60.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	344
60.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	344
60.3	Translating from MSR to MSR (src/passes/msr2msr/)	344
60.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	344
60.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	344
61	Chords handling	345
61.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	345
61.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	345
61.3	Translating from MSR to MSR (src/passes/msr2msr/)	345
61.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	345
61.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	345
62	Tuplets handling	346
62.1	Translating from MXSR to MSR (src/passes/mxsr2msr/)	346
62.2	Translating from MXSR to MSR (src/passes/mxsr2msr/)	346
62.3	Translating from MSR to MSR (src/passes/msr2msr/)	346
62.4	Translating from MSR to LPSR (src/passes/msr2lpsr/)	346
62.5	Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)	346
63	Harmonies handling	347
63.1	Harmonies in MusicXML	347
63.2	Harmonies in MSR	348
63.3	Harmonies staves and voices	349
63.4	Harmonies staves creation	350
63.5	Translating harmonies from MXSR to MSR	351
63.5.1	First <code>S_harmony</code> visit.	352
63.5.2	Second <code>S_harmony</code> visit	353
63.5.3	Attaching <code>msrHarmony</code> instances to notes	355
63.5.4	Populating <code>msrHarmony</code> instances	355
63.5.5	First <code>S_harmony</code> visit.	357
63.5.6	Inserting <code>msrHarmony</code> instances in the part harmonies voice	358
63.6	Translating harmonies from MSR to MSR	358
63.7	Translating harmonies from MSR to LPSR	360
63.8	Translating harmonies from LPSR to LilyPond	363

64	Figured bass handling	365
64.1	Figured bass in MusicXML	365
64.2	Figured bass description	366
64.3	Figured bass staves and voices	367
64.4	Figured bass staves creation	368
64.5	Translating figured bass from MXSR to MSR.	369
64.5.1	First <code>S_figured_bass</code> visit.	371
64.5.2	Second <code>S_figured_bass</code> visit.	372
64.5.3	Attaching <code>msrFiguredBass</code> instances to notes	373
64.5.4	Populating <code>msrFiguredBass</code> instances	374
64.5.5	Inserting <code>S_msFiguredBass</code> instances in the part figured bass voice	376
64.6	Translating figured bass from MSR to MSR	376
64.7	Translating figured bass from MSR to LPSR	378
64.8	Translating figured bass from LPSR to LilyPond	379
65	Lyrics handling.	381
65.1	Translating from MXSR to MSR (<code>src/passes/mxsr2msr/</code>)	381
65.2	Translating from MXSR to MSR (<code>src/passes/mxsr2msr/</code>)	381
65.3	Translating from MSR to MSR (<code>src/passes/msr2msr/</code>)	381
65.4	Translating from MSR to LPSR (<code>src/passes/msr2lpsr/</code>)	381
65.5	Translating from LPSR to LilyPond (<code>src/passes/lpsr2lilypond/</code>)	381
66	MIDI handling	382
66.1	Translating from MXSR to MSR (<code>src/passes/mxsr2msr/</code>)	382
66.2	Translating from MXSR to MSR (<code>src/passes/mxsr2msr/</code>)	382
66.3	Translating from MSR to MSR (<code>src/passes/msr2msr/</code>)	382
66.4	Translating from MSR to LPSR (<code>src/passes/msr2lpsr/</code>)	382
66.5	Translating from LPSR to LilyPond (<code>src/passes/lpsr2lilypond/</code>)	382
XIV	MusicFormats Scripting Language (MFSL)	383
67	MFSL (MusicFormats Scripting Language)	384
67.1	A script example	384
67.2	Implementation principles	385
67.3	The contents of the MFSL folder.	385
67.4	The MFSL basic types	386
67.5	The MFSL Makefile	386
67.6	Locations handling	387
67.7	Tokens description	387
67.8	The driver	388
67.9	Lexical analysis	390
67.9.1	Flex options	390
67.9.2	Flex regular expressions	391
67.10	Syntax and semantic analysis	393
67.10.1	Bison options for MFSL	393
67.10.2	The MFSL tokens	394
67.10.3	The MFSL non-terminals and axiom	394
67.11	Interface to the MFSL parser	395
67.12	Running the example MFSL script.	396
67.12.1	Error recovery	397

XV	Music Scores Description Language (MSDL)	398
68	MSDL (Music Scores Description Language)	399
68.1	Main features of MSDL	399
68.2	MSDL basic types	399
68.3	What the MSDL converter does	400
68.3.1	LilyPond generation	400
68.3.2	Braille generation	400
68.3.3	MusicXML generation	401
68.3.4	Guido generation	401
68.4	A first example	401
68.5	First example output from the MSDL converter	402
68.5.1	LilyPond output	402
68.5.2	Braille output	403
68.5.3	MusicXML output	403
68.5.4	Guido output	404
68.6	A more realistic example	405
68.7	Multi-language support	406
68.7.1	Multi-language messages handling	406
68.7.2	Multi-language keywords handling	406
68.8	Lexical analysis	406
68.9	Music Scores Descriptions Representation (MSDR)	406
68.10	Syntax and semantic analysis	406
68.10.1	Error recovery	406
XVI	Debugging	407
69	Debugging	408
69.1	Useful options	409
69.2	Removing the results of a build	409
69.3	Reverting to a previous MusicFormats version	410
70	Locating a bug with Git's bisection	411
70.1	Locating a bug at random in the Git log	412
70.2	Locating a bug in the commits with Git's bisection	413
70.3	Locating the bug in the code base	414
XVII	Indexes	416

Part I

Overview of MusicFormats

Chapter 1

Introduction

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

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

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

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

1.1 Acknowledgements

Many thanks to Dominique Dominique Fober, the designer and maintainer of the `libmusicxml2` library. His two-phase visitor pattern has proved very strong and useful in the implementation of MusicFormats.

Lars Opfermann of the soundnotation company (<https://soundnotation.com>) has also played an important role, testing `xml2ly` intensively and supplying many MusicXML examples to help fix bugs. This help has been extremely useful and appreciated.

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 MusicXML if relevant, starting maybe from [IntroductionToMusicXML.pdf](#). A deep knowledge of that subject comes with experience;
- study the architecture of MusicFormats, presented in detail at [MusicFormatsFigures.pdf](#).

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

1.3 Chronology

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

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

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

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

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

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

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

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

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

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

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

1.4 Zsh vs Bash

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

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

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

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

Another useful tool is `devtools/CheckGIndenterUsage.cpp`, whose name is self-comprehensive.

1.5 The GitHub repository

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

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

Figure 1.1: The MusicFormats architecture



Chapter 2

Documentation

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

All the L^AT_EX files have an initial '`! TEX root`' comment. This is a TeXShop specific feature, allowing a documentation to be composed from any of the files that it imports, such as:

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

The documentation/ directory contains:

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

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

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

```

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

```

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

```

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

```

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

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

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

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

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

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

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

```

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

```

and:

```

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

```

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

2.1 L^AT_EX macros

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

```

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

```

For example:

```

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```


2.2 About this document

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

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

2.3 The MusicFormats architecture

2.4 User guide

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

2.6 Maintainance guide

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

Chapter 3

Building MusicFormats

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

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

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

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

3.1 Cloning the repository

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

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

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

3.2 One-shot partial make runs

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

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

3.3 cmake configuration

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

3.4 Library Makefile

This Makefile is `build/Makefile`.

3.5 CLI amplex Makefile

This Makefile is `src/clisamples/Makefile`.

3.6 Building MusicFormats in practise

Once in the local repository clone, just execute:

```
1 cd build
2 make
```

The resulting executables are in `build/bin`:

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

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

```

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

```

Chapter 4

Code base structure

The code base of the MusicFormats library contains:

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

4.1 The `libmusicxml` folder

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

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

4.1.1 Embedding libmusicxml in MusicFormats

libmusicxml2 was cloned initially like this:

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

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

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

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

4.1.2 Upgrading the supported MusicXML version

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

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

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

First, create the `Makefile` for version 4.0:

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

The following methods definitions are thus missing:

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

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

4.2 The doc folder

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

```

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

```

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


```

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

```

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

```

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

```

4.3 The schemas folder

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

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

```

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

```

4.4 The src folder

The src folder has the following structure:

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

```

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

```

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

```

- msdl2braille
- msdl2guido
- msdl2lilypond
- msdl2musicxml
- msdlconverter

- msr2braille
- msr2guido
- msr2lilypond
- msr2musicxml

- musicxml2braille
- musicxml2guido
- musicxml2lilypond
- musicxml2musicxml

```

- generators :

```

- LilyPondIssue34
- Mikrokosmos3Wandering

```

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

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll components
2 total 168
3 0 drwxr-xr-x 7 jacquesmenu staff 224 Oct 22 08:53:06 2021 ./
4 0 drwxr-xr-x 19 jacquesmenu staff 608 Oct 22 05:29:29 2021 ../
5 8 -rw-r--r--@ 1 jacquesmenu staff 1106 Oct 22 09:16:21 2021 mfcComponents.h
6 96 -rw-r--r--@ 1 jacquesmenu staff 46691 Nov 8 12:59:57 2021 mfcBasicTypes.cpp
7 40 -rw-r--r--@ 1 jacquesmenu staff 20121 Nov 8 12:59:43 2021 mfcBasicTypes.h
8 16 -rw-r--r--@ 1 jacquesmenu staff 4950 Nov 8 12:59:08 2021 mfcLibraryComponent.
   cpp
9 8 -rw-r--r--@ 1 jacquesmenu staff 605 Oct 22 10:36:30 2021 mfLibraryComponent.h

```

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

```

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

```

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

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll oah
2 total 1456
3 0 drwxr-xr-x 34 jacquesmenu staff 1088 Nov 16 08:12:11 2021 ./
4 0 drwxr-xr-x 20 jacquesmenu staff 640 Nov 16 08:12:03 2021 ../
5 48 -rw-r--r--@ 1 jacquesmenu staff 23743 Nov 16 08:16:55 2021 basicOah2manPage.
  cpp
6 16 -rw-r--r--@ 1 jacquesmenu staff 5202 Nov 15 12:56:16 2021
  oahBasicOah2manPage.h
7 8 -rw-r--r--@ 1 jacquesmenu staff 539 Jun 6 06:38:55 2021
  mfEnableHarmoniesExtraSetting.h
8 8 -rw-r--r--@ 1 jacquesmenu staff 526 Oct 11 11:56:29 2021
  mfPreprocessorSettings.h
9 72 -rw-r--r--@ 1 jacquesmenu staff 34280 Nov 16 08:16:55 2021 harmoniesExtraOah.
  cpp
10 24 -rw-r--r--@ 1 jacquesmenu staff 11848 Nov 15 12:56:16 2021
  oahHarmoniesExtraOah.h
11 16 -rw-r--r--@ 1 jacquesmenu staff 5154 Nov 16 08:16:55 2021
  harmoniesExtraOah2manPage.cpp
12 8 -rw-r--r--@ 1 jacquesmenu staff 1689 Nov 15 12:56:16 2021
  oahHarmoniesExtraOah2manPage.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 mfOutputFileOah.h
34 | 8 -rwxr--r--@ 1 jacquesmenu staff      236 Oct 23 12:02:12 2021 zsh_test.zsh*

```

- `formatsgeneration` : support for various output kinds

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

- `passes` : code for the individual passes

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

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

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

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

```

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

```

4.5 The validation folder

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

```

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

```

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

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

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

Chapter 5

Components

5.1 Components terminology

In compiler writing terminology:

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

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

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

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

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

```

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

```

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

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

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

5.2 Formats

The formats are in `src/formats`:

```

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

```

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

```

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

```

5.3 Representations

The representations are in `src/representations`:

```

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

```


5.4 Passes

The passes are in `src/passes`:

```

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

```

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

```

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

```

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

```

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

```

```

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

```

5.5 Generators

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

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

```

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

```

For example:

```

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

```

```

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

```

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

```

1 jacquesmenu@macmini: ~ > LilyPondIssue34 -musicxml -a
2 What LilyPondIssue34 does:
3
4      This multi-pass generator creates a textual representation
5      of the LilyPondIssue34 score.
6      It basically performs 4 passes when generating MusicXML output output:
7
8      Pass 1:  generate a first MSR for the LilyPondIssue34 score
9      Pass 2:  converts the first MSR a second MSR, to apply options;
10     Pass 3:  converts the second MSR into an MusicXML tree;
11     Pass 4:  converts the MusicXML tree to MusicXML code
12             and writes it to standard output.
13
14     Other passes are performed according to the options, such as
15     displaying views of the internal data or printing a summary of the score.
16
17     The activity log and warning/error messages go to standard error.

```

5.6 Converters

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

```

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

```

5.7 Running a service

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

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

```

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

```

```

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

```

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

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

```

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

```

Then the various run data can be accessed easily:

```

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

```

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

```

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

```

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

Chapter 6

Command line samples

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

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

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

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

Chapter 7

Warning and errors (WAE)

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

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

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

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

```
1 // convert the LPSR into LilyPond code  
2 try {  
3     translateLpsrToLilypond (  
4         theLpsrScore,  
5         gMsr0ahGroup,  
6         gLpsr0ahGroup,  
7         passIDKind,  
8         passDescription,  
9         lilypondStandardOutputStream);  
10 }  
11 catch (lpsr2lilypondException& e) {  
12     mfDisplayException (e, gOutput);  
13     return;  
14 }  
15 catch (std::exception& e) {  
16     mfDisplayException (e, gOutput);  
17     return;  
18 }
```

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

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


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

Trace is controlled by `MF_TRACE_IS_ENABLED`, defined or nor in `src/oah/mfPreprocessorSettings.h`:

```
1 #ifndef __enableTraceIfDesired__
2 #define __enableTraceIfDesired__
3
4 #ifndef MF_TRACE_IS_ENABLED
5     // comment the following definition if no trace is desired
6     #define MF_TRACE_IS_ENABLED
7 #endif // MF_TRACE_IS_ENABLED
8
9 #endif // __enableTraceIfDesired__
```

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

```
1 #include "mfPreprocessorSettings.h"
```

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

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

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

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

```

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

```

8.2 Trace categories

8.3 Using traces in practise

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

```

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

```

8.4 Debugging traces handling

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

```

1 gMsrOahGroup->getUseFilenameAsWorkCreditTypeTitle ()

```

the way to go is to:

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

Chapter 9

Multi-lingual support

MusicFormats supports multiple languages in various areas:

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

Chapter 10

Textual input and output

10.1 Indented output streams

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

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

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

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

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

```

37     return 0;
38 }

```

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

10.2 Creating indented output streams

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

```

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

```

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

So when and where are `mfIndentedOstream` instances created?

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

The parameters to this function are:

```

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

```

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

```

1 #define gIndenter mfOutputIndenter::gGlobalOStreamIndenter

```

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

```

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

```

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

```

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

```

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

10.3 Indenting the output

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

```

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

```

A typical sequence to produce indented output is:

```

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

```

```

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

```

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

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

```

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

```

To debug this:

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

```

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

```

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

```

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

```

10.4 Printing descriptions

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

```

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

```

There are also more specific methods such as:

```

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

```

Note that:

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

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

```

1 std::ostream& operator << (std::ostream& os, const S_msrElement& elt)
2 {
3     if (elt) {
4         elt->print (os);
5     }
6     else {
7         os << "[NULL]" << std::endl;
8     }
9
10    return os;
11 }

```

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

```

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

```

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

```

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

```

A typical sequence to produce indented output is:

```

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

```



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

The global indenter is gIndenter:

```
1  // the global output indenter for general use  
2  EXP mfOutputIndenter& getGlobalOutputIndenter ();  
3  
4  // useful shortcut macros  
5  #define gIndenter getGlobalOutputIndenter ()  
6  #define gTab      getGlobalOutputIndenter ().getSpacer ()
```

The main indented output streams are gOutput and gLog:

```
1  // the global log and output indented streams  
2  EXP extern S_mfIndentedOstream gGlobalOutputIndentedOstream;  
3  EXP extern S_mfIndentedOstream gGlobalLogIndentedOstream;  
4  
5  #define gOutput *gGlobalOutputIndentedOstream  
6  #define gLog    *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)
Pass 1	Handle the options and arguments from <code>argc/argv</code>	mandatory	0.01187
Pass 2a	Create an MXSR from a MusicXML file	mandatory	0.00471
Pass 2b	Create an MSR skeleton from the MXSR	mandatory	0.00222
Pass 4	Populate the MSR skeleton from MusicXML data	mandatory	0.00405
Pass 5	Convert the MSR into an LPSR	mandatory	0.00137
	Convert the LPSR into LilyPond code	mandatory	0.00136
Total (sec)	Mandatory	Optional	
0.02558	0.02558	0.00000	

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

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

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

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

```

1 S_lpsrScore translateMsrToLpsr (
2     const S_msrScore&          originalMsrScore,
3     const S_msr0ahGroup&      msrOpts,
4     const S_lpsr0ahGroup&     lpsrOpts,
5     mfPassIDKind              passIDKind,
6     std::string                passDescription,
7     const S_mfcMultiComponent& multiComponent)
8 {
9     if (gGlobalLpsr2lilypond0ahGroup->getNoLilypondCode ()) {
10         gLog <<
11             "Option '-nolpc, -no-lilypond-code' is set, no LPSR is created" <<
12             std::endl;
13
14         return nullptr;
15     }
16
17 #ifdef MF_SANITY_CHECKS_ARE_ENABLED
18     // sanity check
19     mfAssert (
20         __FILE__, __LINE__,
21         originalMsrScore != nullptr,
22         "originalMsrScore is null");
23 #endif // MF_SANITY_CHECKS_ARE_ENABLED
24
25     // start the clock
26     clock_t startClock = clock ();
27
28     // set the global current passID
29     setGlobalCurrentPassIDKind (passIDKind);
30
31 #ifdef MF_TRACE_IS_ENABLED
32     if (gEarlyOptions.getEarlyTracePasses ()) {
33         std::string separator =
34             "%-----";
35
36         std::stringstream ss;
37
38         ss <<
39             std::endl <<
40             separator <<
41             std::endl <<
42             gTab <<
43             gLanguage->passIDKindAsString (passIDKind) << ": " << passDescription <<
44             std::endl <<
45             separator;
46
47         gWaeHandler->waeTraceWithoutInputLocation (
48             __FILE__, __LINE__,
49             ss.str ());
50     }
51 #endif // MF_TRACE_IS_ENABLED
52
53     // create an msr2lpsrTranslator
54     msr2lpsrTranslator
55         translator (
56             originalMsrScore);
57
58     // build the LPSR score
59     S_lpsrScore
60         resultingLpsr =
61         translator.translateMsrToLpsr (
62             originalMsrScore,
63             multiComponent);
64

```

```
65  clock_t endClock = clock ();
66
67  // register time spent
68  gGlobalTimingItemsList.appendTimingItem (
69      passIDKind,
70      passDescription,
71      mfTimingItemKind::kMandatory,
72      startClock,
73      endClock);
```

Part II

Programming style and conventions

Chapter 13

Programming style and conventions

13.1 Files naming conventions

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

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

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

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

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

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

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

```

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

```


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

```

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

```

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

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

13.2 Adding C++ files

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

When building MusicFormats with:

```

1 cd build
2 make

```

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

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

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

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

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

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

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

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

13.3 Renaming C++ files

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

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

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

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

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

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

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

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

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

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

13.4 Source code layout

The following text-editing conventions are used:

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

13.5 Defensive programming

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

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

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

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

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

```

1  template< typename T >
2  class    mfEnumAll
3  {
4      public:
5
6          class    Iterator
7          {
8              public:
9
10             Iterator (int value)
11                 : fIterationIndex (value)
12                 {}
13
14             T operator* (void) const
15                 { return (T) fIterationIndex; }
16
17             void operator++ (void)
18                 { ++fIterationIndex; }
19
20             Bool operator != (Iterator rhs)
21                 { return fIterationIndex != rhs.fIterationIndex; }
22
23             private:
24
25                 int fIterationIndex;
26
27     };
28 };

```

13.6 Sanity checks

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

```

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

```

13.7 JMI comments

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

13.8 Exported symbols

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

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

```
1 EXP S_mxs0ahGroup createGlobalMxs0ahGroup ();
```

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

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

13.9 Smart pointers

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

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

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

The reference counting is done in class `smartable`:

```

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

```

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

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

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

```

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

```

```

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

```

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

```

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

```

13.10 Files contents layout

Indentation is done by two spaces, avoiding TAB characters.

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

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

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

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

```

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

```

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

13.11 #define DEBUG* code sections

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

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

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

13.12 Identifiers choice conventions

The following rules apply:

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

```

1 enum class msrBassFigurePrefixKind {
2     kBassFigurePrefix_UNKNOWN_ ,
3     kBassFigurePrefixDoubleFlat, kBassFigurePrefixFlat, kBassFigurePrefixFlat,
4     kBassFigurePrefixNatural,
5     kBassFigurePrefixSharpSharp, kBassFigurePrefixSharp, kBassFigurePrefixDoubleSharp
6 };

```

- all classes names have a prefix indicating which part of MusicFormats there belong to, such as class `msrTimeSignature`, `oahAtomStoringAValue` and `msdlKeywordsLanguageAtom`;
- all classes member fields start with 'f';
- all class static fields start with 's';
- all global variables start with 'pGlobal', in which case there are `set...` and `get...` macros to access them, starting with 'g' – this is to facilitate typing and names completion in code editors;
- some (all? JMI) start with 'g';
- all variables private to methods start with 'pPrivate';
- some `K_*` constants are declared as static class constant members, such as in `src/formats/msr/msrParts.h`:


```

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

```

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

```

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

```

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

```

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

```

or

```

1 fStringToDalSegnoKindMapVariable;

```

or

```

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

```

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

```

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

```

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

13.13 Exceptions and warnings/errors reporting

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

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

```

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

```

An example of `std::exception` is:

```

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

```

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

```

1 void oahValueFittedAtom::applyValueLessAtom (std::ostream& os)
2 {
3     std::stringstream ss;
4
5     ss <<
6         "Applying value-fitted atom '" <<
7         fetchNames () <<
8         "' without a value";
9
10    oahInternalError (ss.str ());

```

```

11 }
12
13 fSelected = true;

```

and:

```

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

```

Another one is:

```

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

```

```

40 // append it to the current part
41 fCurrentPart->
42     appendAccordionRegistrationToPart (
43         accordionRegistration);
44 }
45 }

```

13.14 Exporting symbols for Windows DLLs

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

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

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

```

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

```

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

```

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

```

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

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

```

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

```

```

32
33 #endif
34
35 #endif

```

13.15 Dynamic type checking

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

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

```

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

```

13.16 Input line numbers

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

```

1  int fInputStartLineNumber;

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

1 line <number>

```

with a single space before the number, as in:

```

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

```

This helps locating such occurrences in the debug process.

13.17 Static declarations

They are used for:

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

13.18 Avoiding MusicFormats multiple initializations

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

```

1 void initializeBSR ()
2 {
3     // protect library against multiple initializations
4     static Bool pPrivateThisMethodHasBeenRun (false);
5
6     if (! pPrivateThisMethodHasBeenRun) {
7 #ifdef MF_TRACE_IS_ENABLED
8         if (gEarlyOptions.getTraceEarlyOptions () && ! gEarlyOptions.getEarlyQuietOption ()) {
9             gLog <<
10                "Initializing BSR basic types handling";
11
12                gWaeHandler->waeTraceWithoutInputLocation (
13                    __FILE__, __LINE__,
14                    ss.str ());
15            }
16 #endif // MF_TRACE_IS_ENABLED
17
18            // BSR braille output kinds handling
19            // -----
20
21            initializeBsrBrailleOutputKindsMap ();
22
23            // BSR texts languages handling
24            // -----
25
26            initializeBsrTextsLanguageKindsMap ();
27
28            pPrivateThisMethodHasBeenRun = true;
29        }
30    }

```

13.19 Enumeration types

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

```

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

```

This prevents enumeration constants name conflicts across enumeration 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*_UNKNOWN_` 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++17 implicit initialization to the the equivalent of 0.

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

```

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

```

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

```

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

```

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

```

1 enum class msrChordInKind {
2     kChordIn_UNKNOWN_,
3
4     kChordInMeasure,
5     kChordInTuplet,

```



```

6   kChordInGraceNotesGroup
7   };
8
9   EXP std::string msrChordInKindAsString (
10      msrChordInKind chordInKind);
11
12   std::ostream& operator << (std::ostream& os, const msrChordInKind& elt);

```

The implementation is:

```

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

```

13.20 yes/no enumerations types

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

```

1   void setCombinedBooleanVariables (Bool value);

```

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

This is why we use enum classes such as:

```

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

```

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

```

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

```

13.21 Boolean values anyway

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

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

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

13.22 On/off values

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

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

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

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

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

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

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

```

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

```

13.23 Iterating over numeration types

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

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

13.24 mfRational numbers

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

```

1 class EXP mfRational {
2     // ... ..
3
4     private:
5
6         // private fields
7         // -----
8
9         long int          fNumerator;
10        long int          fDenominator;
11
12        // used by rationalise()
13        // long int          gcd(long int a, long int b);
14
15        // ... ..
16 };

```

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

13.25 Don't mix pure and typed scalar types

A whole notes value is described by class `msrWholeNotes`, defined in `src/formats/msr/msrNotesDurations.h/.cpp`. It contains a numerator and denominator, as class `msrRational` does, but those types are distinct, since an `msrWholeNotes` instance is a fraction of a whole note, note a mere typeless fraction.

MusicFormats uses `msrWholeNotes` instances for notes sounding and display whole notes values;

For the same reason, class `msrTupletFactor`, defined in `src/formats/msr/msrTupletFactors.h/.cpp`, is used with specifically named fields:

```

1 class EXP msrTupletFactor
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        int                fTupletActualNotes;
11        int                fTupletNormalNotes;
12 };

```

13.26 Default values

The guide lines for MusicFormats in this matter are:

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

```

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

```

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

13.27 create* methods

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

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

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

```

1  class EXP msrTempo : public msrMeasureElement
2  {
3      // ... ..
4
5      static SMARTP<msrTempo> createTempoWordsOnly (
6          int                inputLineNumber ,
7          S_msrWords         tempoWords);
8
9      static SMARTP<msrTempo> createTempoPerMinute (
10         int                inputLineNumber ,
11         msrDottedNotesDuration tempoBeatUnit ,
12         std::string         tempoPerMinute ,
13         msrTempoParenthesizedKind
14             tempoParenthesizedKind ,
15         msrPlacementKind    tempoPlacementKind);
16
17     static SMARTP<msrTempo> createTempoBeatUnitEquivalent (
18         int                inputLineNumber ,
19         msrDottedNotesDuration tempoBeatUnit ,
20         msrDottedNotesDuration tempoEquivalentBeatUnit ,
21         msrTempoParenthesizedKind
22             tempoParenthesizedKind ,
23         msrPlacementKind    tempoPlacementKind);
24
25     static SMARTP<msrTempo> createTempoNotesRelationship (
26         int                inputLineNumber ,
27         S_msrTempoNotesRelationshipElements
28             tempoNotesRelationshipLeftElements ,
29         msrTempoNotesRelationshipKind
30             tempoNotesRelationshipKind ,
31         S_msrTempoNotesRelationshipElements
32             tempoNotesRelationshipRightElements ,
33         msrTempoParenthesizedKind
34             tempoParenthesizedKind ,
35         msrPlacementKind    tempoPlacementKind);
36
37     // ... ..
38 };

```

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

```

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

```

```

14 // ... ..
15 };

```

13.28 `get*()`, `set*()` and `fetch*()` methods

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

```

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

```

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

```

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

```

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

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

```

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

```

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


```

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

```

13.29 initialize*() and finalize*() methods

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

```

1  msrPart::msrPart (
2      int          inputLineNumber,
3      std::string  partID,
4      S_msrPartGroup partUpLinkToPartGroup)
5      : msrPartGroupElement (inputLineNumber)
6  {
7      // replace spaces in partID to set fPartID
8      for_each (
9          partID.begin (),
10         partID.end (),
11         mfStringSpaceReplacer (fPartID, '_'));
12
13     /* JMI
14     #ifdef MF_SANITY_CHECKS_ARE_ENABLED
15         // sanity check
16         mfAssert (
17             __FILE__, __LINE__,
18             partUpLinkToPartGroup != nullptr,
19             "partUpLinkToPartGroup is null");
20     #endif // MF_SANITY_CHECKS_ARE_ENABLED
21
22     // set part number
23     fPartAbsoluteNumber = ++sPartsCounter;
24
25     // set part's part group upLink
26     fPartUpLinkToPartGroup = partUpLinkToPartGroup;
27
28     // do other initializations
29     initializePart ();
30 }

```

Some `finalize*()` () methods exist.

13.30 *asString() and *fromString() functions

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

```

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

```

```

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

```

```

1 std::string msrPlacementKindAsString (
2     msrPlacementKind placementKind)
3 {
4     std::string result;
5
6     switch (placementKind) {
7         case msrPlacementKind::kPlacement_UNKNOWN_:
8             result = "noPlacement";
9             break;
10        case msrPlacementKind::kPlacementAbove:
11            result = "placementAbove";
12            break;
13        case msrPlacementKind::kPlacementBelow:
14            result = "placementBelow";
15            break;
16    } // switch
17
18    return result;
19 }

```

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

13.31 `translate*()` methods and `convert*()` functions

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

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

```

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

```

```

1 S_msrScore msr2msrTranslator::translateMsrToMsr (
2     const S_msrScore& theMsrScore)
3 {
4     #ifdef MF_SANITY_CHECKS_ARE_ENABLED
5         // sanity check
6         mfAssert (
7             __FILE__, __LINE__,
8             theMsrScore != nullptr,
9             "theMsrScore is null");
10    #endif // MF_SANITY_CHECKS_ARE_ENABLED
11
12    // the MSR score we're visiting
13    fVisitedMsrScore = theMsrScore;
14
15    // create the resulting MSR score
16    fResultingNewMsrScore =
17        msrScore::create (
18            K_MF_INPUT_LINE_UNKNOWN_,
19            "msrScore::create()");
20
21    // create a msrScore browser
22    msrBrowser<msrScore> browser (this);
23
24    // set the parts browsing order
25    theMsrScore->
26        setStavesBrowsingOrderKind (
27            msrStavesBrowsingOrderKind::kStavesBrowsingOrderHarmoniesFiguredBassesRegulars);
28
29    // browse the visited MSR score
30    browser.browse (*fVisitedMsrScore);
31
32    // forget about the visited MSR score
33    fVisitedMsrScore = nullptr;
34
35    return fResultingNewMsrScore;
36 }

```

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

```

1 S_msrScore translateMsrToMsr (
2     S_msrScore originalMsrScore,
3     const S_msrOahGroup& msrOpts,
4     S_msr2msrOahGroup msr2msrOpts,

```

```

5   mfPassIDKind          passIDKind,
6   const std::string&    passDescription)
7 {
8   // ... ..
9
10  // the msr2msrTranslator
11  msr2msrTranslator
12  translator;
13
14  // build the resulting MSR score
15  S_msrScore
16  resultingNewMsrScore =
17    translator.translateMsrToMsr (
18      originalMsrScore);
19
20  // ... ..
21 }

```

13.32 context arguments

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

```

1 void msrMeasureRepeat::displayMeasureRepeat (
2   int    inputLineNumber,
3   const std::string& context)
4 {
5   gLog <<
6   std::endl <<
7   "*****>> MeasureRepeat " <<
8   ", measureRepeatMeasuresNumber: '" <<
9   fMeasureRepeatMeasuresNumber <<
10  ", measureRepeatSlashesNumber: '" <<
11  fMeasureRepeatSlashesNumber <<
12  "', voice:" <<
13  std::endl <<
14  fUpLinkToMeasureRepeatToVoice->getVoiceName () <<
15  " (" << context << ")" <<
16  ", line " << inputLineNumber <<
17  " contains:" <<
18  std::endl;
19
20  ++gIndenter;
21  print (gLog);
22  --gIndenter;
23
24  gLog <<
25  " <*****" <<
26  std::endl << std::endl;

```

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

```

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

```

13.33 Sorting and compare*() methods

MusicFormats sometimes needs to sort some data structures:

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

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

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

An example is:

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

```

19     firstStaffNumber > secondStaffNumber;
20
21     return result;
22 }

```

13.34 Mutually dependent classes

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

std::endl;

In some cases, two classes have to know about each other, such as `msrPartGroup` and `msrPartGroupElement`. This is because part groups can be nested: a part group contains part group element, which can be staves as well as other part groups. This loop can be seen in red in figure 20.3 [The MSR classes hierarchy], page 199.

13.34.1 Pre-declarations

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

13.34.2 Simple mutual dependency using separate header files

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

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

```

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

```

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

```

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

```

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

```

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

```

```

23     */
24 };

```

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

```

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

```

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

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

```

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

```



```

24 private:
25
26 // private fields
27 // -----
28
29 // upLinks
30
31 S_msrPartGroup          fPartUpLinkToPartGroup;
32
33 // ... ..
34 };

```

This rather complex situation is depicted at the top of figure 20.3 [The MSR classes hierarchy], page 199.

13.34.3 More complex mutual dependencies

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

- a note can be standalone in a measure;
- a note can be part of:
 - a chord;
 - a triplet;
 - a grace notes group;
 - a double tremolo;
- a chord can be standalone in a measure;
- a chord can be part of:
 - a triplet;
 - a grace notes group;
- a triplet can be standalone in a measure;
- a triplet can be part of:
 - another triplet;
- a grace notes group is attached to:
 - a note;
- a double tremolo is standalone in a measure.

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

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

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

This leads to the following hierarchy:

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

This rather complex situation is depicted at the bottom of figure 20.3 [The MSR classes hierarchy], page 199.

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

13.35 Templates and functional programming usage

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

- some are used by the two-phase visitors pattern, see chapter 15 [The two-phase visitors pattern], page 102;
- some exist for enumeration types, such as:

```

1  template< typename T >
2  class EnumNonSeparators
3  {
4      public:
5
6          class Iterator
7          {
8              public:
9
10             Iterator (int value)
11                 : fIterationIndex (value)
12                 {}
13
14             T operator* (void) const
15                 { return (T) fIterationIndex; }
16
17             void operator++ (void)
18                 { ++fIterationIndex; }
19
20             Bool operator != (Iterator rhs)
21                 { return fIterationIndex != rhs.fIterationIndex; }
22
23             private:
24
25                 int fIterationIndex;
26             };
27 };
28
29 template< typename T >
30 typename EnumNonSeparators<T>::Iterator begin (EnumNonSeparators<T>)
31 {
32     return typename EnumNonSeparators<T>::Iterator (((int)T::NonSeparatorsFirst);
33 }
34
35 template< typename T >
36 typename EnumNonSeparators<T>::Iterator end (EnumNonSeparators<T>)
37 {
38     return typename EnumNonSeparators<T>::Iterator (((int)T::NonSeparatorsLast) + 1);
39 }

```

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

```

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

```

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

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

Part III

Maintenance tools

Chapter 14

Maintenance tools

They are located in `devtools/` and meant to facilitate the MusicFormats maintenance activities:

```
1 jacquesmenu@macstudio-1:/Volumes/JMI_Volume/JMI_Developpement/musicformats-git-dev/  
  devtools > ls -sal  
2 total 120  
3  0 drwxr-xr-x  12 jacquesmenu  staff    384 Feb 15 10:53 .  
4  0 drwxr-xr-x  31 jacquesmenu  staff    992 Feb 28 15:52 ..  
5  8 -rwxr-xr-x@  1 jacquesmenu  staff   1568 Jan 29 13:33 BuildMusicFormats.zsh  
6  8 -rwxr-xr-x@  1 jacquesmenu  staff   1662 Jan 29 13:33  
  CreateMusicFormatsDocumentationPDFs.zsh  
7 16 -rwxr-xr-x@  1 jacquesmenu  staff   4140 Jan 31 10:22 CreateMusicFormatsReleases.zsh  
8  8 -rwxr--r--@  1 jacquesmenu  staff    883 Feb 15 11:21 InstallLilyJazz.zsh  
9  8 -rwxr-xr-x@  1 jacquesmenu  staff   1231 Feb  6 13:58  
  LocateFilesContainingFirstAndNotSecond.zsh  
10 8 -rwxr-xr-x   1 jacquesmenu  staff   1251 Jan 29 13:33 SetMusicFormatsVersionDate.zsh  
11 8 -rwxr-xr-x   1 jacquesmenu  staff   1304 Jan 29 13:33 SetMusicFormatsVersionNumber.zsh  
12 8 -rwxr-xr-x   1 jacquesmenu  staff    436 Jan 29 13:33 ShowMusicFormatsVersion.zsh  
13 48 -rwxr-xr-x@  1 jacquesmenu  staff  22752 Feb 23 11:43 ZshDefinitionsForMusicFormats.zsh
```

14.1 Building the library

`devtools/BuildMusicFormats.zsh` executes `cmake`, whose configuration is in `build/`:

```
1 jacquesmenu@macstudio-1:/Volumes/JMI_Volume/JMI_Developpement/musicformats-git-dev/build >  
  ls -sal  
2 total 992  
3  0 drwxr-xr-x  13 jacquesmenu  staff    416 Feb 22 15:55 .  
4  0 drwxr-xr-x  31 jacquesmenu  staff    992 Feb 28 15:52 ..  
5 16 -rw-r--r--@  1 jacquesmenu  staff   6148 Feb 16 18:17 .DS_Store  
6 904 -rw-r--r--@  1 jacquesmenu  staff  405409 Mar  1 06:03 BuildMusicFormats.zsh.log  
7  8 -rw-r--r--   1 jacquesmenu  staff   1815 Jan 29 13:33 Building.md  
8 40 -rw-r--r--@  1 jacquesmenu  staff  18499 Feb  6 12:48 CMakeLists.txt  
9  8 -rw-r--r--   1 jacquesmenu  staff    292 Jan 29 13:33 MakePkg.bat  
10 16 -rw-r--r--   1 jacquesmenu  staff   6227 Jan 29 13:33 Makefile  
11  0 drwxr-xr-x   3 jacquesmenu  staff    96 Jan 29 13:33 android  
12  0 drwxr-xr-x  27 jacquesmenu  staff    864 Feb 28 15:49 bin  
13  0 drwxr-xr-x   3 jacquesmenu  staff    96 Jan 29 13:33 docker  
14  0 drwxr-xr-x   4 jacquesmenu  staff   128 Feb 28 15:49 lib  
15  0 drwxr-xr-x   8 jacquesmenu  staff   256 Feb 22 15:55 libdir
```

14.2 Building the documentation

`devtools/CreateMusicFormatsDocumentationPDFs.zsh` executes lives up to its name. The \LaTeX source files are in `doc/`:

```

1 jacquesmenu@macstudio-1:/Volumes/JMI_Volume/JMI_Developpement/musicformats-git-dev/
  documentation > ll
2 total 24
3  0 drwxr-xr-x   12 jacquesmenu  staff    384 Jan 29 13:33:19 2023 ./
4  0 drwxr-xr-x   31 jacquesmenu  staff    992 Feb 28 15:52:03 2023 ../
5 24 -rw-r--r--@   1 jacquesmenu  staff  10244 Feb 23 11:57:51 2023 .DS_Store
6  0 drwxr-xr-x   22 jacquesmenu  staff    704 Jan 29 13:33:14 2023 CommonLaTeXFiles/
7  0 drwxr-xr-x   32 jacquesmenu  staff   1024 Jan 29 13:33:14 2023 IntroductionToMusicXML/
8  0 drwxr-xr-x   47 jacquesmenu  staff   1504 Feb 17 11:12:27 2023 mfapiguide/
9  0 drwxr-xr-x    8 jacquesmenu  staff    256 Feb 17 11:12:49 2023 MusicFormatsFigures/
10 0 drwxr-xr-x  105 jacquesmenu  staff   3360 Mar  1 06:12:41 2023 mfmaintainanceguide/
11 0 drwxr-xr-x   54 jacquesmenu  staff   1728 Feb 17 11:13:30 2023 mfuserguide/
12 0 drwxr-xr-x    3 jacquesmenu  staff     96 Jan 29 13:33:14 2023 bulddoc/
13 0 drwxr-xr-x   45 jacquesmenu  staff   1440 Jan 29 13:33:14 2023 mfgraphics/mfgraphics
14 0 drwxr-xr-x    5 jacquesmenu  staff    160 Jan 29 13:33:14 2023 libmusicxml2Presentation
  /

```

When modifying a documentation's structure information, i.e. `\chapter`, `\section` and `\subsection` commands contents, `pdflatex` should be run three times to get the corresponding PDF.

14.3 Releases creation

`devtools/CreateMusicFormatsReleases.zsh` lives up to its name too.

14.4 Checking source files contents

`devtools/LocateFilesContainingFirstAndNotSecond.zsh` helps to find superfluous C++17 includes or other such checking tasks.

14.5 Handly ZSH functions and aliases

There are a couple of them in `devtools/ZshDefinitionsForMusicFormats.zsh`

Part IV

The two-phase visitors pattern

Chapter 15

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.

15.1 Basic mechanism

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

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

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

A visitor class should:

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

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

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

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

```

1 void msrWords::acceptIn (basevisitor* v)
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     if (gMsrOahGroup->getTraceMsrVisitors ()) {
5         std::stringstream ss;
6
7         ss <<
8             "% ==> msrWords::acceptIn ()";
9
10        gWaeHandler->waeTraceWithoutInputLocation (
11            __FILE__, __LINE__,
12            ss.str ());
13    }
14 #endif // MF_TRACE_IS_ENABLED
15
16    if (visitor<S_msrWords>*>
17        p =
18        dynamic_cast<visitor<S_msrWords>*> (v)) {
19        S_msrWords elem = this;
20
21 #ifdef MF_TRACE_IS_ENABLED
22     if (gMsrOahGroup->getTraceMsrVisitors ()) {
23         std::stringstream ss;
24
25         ss <<
26             "% ==> Launching msrWords::visitStart ()";
27
28         gWaeHandler->waeTraceWithoutInputLocation (
29             __FILE__, __LINE__,
30             ss.str ());
31     }
32 #endif // MF_TRACE_IS_ENABLED
33     p->visitStart (elem);
34 }
35 }

```

15.2 Browser template classes

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

```

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

```



```

21
22 protected:
23
24     basevisitor*   fVisitor;
25
26     virtual void enter (T& t) { t.acceptIn (fVisitor); }
27     virtual void leave (T& t) { t.acceptOut (fVisitor); }
28 };

```

15.3 A first example: counting notes in MusicXML data

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

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

Listing 15.1: `countnotes.cpp`

```

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

```

15.4 A more complex example

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

```

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

```

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

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

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

```

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

```

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

```

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

```

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

Listing 15.2: Visiting `<scaling />`

```

1  void mxsr2msrTranslator::visitStart ( S_millimeters& elt )
2  {
3      #ifdef MF_TRACE_IS_ENABLED
4          if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
5              std::stringstream ss;
6
7              ss <<
8                  "--> Start visiting S_millimeters" <<
9                  ", line " << elt->getInputStartLineNumber ();
10
11              gWaeHandler->waeTraceWithoutInputLocation (
12                  __FILE__, __LINE__,
13                  ss.str ());
14          }
15      #endif // MF_TRACE_IS_ENABLED
16
17      fCurrentMillimeters = (float)(*elt);
18  }
19
20  void mxsr2msrTranslator::visitStart ( S_tenths& elt )
21  {
22      #ifdef MF_TRACE_IS_ENABLED
23          if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
24              std::stringstream ss;
25
26              ss <<
27                  "--> Start visiting S_tenths" <<
28                  ", line " << elt->getInputStartLineNumber ();
29
30              gWaeHandler->waeTraceWithoutInputLocation (
31                  __FILE__, __LINE__,
32                  ss.str ());
33          }
34      #endif // MF_TRACE_IS_ENABLED
35
36      fCurrentTenths = (float)(*elt);
37  }
38
39  void mxsr2msrTranslator::visitEnd ( S_scaling& elt)
40  {
41      int inputLineNumber =
42          elt->getInputStartLineNumber ();
43
44      #ifdef MF_TRACE_IS_ENABLED
45          if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
46              std::stringstream ss;
47
48              ss <<
49                  "--> End visiting S_scaling" <<
50                  ", line " << inputLineNumber;
51
52              gWaeHandler->waeTraceWithoutInputLocation (
53                  __FILE__, __LINE__,
54                  ss.str ());
55          }
56      #endif // MF_TRACE_IS_ENABLED
57  }

```

```

58 // create a scaling
59 S_msrScaling
60     scaling =
61         msrScaling::create (
62             inputLineNumber,
63             fCurrentMillimeters,
64             fCurrentTenths);
65
66 #ifdef MF_TRACE_IS_ENABLED
67     if (gTraceOahGroup->getTraceGeometry ()) {
68         gLog <<
69             "There are " << fCurrentTenths <<
70             " tenths for " << fCurrentMillimeters;
71
72         gWaeHandler->waeTraceWithoutInputLocation (
73             __FILE__, __LINE__,
74             ss.str ());
75     }
76 #endif // MF_TRACE_IS_ENABLED
77
78 // set the MSR score's scaling
79 fMsrScore->
80     setScaling (scaling);
81 }

```

15.5 Data browsing order

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

Listing 15.3: `msrDoubleTremolo::browseData (basevisitor* v)`

```

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

```

Since this order is set in the `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.

15.6 Selectively inhibiting data browsing

15.6.1 Inhibiting data browsing in the code

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

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

```

1 // in <multiple-rest/>, the multi-measure rests are explicit,
2 // whereas LilyPond only needs the number of multi-measure rests
3 Bool                                fInhibitMultiMeasureRestsBrowsing;
4
5 void                                setInhibitMultiMeasureRestsBrowsing ()
6 {
7     fInhibitMultiMeasureRestsBrowsing = true;
8 }
9
10 Bool                                getInhibitMultiMeasureRestsBrowsing () const
11 {
12     return
13         fInhibitMultiMeasureRestsBrowsing;
14 };

```

Class `lpsr2lilypondTranslator` checks this setting:

```

1 void lpsr2lilypondTranslator::visitEnd (S_msrNote& elt)
2 {
3     // ... ..
4
5     if (fOnGoingMultiMeasureRests) {
6         switch (elt->getNoteKind ()) {
7             case msrNoteKind::kNoteRestInMeasure:
8                 // don't handle multi-measure rests, that's done in visitEnd (
9                 S_msrMultiMeasureRest&)
10                 if (elt->getNoteOccupiesAFullMeasure ()) {
11                     Bool inhibitMultiMeasureRestsBrowsing =
12                         fVisitedLpsrScore->
13                         getMsrScore ()->
14                         getInhibitMultiMeasureRestsBrowsing ();
15
16                     if (inhibitMultiMeasureRestsBrowsing) {
17                         #ifdef MF_TRACE_IS_ENABLED
18                             if (
19                                 gTraceOahGroup->getTraceNotes ()
20                                 ||
21                                 gTraceOahGroup->getTraceMultiMeasureRests ()
22                             ) {
23                                 gLog <<
24                                     "% ==> end visiting multi-measure rests is ignored" <<
25                                     std::endl;
26                             }
27                         #endif // MF_TRACE_IS_ENABLED
28
29                         #ifdef MF_TRACE_IS_ENABLED
30                             if (gTraceOahGroup->getTraceNotesDetails ()) {
31                                 gLog <<
32                                     "% ==> returning from visitEnd (S_msrNote&)" <<
33                                     std::endl;
34                             }
35                         #endif // MF_TRACE_IS_ENABLED
36
37                         noteIsToBeIgnored = true;
38                     }
39                 }
40                 break;
41             // ... ..
42         }
43     }
44 }

```

Another example is in the class `lpsr2lilypondTranslator` constructor:

```

1 lpsr2lilypondTranslator::lpsr2lilypondTranslator (
2     S_lpsrScore    lpsrScore,
3     const S_msrOahGroup& msrOpts,

```

```

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

```

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

```

1  void msrNote::browseData (basevisitor* v)
2  {
3    // browse the grace notes group before if any
4    if (fNoteGraceNotesGroupBefore) {
5      // fetch the score
6      S_msrScore
7      score =
8        fetchUpLinkToNoteToScore ();
9
10     if (score) {
11       Bool
12       inhibitGraceNotesGroupsBeforeBrowsing =
13         score->
14         getInhibitGraceNotesGroupsBeforeBrowsing ();
15
16       if (inhibitGraceNotesGroupsBeforeBrowsing) {
17 #ifdef MF_TRACE_IS_ENABLED
18         if (
19           gMsrOahGroup->getTraceMsrVisitors ()
20           ||
21           gTraceOahGroup->getTraceNotes ()
22           ||
23           gTraceOahGroup->getTraceGraceNotes ()
24         ) {
25           std::stringstream ss;
26
27           ss <<
28             "% ==> visiting grace notes groups 'before' is inhibited";
29
30           gWaeHandler->waeTraceWithoutInputLocation (
31             __FILE__, __LINE__,
32             ss.str ());
33         }
34 #endif // MF_TRACE_IS_ENABLED
35       }
36       else {
37         // browse the grace notes group before
38         msrBrowser<msrGraceNotesGroup> browser (v);
39         browser.browse (*fNoteGraceNotesGroupBefore);
40       }
41     }

```

```

42 }
43
44 // ... ..
45 }

```

15.6.2 Inhibiting data browsing by options

Choosing which elements to browse can be more selective:

```

1 void msrStaff::browseData (basevisitor* v)
2 {
3     // ... ..
4
5     if (fStaffAllVoicesList.size ()) {
6         for (const S_msrVoice& voice : fStaffAllVoicesList) {
7             // is this voice name in the ignore voices set?
8             Bool ignoreVoice (false);
9
10            std::string voiceName =
11                voice->
12                getVoiceName ();
13
14            const std::set<std::string>&
15                ignoreMsrVoicesSet =
16                gGlobalMsr2msr0ahGroup->
17                getIgnoreMsrVoicesSet ();
18
19            // ... ..
20
21            if (ignoreMsrVoicesSet.size ()) {
22                ignoreVoice =
23                    mfStringIsInStringSet (
24                        voiceName,
25                        ignoreMsrVoicesSet);
26            }
27
28            if (ignoreVoice) {
29                #ifdef MF_TRACE_IS_ENABLED // JMI
30                    if (gTrace0ahGroup->getTraceVoices ()) {
31                        std::stringstream ss;
32
33                        ss <<
34                            "Ignoring voice \"" <<
35                            voiceName <<
36                            "\"";
37
38                        gWaeHandler->waeTraceWithoutInputLocation (
39                            __FILE__, __LINE__,
40                            ss.str ());
41                    }
42                #endif // MF_TRACE_IS_ENABLED
43            }
44
45            else {
46                msrBrowser<msrVoice> browser (v);
47                browser.browse (*voice);
48            }
49        } // for
50    }
51
52    // ... ..
53 }

```

15.7 Adapting visitors to data browsing order with booleans

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

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

```

1  <defaults>
2    <scaling>
3      <millimeters>7.3</millimeters>
4      <tenths>40</tenths>
5    </scaling>
6    <page-layout>
7      <page-height>1534</page-height>
8      <page-width>1151</page-width>
9      <page-margins type="both">
10       <left-margin>54.7945</left-margin>
11       <right-margin>54.7945</right-margin>
12       <top-margin>27.3973</top-margin>
13       <bottom-margin>27.3973</bottom-margin>
14     </page-margins>
15   </page-layout>
16   <system-layout>
17     <system-margins>
18       <left-margin>15</left-margin>
19       <right-margin>0</right-margin>
20     </system-margins>
21     <system-distance>92.5</system-distance>
22     <top-system-distance>27.5</top-system-distance>
23   </system-layout>
24
25   // ... ..
26
27   <part id="P1">
28     <measure number="1">
29       <print>
30         <system-layout>
31           <system-margins>
32             <left-margin>75.625</left-margin>
33             <right-margin>0</right-margin>
34           </system-margins>
35           <top-system-distance>410.9375</top-system-distance>
36         </system-layout>
37         <staff-layout>
38           <?DoletSibelius JustifyAllStaves=false?>
39           <?DoletSibelius ExtraSpacesAbove=3?>
40         </staff-layout>
41         <measure-layout>
42           <measure-distance>20</measure-distance>
43         </measure-layout>
44       </print>

```

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

It is assigned in:

```

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

```

```

9 {
10 // ... ..
11
12 fOnGoingPrintLayout = false;
13 }

```

and checked for example in:

```

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

```

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

```

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

```


15.8 Adapting visitors to data browsing order with stacks

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

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

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

```

1      std::list<S_msrTuplet>      fOnGoingTupletsStack;

1 void lpsr2lilypondTranslator::visitStart (S_msrTuplet& elt)
2 {
3     // ... ..
4
5     if (fOnGoingTupletsStack.size ()) {
6         // elt is a nested tuplet
7
8         S_msrTuplet
9             containingTuplet =
10             fOnGoingTupletsStack.front ();
11
12         // unapply containing tuplet factor,
13         // i.e 3/2 inside 5/4 becomes 15/8 in MusicXML...
14         elt->
15             unapplySoundingFactorToTupletMembers (
16                 containingTuplet->
17                     getTupletFactor ());
18     }
19
20     // ... ..
21
22     // push the tuplet on the tuplets stack
23     fOnGoingTupletsStack (elt);
24
25     // ... ..
26 }
27
28 void lpsr2lilypondTranslator::visitEnd (S_msrTuplet& elt)
29 {
30     // ... ..
31
32     // pop the tuplet from the tuplets stack
33     fOnGoingTupletsStack ();
34
35     // ... ..
36 }

```

15.9 Avoiding the visiting pattern by cascading

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

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

```

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

```

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

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

The time-oriented representation of scores in MSR is also printed by cascading through `printSlices ()` methods, see chapter 21 [MSR time-oriented representation], page 218.

Part V

MusicFormats components

Chapter 16

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/](#).

16.1 Versions numbers

The basic data structure is class `mfcVersionNumber`:

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

```

39     std::string          fPreRelease;
40 };

```

16.2 Versions descriptions

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

```

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

```

16.3 Versions histories

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

```

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

```

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

```

1 S_mfcVersionDescr mfcVersionsHistory::fetchMostRecentVersion () const
2 {
3     #ifdef MF_SANITY_CHECKS_ARE_ENABLED
4         // sanity check
5         mfAssert (
6             __FILE__, __LINE__,
7             fVersionsList.size () > 0,
8             "fVersionsList is empty");
9     #endif // MF_SANITY_CHECKS_ARE_ENABLED
10
11     return fVersionsList.back ();
12 }

```

16.4 Components descriptions

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

```

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

```

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

```

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

```

```

35 // protected services
36 // -----
37
38 virtual void          printOwnHistory (std::ostream& os) const;
39
40 protected:
41
42 // protected fields
43 // -----
44
45 std::string          fComponentName;
46
47 mfcComponentKind     fComponentKind;
48
49 S_mfcVersionsHistory fVersionsHistory;
50 };

```

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

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

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

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

16.5 Multi-components

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

```

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

```

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

```

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

```

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

```

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

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

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

```

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

```

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

```

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

```



```

14      Start of sequential versions numbering
15
16  MSR
17      v0.9.5 (October 6 2021):
18      Start of sequential versions numbering
19
20      v0.9.51 (October 14 2021):
21      Refined MSR names and summary display options
22
23  BSR
24      v0.9.5 (October 6 2021):
25      Start of sequential versions numbering
26
27  Passes history:
28  mxsr2msr
29      v0.9.5 (October 6 2021):
30      Start of sequential versions numbering
31
32  msr2msr
33      v0.9.5 (October 6 2021):
34      Start of sequential versions numbering
35
36  msr2bsr
37      v0.9.5 (October 6 2021):
38      Start of sequential versions numbering
39
40  bsr2bsr
41      v0.9.5 (October 6 2021):
42      Start of sequential versions numbering
43
44  bsr2braille
45      v0.9.5 (October 6 2021):
46      Start of sequential versions numbering

```

16.6 Versions history creation

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

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

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

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

16.6.1 Representations and passes components creation

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

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

```

1 S_mfcRepresentationComponent EXP createMsrRepresentationComponent ()
2 {
3     static S_mfcRepresentationComponent pRepresentationComponent;
4
5     // protect library against multiple initializations
6     if (! pRepresentationComponent) {
7
8 #ifdef MF_TRACE_IS_ENABLED
9     if (gEarlyOptions.getEarlyTraceComponents ()) {
10         std::stringstream ss;
11
12         ss <<
13             "Initializing MSR format component";
14
15         gWaeHandler->waeTraceWithoutInputLocation (
16             __FILE__, __LINE__,
17             ss.str ());
18     }
19 #endif // MF_TRACE_IS_ENABLED
20
21     // create the format component
22     pRepresentationComponent =
23         mfcRepresentationComponent::create (
24             "MSR");
25
26     // populate it
27     pRepresentationComponent->
28         appendVersionDescrToComponent (
29             mfcVersionDescr::create (
30                 mfcVersionNumber::createFromString ("0.9.50"),
31                 "October 6, 2021",
32                 std::list<std::string> {
33                     "Start of sequential versions numbering"
34                 }
35             ));
36
37     pRepresentationComponent->
38         appendVersionDescrToComponent (
39             mfcVersionDescr::create (
40                 mfcVersionNumber::createFromString ("0.9.51"), // JMI
41                 "October 14, 2021",
42                 std::list<std::string> {
43                     "Refined MSR names and summary display options"
44                 }
45             ));
46     }
47
48     return pRepresentationComponent;
49 }

```

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

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

```

```

67
68     pConverterComponent->
69         appendPassToMultiComponent (
70             createMsr2bsrComponent ());
71
72     pConverterComponent->
73         appendPassToMultiComponent (
74             createBsr2bsrComponent ());
75
76     pConverterComponent->
77         appendPassToMultiComponent (
78             createBsr2brailleComponent ());
79 }
80
81 return pConverterComponent;
82 }

```

16.6.3 MusicFormats library component creation

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

```

1 S_mfcLibraryComponent EXP createLibraryComponent ()
2 {
3     static S_mfcLibraryComponent pLibraryComponent;
4
5     // protect library against multiple initializations
6     if (! pLibraryComponent) {
7
8     #ifdef MF_TRACE_IS_ENABLED
9         if (gEarlyOptions.getTraceEarlyOptions ()) {
10             std::stringstream ss;
11
12             ss <<
13                 "Creating the  MFC library component";
14
15             gWaeHandler->waeTraceWithoutInputLocation (
16                 __FILE__, __LINE__,
17                 ss.str ());
18         }
19     #endif // MF_TRACE_IS_ENABLED
20
21     // create the library's history
22     pLibraryComponent =
23         mfcLibraryComponent::create (
24             "musicformats",
25             mfcMultiComponentEntropicityKind::kComponentEntropicityNo,
26             mfcMultiComponentUsedFromTheCLIKind::kComponentUsedFromTheCLISYes); // JMI ???
27
28     // populate the library's own history
29     pLibraryComponent->
30         appendVersionDescrToComponent (
31             mfcVersionDescr::create (
32                 mfcVersionNumber::createFromString ("0.9.50"),
33                 "October 6, 2021",
34                 std::list<std::string> {
35                     "Start of sequential versions numbering"
36                 }
37             ));
38
39     pLibraryComponent->
40         appendVersionDescrToComponent (
41             mfcVersionDescr::create (
42                 mfcVersionNumber::createFromString ("0.9.51"),
43                 "October 12, 2021",

```

```

44         std::list<std::string> {
45             "Adding a version number to the MusicFormats library",
46             "Fixed trace OAH issue in the musicxml2* converters)"
47         }
48     });
49
50     pLibraryComponent->
51     appendVersionDescrToComponent (
52         mfcVersionDescr::create (
53             mfcVersionNumber::createFromString ("0.9.52"),
54             "October 12, 2021",
55             std::list<std::string> {
56                 "Added MusicFormats library versions history to '-hist, -history'"
57             }
58         ));
59
60     pLibraryComponent->
61     appendVersionDescrToComponent (
62         mfcVersionDescr::create (
63             mfcVersionNumber::createFromString ("0.9.53"),
64             "October 22, 2021",
65             std::list<std::string> {
66                 "Replaced bool by class Bool in variables and fields",
67                 "Created MFC (MusicFormats components)"
68             }
69         ));
70
71     pLibraryComponent->
72     appendVersionDescrToComponent (
73         mfcVersionDescr::create (
74             mfcVersionNumber::createFromString ("0.9.54"),
75             "November 6, 2021",
76             std::list<std::string> {
77                 "Replaced std::cout and std::cerr by gOutput and gLog respectively in the CLI
78 samples",
79                 "Finalized components numbering (MFC)"
80             }
81         ));
82
83     // populate the library's components history
84     pLibraryComponent->
85     appendRepresentationToMultiComponent (
86         createMsrRepresentationComponent ());
87     pLibraryComponent->
88     appendRepresentationToMultiComponent (
89         createLpsrRepresentationComponent ());
90     pLibraryComponent->
91     appendRepresentationToMultiComponent (
92         createBsrRepresentationComponent ());
93     pLibraryComponent->
94     appendRepresentationToMultiComponent (
95         createMxsrRepresentationComponent ());
96
97     pLibraryComponent->
98     appendPassToMultiComponent (
99         createMsr2msrComponent ());
100
101     pLibraryComponent->
102     appendPassToMultiComponent (
103         createMsr2lpsrComponent ());
104     pLibraryComponent->
105     appendPassToMultiComponent (
106         createLpsr2lilypondComponent ());
107
108     pLibraryComponent->
109     appendPassToMultiComponent (
110         createMsr2bsrComponent ());

```

```

110     pLibraryComponent->
111         appendPassToMultiComponent (
112             createBsr2bsrComponent ());
113     pLibraryComponent->
114         appendPassToMultiComponent (
115             createBsr2brailleComponent ());
116
117     pLibraryComponent->
118         appendPassToMultiComponent (
119             createMsr2mxsrComponent ());
120
121     pLibraryComponent->
122         appendPassToMultiComponent (
123             createMxsr2musicxmlComponent ());
124
125     pLibraryComponent->
126         appendPassToMultiComponent (
127             createMxsr2guidoComponent ());
128 }
129
130 return pLibraryComponent;
131 }

```

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

16.6.4 Version and history options handling

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

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

```

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

```

```

32 // compound versions
33 S_mfcMultiComponent    fHandlerMultiComponent;
34 };

```

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

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

```

1 xml2xmlInsiderHandler::xml2xmlInsiderHandler (
2     const std::string& serviceName,
3     std::string handlerHeader)
4     : oahInsiderHandler (
5         serviceName,
6         handlerHeader,
7     R"(
8         Welcome to the MusicXML to MusicXML converter
9         delivered as part of the MusicFormats library.
10
11         --- https://github.com/jacques-menu/musicformats ---
12     )",
13     R"(
14     Usage: xml2xml [[option]* [MusicXMLFile|-] [[option]*
15     )")
16     {
17         // ... ..
18
19         // initialize the multi-component
20         initializeHandlerMultiComponent ();
21
22         // ... ..
23     }

```

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

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

```

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

```

16.7 Accessing versions in regular handlers

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

```

1 class EXP oahRegularHandler : public oahHandler
2 /*
3     A regular OAH handler relies on the existence of so-called 'insider' handler,
4     that contains all the options values gathered from the user,
5     grouped according to the internal representations and passes used.
6
7     The variables containing the values of the options selected by the user
8     are actually held by the insider handler.
9 */
10 {
11     // ... ..
12

```

```

13  protected:
14
15      // protected initialization
16      // -----
17
18      // ... ..
19
20      void                initializeHandlerMultiComponent () override
21      {
22          fHandlerMultiComponent =
23              fInsiderHandler->
24                  getHandlerMultiComponent ();
25      }
26
27      // ... ..
28  };

```

16.8 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.

16.8.1 Current version numbers in options

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

```

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

```

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

```

1  jacquesmenu@macmini: ~/musicformats-git-dev/musicxml > xml2gmn -history
2  Command line version of musicxml2guido converter v0.9.51 (October 12 2021)
3
4  Own history:
5      v0.9.5 (October 6 2021):
6          Start of sequential versions numbering
7
8      v0.9.51 (October 12 2021):
9          Fixed trace OAH issue
10
11 Representations history:
12     MXSR
13         v0.9.5 (October 6 2021):

```



```

14      Start of sequential versions numbering
15
16  MSR
17    v0.9.5 (October 6 2021):
18      Start of sequential versions numbering
19
20    v0.9.51 (October 14 2021):
21      Refined MSR names and summary display options
22
23  Passes history:
24    mxsr2msr
25      v0.9.5 (October 6 2021):
26        Start of sequential versions numbering
27
28    msr2msr
29      v0.9.5 (October 6 2021):
30        Start of sequential versions numbering
31
32    msr2mxsr
33      v0.9.5 (October 6 2021):
34        Start of sequential versions numbering
35
36    mxsr2guido
37      v0.9.5 (October 6 2021):
38        Start of sequential versions numbering

```

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

```

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

```

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

```

1  void oahVersionAtom::applyValueLessAtom (std::ostream& os)
2  {
3      #ifdef MF_TRACE_IS_ENABLED
4          if (gEarlyOptions.getTraceEarlyOptions ()) {
5              std::stringstream ss;
6
7              ss <<
8                  "=="> option ' " << fetchNames () << " ' is a oahVersionAtom";
9
10             gWaeHandler->waeTraceWithoutInputLocation (
11                 __FILE__, __LINE__,
12                 ss.str ());
13         }

```

```

14 #endif // MF_TRACE_IS_ENABLED
15
16 int saveIndent = gIndenter.getIndent ();
17
18 gIndenter.resetToZero ();
19
20 printVersion (os);
21
22 gIndenter.setIndent (saveIndent);
23
24 fSelected = true;
25 }

```

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

```

1 void oahVersionAtom::printVersion (std::ostream& os) const
2 {
3     // get the handler version
4     S_mfcMultiComponent
5     handlerMultiComponent =
6     fetchAtomUpLinkToHandler ()->
7     getHandlerMultiComponent ();
8
9 #ifdef MF_SANITY_CHECKS_ARE_ENABLED
10    // sanity check
11    mfAssert (
12        __FILE__, __LINE__,
13        handlerMultiComponent != nullptr,
14        "handlerMultiComponent is null");
15 #endif // MF_SANITY_CHECKS_ARE_ENABLED
16
17    handlerMultiComponent->
18    printVersion (os);
19 }

```

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

16.8.2 Current version numbers in formats

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

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/musicxml > xml2ly --lilypond-generation-infos
   basic/HelloWorld.xml
2 \version "2.24.0"
3
4 % Pick your choice from the next two lines as needed
5 %myBreak = { \break }
6 myBreak = {}
7
8 % Pick your choice from the next two lines as needed
9 %myPageBreak = { \pageBreak }
10 myPageBreak = {}
11
12 % Generated by xml2ly v0.9.51 (October 12 2021)
13 % on Thursday 2021-11-11 @ 11:15:56 CET
14 % from "basic/HelloWorld.xml"
15
16 % ... ..

```

Class `lpsrScore` contains an MFC component field:

```

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

```

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

```

1 lpsrScore::lpsrScore (
2     int                inputLineNumber,
3     const S_msrScore&   theMsrScore,
4     const S_mfcMultiComponent& multiComponent)
5     : lpsrElement (inputLineNumber)
6 {
7     // ... ..
8
9     fMsrScore = theMsrScore;
10
11    fMultiComponent = multiComponent;
12
13    // should the initial comments about the service and the options used
14    // be generated?
15    if (gGlobalLpsr2lilypondOahGroup->getXml2lyInfos ()) {
16        // create the 'input source name and translation date' comment
17        {
18            std::stringstream ss;
19
20            ss <<
21                "Generated by " <<
22                gOahOahGroup->getOahOahGroupServiceName () <<
23                ' ' <<
24                fMultiComponent->
25                    mostRecentVersionNumberAndDateAsString () <<
26                std::endl <<
27
28                "% on " <<
29                gServiceRunData->getTranslationDateFull () <<
30                std::endl <<
31
32                "% from ";
33
34            if (gServiceRunData->getInputSourceName () == "-") {
35                ss << "standard input";
36            }
37            else {
38                ss << "\"" << gServiceRunData->getInputSourceName () << "\"";
39            }
40
41            fInputSourceNameComment =
42                lpsrComment::create (
43                    inputLineNumber,
44                    ss.str (),

```

```

45         lpsrCommentGapAfterwardsKind::kCommentGapAfterwardsNo);
46     }
47
48     // ... ..
49 }
50
51 // ... ..
52 }

```

16.8.3 Current version numbers in passes

Another case is that of the generation of MusicXML output:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/musicxml > xml2xml -musicxml-generation-infos
  basic/HelloWorld.xml
2 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
3 <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
4   "http://www.musicxml.org/dtds/partwise.dtd">
5 <score-partwise version="3.1">
6   <!--
7   =====
8   Created by xml2xml v0.9.5 (October 6 2021)
9   on Thursday 2021-11-11 @ 11:04:06 CET
10  from basic/HelloWorld.xml
11  =====
12  -->
13   <work>
14     <work-number/>
15     <work-title>Hello World!</work-title>
16   </work>
17   <movement-number/>
18   <movement-title/>
19   <identification>
20     <encoding>
21       <software>xml2xml v0.9.5 (October 6 2021), https://github.com/jacques-menu/
musicformats</software>
22       <encoding-date>2021-11-10</encoding-date>
23     </encoding>
24     <miscellaneous>
25       <miscellaneous-field name="description"/>
26     </miscellaneous>
27   </identification>
28
29   <!-- ... .. -->

```

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

```

1 void msr2mxsrTranslator::visitStart (S_msrScore& elt)
2 {
3   // ... ..
4
5   // get the pass component
6   S_mfcPassComponent
7     passComponent =
8     createMsr2mxsrComponent ();
9
10  // get the pass component current version number and date
11  std::string
12    passComponentMostRecentVersionNumberAndDateAsString =
13    passComponent->
14    mostRecentVersionNumberAndDateAsString ();
15
16  // create the initial creation comment

```

```

17  std::stringstream ss;
18  ss <<
19      std::endl <<
20      "===== " <<
21      std::endl <<
22      "Created by " <<
23      gOahOahGroup->getOahOahGroupServiceName () <<
24      ' ' <<
25      passComponentMostRecentVersionNumberAndDateAsString <<
26      std::endl <<
27
28      "on " <<
29      gServiceRunData->getTranslationDateFull () <<
30      std::endl <<
31
32      "from " <<
33      gServiceRunData->getInputSourceName () <<
34      std::endl <<
35
36      "===== " <<
37      std::endl;
38
39  // append the initial creation comment to the score part wise element
40  fResultingMusicxmlelement->push (createMxmlelement (kComment, ss.str ()));
41
42  // create a software element
43  Sxmlelement
44      softwareElement =
45      createMxmlelement (
46          k_software,
47          gOahOahGroup->getOahOahGroupServiceName ()
48          + ' ',
49          + passComponentMostRecentVersionNumberAndDateAsString +
50          ", https://github.com/jacques-menu/musicformats");
51
52  // append it to the identification encoding
53  appendToScoreIdentificationEncoding (softwareElement);
54
55  // ... ..
56  }

```

Part VI

Options and help

Chapter 17

Options and help (OAH)

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

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

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

17.1 OAH basics

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

17.2 Features

- the values of the various options can be displayed with the option `-display-options-values`, `-dov` option;
- partial help can be obtained, i.e. help about any group, subgroup or atom, showing the path in the hierarchy down to the corresponding option;
- there are various sub-classes of class `oahAtom` such as class `oahIntegerAtom`, class `oahBooleanAtom` and class `oahRationalAtom`, to control options values of common types;
- class `oahThreeBooleansAtom`, for example, allows for three boolean settings to be controlled at once with a single option;
- class `oahAtomStoringAValue` describes options for which a value is supplied in the command line or in option vectors (API);
- a class such as class `lpsrPitchesLanguageAtom` is used to supply a 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`.

17.3 OAH classes inheritance

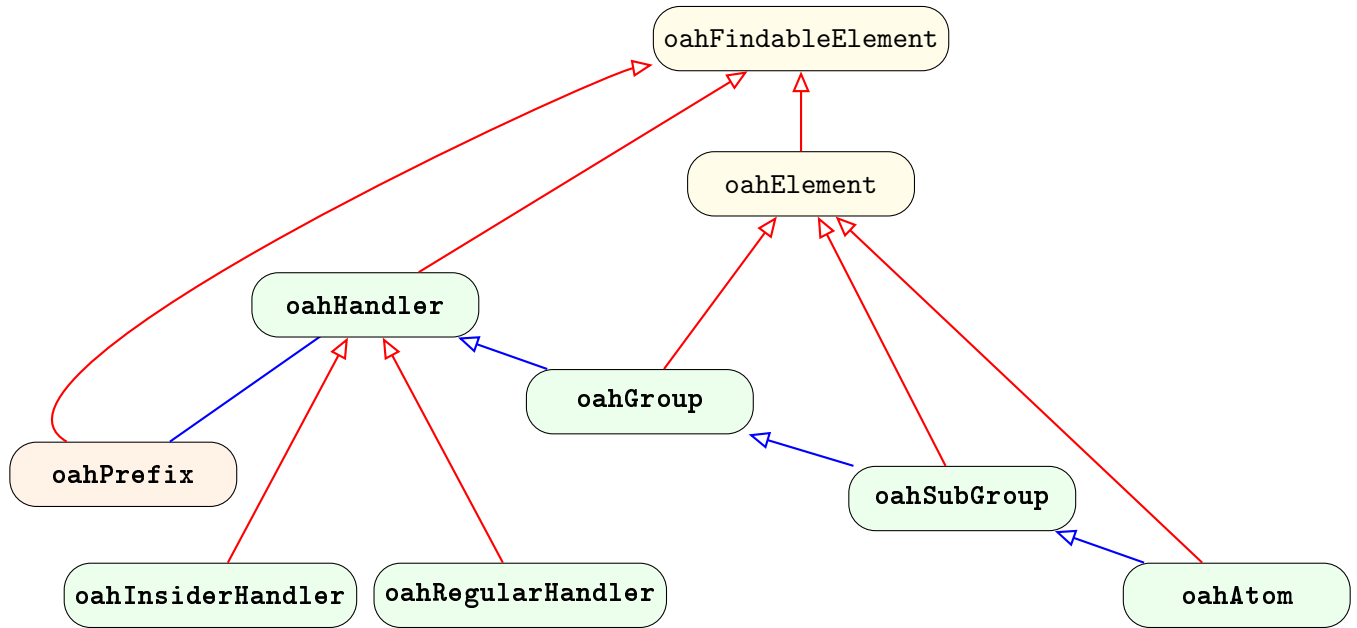
The picture at figure 17.1 [The OAH classes hierarchy], page 136, shows the hierarchy of the main OAH classes. The colors are used as follows:

The background colors are used as follows:

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

The arrows colors have the following meaning:

Figure 17.1: The OAH classes hierarchy



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

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

```

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

```

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

17.4 Value-fitted atoms

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

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

```

1 class EXP oahValueFittedAtom : 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                applyValueLessAtom (std::ostream& os) override;
15                         // reports an error
16
17     virtual void        applyAtomWithValue (
18                         const std::string& theString,
19                         std::ostream&      os) = 0;
20
21     virtual void        applyAtomWithDefaultValue (std::ostream& os);
22 };

```

The classes derived from `oahValueFittedAtom` are:

```

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

```

For example:

```

1 class EXP lilypondFixedOctaveEntryAtom : public oahValueFittedAtom
2 {
3     public:
4
5     // creation
6     // -----
7
8     static SMARTP<lilypondFixedOctaveEntryAtom> create (
9                         const std::string& longName,
10                        const std::string& shortName,
11                        const std::string& description,
12                        const std::string& valueSpecification,
13                        const std::string& variableName,
14                        msrOctaveEntryVariable&
15                        octaveEntryKindVariable,
16                        S_msrSemiTonesPitchAndOctave&
17                        lilypondFixedOctaveEntryVariable);
18

```

```

19 protected:
20
21 // constructors/destructor
22 // -----
23
24         lilypondFixedOctaveEntryAtom (
25             const std::string& longName,
26             const std::string& shortName,
27             const std::string& description,
28             const std::string& valueSpecification,
29             const std::string& variableName,
30             msrOctaveEntryVariable&
31                 octaveEntryKindVariable,
32             S_msrSemiTonesPitchAndOctave&
33                 lilypondFixedOctaveEntryVariable);
34
35 virtual         ~lilypondFixedOctaveEntryAtom ();
36
37 public:
38
39 // set and get
40 // -----
41
42 void             setFixedOctaveEntryVariableValue (
43                 const S_msrSemiTonesPitchAndOctave& value)
44                 { fSemiTonesPitchAndOctaveVariable = value; }
45
46 S_msrSemiTonesPitchAndOctave&
47         getFixedOctaveEntryVariableValue () const
48         { return fSemiTonesPitchAndOctaveVariable; }
49
50 // ... ..
51
52 private:
53
54 // private fields
55 // -----
56
57 msrOctaveEntryVariable&
58         fOctaveEntryKindVariable;
59
60 S_msrSemiTonesPitchAndOctave&
61         fSemiTonesPitchAndOctaveVariable;
62 };

```

This class is instantiated this way:

```

1 void lpsr2lilypondOahGroup::initializeNotesOptions ()
2 {
3     // ... ..
4
5     subGroup->
6         appendAtomToSubGroup (
7             lilypondFixedOctaveEntryAtom::create (
8                 "fixed", "",
9 R"(Use fixed octave entry reference PITCH_AND_OCTAVE in the generated LilyPond code.
10 PITCH_AND_OCTAVE is made of a diatonic pitch and
11 an optional sequence of commas or single quotes.
12 It should be placed between double quotes if it contains single quotes, such as:
13 -fixed "c''")",
14         "PITCH_AND_OCTAVE",
15         "fFixedOctaveEntrySemiTonesPitchAndOctave",
16         fOctaveEntryVariable,
17         fFixedOctaveEntrySemiTonesPitchAndOctave));
18
19     // ... ..

```

20 }

Applying such an atom with a value is done this way:

```

1 void lilypondFixedOctaveEntryAtom::applyAtomWithValue (
2     const std::string& theString,
3     std::ostream&      os)
4 {
5     #ifdef MF_TRACE_IS_ENABLED
6         if (gEarlyOptions.getTraceEarlyOptions ()) {
7             std::stringstream ss;
8
9             ss <<
10                "=> oahAtom is of type 'lilypondFixedOctaveEntryAtom'";
11
12             gWaeHandler->waeTraceWithoutInputLocation (
13                 __FILE__, __LINE__,
14                 ss.str ());
15         }
16     #endif // MF_TRACE_IS_ENABLED
17
18     // theString contains the fixed reference note
19
20     // set octave entry kind
21     fOctaveEntryKindVariable.setOctaveEntryKind (
22         msrOctaveEntryKind::kOctaveEntryFixed);
23
24     // set fixed octave reference value
25     setFixedOctaveEntryVariableValue (
26         msrSemiTonesPitchAndOctave::createFromString (
27             K_MF_INPUT_LINE_UNKNOWN_,
28             theString));
29
30     fSelected = true;
31 }

```

17.4.1 The oahAtomStoringAValue class

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

```

1 class EXP oahAtomStoringAValue : public oahValueFittedAtom
2 {
3     // ... ..
4
5     public:
6
7         // print
8         // -----
9
10        virtual void      printAtomWithVariableEssentials (
11                            std::ostream& os,
12                            int fieldWidth) const;
13        virtual void      printAtomWithVariableEssentials (
14                            std::ostream& os,
15                            int fieldWidth) const;
16
17        void              print (std::ostream& os) const override;
18        void              printFull (std::ostream& os) const override;
19
20        void              printHelp (std::ostream& os) const override;
21
22        virtual void      displayAtomWithVariableOptionsValues (
23                            std::ostream& os,

```

```

24         int          valueFieldWidth) const override;
25
26     protected:
27
28         // protected fields
29         // -----
30
31         std::string      fValueSpecification;
32
33         std::string      fVariableName;
34 };

```

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

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

```

1 void oahIntegerAtom::setIntegerVariable (int value)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gEarlyOptions.getTraceEarlyOptions ()) {
5             std::stringstream ss;
6
7             ss <<
8                 "Setting option '" <<
9                 fetchNames () <<
10                 "' integer variable to '" <<
11                 value <<
12                 "'";
13
14             gWaeHandler->waeTraceWithoutInputLocation (
15                 __FILE__, __LINE__,
16                 ss.str ());
17         }
18     #endif // MF_TRACE_IS_ENABLED
19
20     fIntegerVariable = value;
21
22     fSelected = true;
23 }

```

17.4.2 The `oahBooleanAtom` special case

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

```

1 // -----
2 class EXP oahBooleanAtom : public oahAtom
3 {
4     /*
5     an atom controlling a Bool variable variableName,
6     but expecting no value to be supplied:
7     the variable is false initially,
8     and is set to true by the mere occurrence of the atom
9     */
10
11     // ... ..
12
13     protected:
14

```

```

15 // protected fields
16 // -----
17
18 std::string          fVariableName;
19 Bool&                fBooleanVariable;
20 };

```

17.4.3 Checking whether an option has been selected

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

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

```

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

```

The minimum LilyPond version number is 2.24.0. Another can be selected with the `-lilypond-version`, `-lpv` option:

```

1 jacquesmenu@macmini > xml2ly -find lilypond-version
2 1 occurrence of string "lilypond-version" has been found:
3 1:
4 -lilypond-version, -lpv
5 Set the Lilypond '\version' to STRING in the Lilypond code.
6 The default is '2.24.0'.

```

17.4.4 The `oahAtomStoringAValue` sub-classes

The classes derived from `oahAtomStoringAValue` are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grh 'public oahAtomStoringAValue '
2 oah/oahHarmoniesExtraOah.h:class EXP extraDisplayAllHarmoniesStructuresAtom : public
   oahAtomStoringAValue
3 oah/oahHarmoniesExtraOah.h:class EXP extraDisplayAllHarmoniesContentsAtom : public
   oahAtomStoringAValue
4 oah/oahHarmoniesExtraOah.h:class EXP extraDisplayHarmonyDetailsAtom : public
   oahAtomStoringAValue
5 oah/oahHarmoniesExtraOah.h:class EXP extraDisplayHarmonyAnalysisAtom : public
   oahAtomStoringAValue

```

```

6 oah/oahAtomsCollection.h:class EXP oahIntegerAtom : public oahAtomStoringAValue
7 oah/oahAtomsCollection.h:class EXP oahFloatAtom : public oahAtomStoringAValue
8 oah/oahAtomsCollection.h:class EXP oahStringAtom : public oahAtomStoringAValue
9 oah/oahAtomsCollection.h:class EXP oahRationalAtom : public oahAtomStoringAValue
10 oah/oahAtomsCollection.h:class EXP oahNaturalNumbersSetElementAtom : public
    oahAtomStoringAValue
11 oah/oahAtomsCollection.h:class EXP oahColorRGBAtom : public oahAtomStoringAValue
12 oah/oahAtomsCollection.h:class EXP oahIntSetElementAtom : public oahAtomStoringAValue
13 oah/oahAtomsCollection.h:class EXP oahStringSetElementAtom : public oahAtomStoringAValue
14 oah/oahAtomsCollection.h:class EXP oahStringToIntMapElementAtom : public
    oahAtomStoringAValue
15 oah/oahAtomsCollection.h:class EXP oahStringAndIntegerAtom : public oahAtomStoringAValue
16 oah/oahAtomsCollection.h:class EXP oahStringAndTwoIntegersAtom : public
    oahAtomStoringAValue
17 oah/oahAtomsCollection.h:class EXP oahLengthUnitKindAtom : public oahAtomStoringAValue
18 oah/oahAtomsCollection.h:class EXP oahLengthAtom : public oahAtomStoringAValue
19 oah/oahAtomsCollection.h:class EXP oahMidiTempoAtom : public oahAtomStoringAValue
20 formatsgeneration/brailleGeneration/brailleGenerationOah.h:class EXP brailleOutputKindAtom
    : public oahAtomStoringAValue
21 formatsgeneration/brailleGeneration/brailleGenerationOah.h:class EXP brailleUTFKindAtom :
    public oahAtomStoringAValue
22 formatsgeneration/brailleGeneration/brailleGenerationOah.h:class EXP
    brailleByteOrderingKindAtom : public oahAtomStoringAValue
23 formatsgeneration/msrGeneration/msrGenerationBasicTypes.h:class EXP
    msrGenerationAPIKindAtom : public oahAtomStoringAValue
24 formatsgeneration/multiGeneration/mfMultiGenerationOah.h:class EXP
    mfMultiGenerationOutputKindAtom : public oahAtomStoringAValue
25 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondScoreOutputKindAtom : public
    oahAtomStoringAValue
26 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondTransposePartNameAtom : public
    oahAtomStoringAValue
27 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondTransposePartIDAtom : public
    oahAtomStoringAValue
28 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondAccidentalStyleKindAtom : public
    oahAtomStoringAValue
29 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondChordsDisplayAtom : public
    oahAtomStoringAValue
30 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondLyricsNotesDurationsKindAtom :
    public oahAtomStoringAValue
31 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP
    lilypondDynamicsTextSpannersStyleKindAtom : public oahAtomStoringAValue
32 passes/lpsr2lilypond/lpsr2lilypondOah.h:class EXP lilypondBreakPageAfterMeasureNumberAtom
    : public oahAtomStoringAValue
33 passes/msr2lpsr/msr2lpsrOah.h:class EXP msr2lpsrScoreOutputKindAtom : public
    oahAtomStoringAValue
34 passes/msr2msr/msr2msrOah.h:class EXP msrIgnorePartAtom : public oahAtomStoringAValue
35 passes/msr2msr/msr2msrOah.h:class EXP msrKeepPartAtom : public oahAtomStoringAValue
36 passes/mxsr2msr/mxsr2msrOah.h:class EXP msrReplaceClefAtom : public oahAtomStoringAValue
37 formats/bsr/bsrOah.h:class EXP bsrFacSimileKindAtom : public oahAtomStoringAValue
38 formats/bsr/bsrOah.h:class EXP bsrTextsLanguageAtom : public oahAtomStoringAValue
39 formats/lpsr/lpsrOah.h:class EXP lpsrPitchesLanguageAtom : public oahAtomStoringAValue
40 formats/lpsr/lpsrOah.h:class EXP lpsrChordsLanguageAtom : public oahAtomStoringAValue
41 formats/lpsr/lpsrOah.h:class EXP lpsrTransposeAtom : public oahAtomStoringAValue
42 formats/msdl/msdlInputOah.h:class EXP msdlKeywordsLanguageAtom : public
    oahAtomStoringAValue
43 formats/msdl/msdlInputOah.h:class EXP msdlCommentsTypeAtom : public oahAtomStoringAValue
44 formats/msdl/msdlInputOah.h:class EXP msdlUserLanguageAtom : public oahAtomStoringAValue
45 formats/msdl/msdlInputOah.h:class EXP msdlPitchesLanguageAtom : public
    oahAtomStoringAValue
46 formats/msr/msrOah.h:class EXP msrPitchesLanguageAtom : public oahAtomStoringAValue
47 formats/msr/msrOah.h:class EXP msrRenamePartAtom : public oahAtomStoringAValue

```

17.5 Pure help atoms

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

17.5.1 Pure help atoms without a value

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

```

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

```

The actual pure help atoms without a value are:

```

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

```

17.5.2 Pure help value-fitted atoms

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

```

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

```

The actual pure help value-fitted atoms are:

```

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

```


17.6 Options implicitly storing a value

There are options in multiGeneration to select the generated output :

```

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

```

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

Purely virtual class oahAtomImplicitlySettingAVariable is the base class for this:

```

1 class EXP oahAtomImplicitlySettingAVariable : public oahAtom
2 /*
3  a purely virtual common ancestor for all atom classes
4  that store a value in a variable
5  without taking it from argv or an oahOptionsVector
6 */
7 {
8   // ... ..
9
10  protected:
11
12     // protected fields
13     // -----
14
15     std::string          fVariableName;
16 };

```

This is used by class mfMultiGenerationOutputKindAtom defined in [src/formatsgeneration/multiGeneration/multiGenerationOah.h/.cpp](#):

```

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

```

The value is stored in the variable in

constructor mfMultiGenerationOutputKindAtom::mfMultiGenerationOutputKindAtom ():

```

1 mfMultiGenerationOutputKindAtom::mfMultiGenerationOutputKindAtom (
2   const std::string&      longName,
3   const std::string&      shortName,
4   const std::string&      description,
5   const std::string&      variableName,

```

```

6  mfMultiGenerationOutputKind& mfMultiGenerationOutputKindVariable)
7  : oahAtomImplicitlySettingAVariable (
8      longName,
9      shortName,
10     description,
11     variableName,
12     oahElementValueKind::kElementValueLess),
13     fMultiGenerationOutputKindVariable ( // this is where the value is supplied
14     mfMultiGenerationOutputKindVariable)
15 {}

```

17.7 Options and help handling

- each option long name and non-empty short name must be unique in a given handler, to avoid ambiguities;
- an service main () calls method oahHandler::handleOptionsAndArgumentsFromArgcArgv (), in which:
- method oahHandler::handleOptionNameCommon () handles the option names;
- handleOptionValueOrArgument() and the arguments to the service.
- contracted forms are expanded in method oahHandler::handleOptionNameCommon () before the resulting, uncontracted options are handled;
- options handling works in two passes:
- the first one creates a list of class oahElementUse instances from argc/argv or an options and arguments;
- the second one traverses this list to apply the options that are used.
- the options are applied by virtual method applyElement (), virtual method applyAtomWithValue () and virtual method applyAtomWithDefaultValue ();
- method oahHandler::handleKnownArgvAtom () associates the value to the (preceding) field oahHandler::f if not null, or appends it to field oahHandler::fHandlerArgumentsVector to otherwise;
- fPendingArgvValueFittedAtom is used in argv contents handling to associate an option name with its value, which is the next element in argv.

17.8 Basic OAH types

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

```

1  jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class oahBasicTypes.h
2  // PRE-declarations for class mutual dependencies
3  class oahAtom;
4  class oahSubGroup;
5  class oahGroup;
6  class oahHandler;
7  enum class oahOptionsDefaultValuesStyle {
8  enum class oahHandlerUsedThruKind {
9  class oahPrefix;
10 class EXP oahPrefix : public smartable
11     a common ancestor for all atom classes,
12     this class contains only an uplink to the containing subgroup
13 class EXP oahAtom : public oahElement

```

```

14 /* this class is purely virtual
15 class EXP oahValueFittedAtom : public oahAtom
16     a common ancestor for all atom classes
17 /* this class is purely virtual
18     a common ancestor for all atom classes
19 class EXP oahAtomStoringAValue : public oahValueFittedAtom
20 /* this class is purely virtual
21 class EXP oahPureHelpValueLessAtom : public oahAtom
22 /* this class is purely virtual
23 class EXP oahPureHelpValueFittedAtom : public oahValueFittedAtom
24 /* this class is purely virtual
25 class EXP oahSubGroup : public oahElement
26 class EXP oahGroup : public oahElement
27 class EXP oahHandler : public smartable
28 /* this class is purely virtual
29 enum class oahOptionalValuesStyleKind {
30 class EXP oahValueLessAtom : public oahAtom
31 / * this class is purely virtual

```

17.9 Prefixes handling

17.10 argc/argv versus oahOptionsVector

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

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

```

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

```

Using an `oahOptionsVector` can be done for example:

- in Web sites;
- in the generators CLI services 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 services, an `oahOptionsVector` is instantiated and populated from `argc/argv` with `convertArgcArgvToOptions` defined in `src/oah/oahBasicTypes.h`:

```

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

```

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

```

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

```

17.11 Applying options

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

```

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

```

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

```

1 virtual void      applyAtomWithValue (
2             const std::string& theString,
3             std::ostream& os) = 0;
4
5 virtual void      applyAtomWithDefaultValue (std::ostream& os);

```

There are two methods for that:

```

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

```

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

17.12 A OAH atoms collection

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

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class      oahAtomsCollection.h
2 class EXP oahValueLessAtomAlias : public oahAtom
3 class EXP oahMacroAtom : public oahAtom
4 class EXP oahOptionsUsageAtom : public oahPureHelpValueLessAtom
5 class EXP oahHelpAtom : public oahPureHelpValueLessAtom
6 class EXP oahHelpSummaryAtom : public oahPureHelpValueLessAtom
7 class EXP oahAboutAtom : public oahPureHelpValueLessAtom
8 class EXP oahVersionAtom : public oahPureHelpValueLessAtom
9 class EXP oahContactAtom : public oahPureHelpValueLessAtom
10 class EXP oahBooleanAtom : public oahAtom
11 class EXP oahTwoBooleansAtom : public oahBooleanAtom
12 class EXP oahThreeBooleansAtom : public oahBooleanAtom
13 class EXP oahCombinedBooleansAtom : public oahAtom
14 class EXP oahCommonPrefixBooleansAtom : public oahAtom
15 class EXP oahIntegerAtom : public oahAtomStoringAValue
16 class EXP oahTwoIntegersAtom : public oahIntegerAtom

```

```

17 class EXP oahFloatAtom : public oahAtomStoringAValue
18 class EXP oahStringAtom : public oahAtomStoringAValue
19 class EXP oahFactorizedStringAtom : public oahAtom
20 class EXP oahDefaultedStringAtom : public oahStringAtom
21 class EXP oahRationalAtom : public oahAtomStoringAValue
22 class EXP oahNaturalNumbersSetElementAtom : public oahAtomStoringAValue
23 class EXP oahColorRGBAtom : public oahAtomStoringAValue
24 class EXP oahIntSetElementAtom : public oahAtomStoringAValue
25 class EXP oahStringSetElementAtom : public oahAtomStoringAValue
26 class EXP oahStringToIntMapElementAtom : public oahAtomStoringAValue
27 class EXP oahStringAndIntegerAtom : public oahAtomStoringAValue
28 class EXP oahStringAndTwoIntegersAtom : public oahAtomStoringAValue
29 class EXP oahLengthUnitKindAtom : public oahAtomStoringAValue
30 class EXP oahLengthAtom : public oahAtomStoringAValue
31 class EXP oahMidiTempoAtom : public oahAtomStoringAValue
32 class EXP oahOptionNameHelpAtom : public oahDefaultedStringAtom
33 class EXP oahQueryOptionNameAtom : public oahPureHelpValueFittedAtom
34 class EXP oahFindStringAtom : public oahPureHelpValueFittedAtom

```

See chapter 39 [The OAH atoms collection], page 283, for more details.

17.13 An option and help example

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

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

```

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

```

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

```

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

```

And that's it.

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

```

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

```

17.14 Options and help introspection

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

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

```

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

```

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

```

1 jacquesmenu@macmini > xml2ly -find grace
2 5 occurrences of string "grace" have been found:
3     1:
4         -hgraces-group, -help-grace-notes-group
5
6     2:
7         -hgraces, -help-grace-notes
8
9     3:
10        -slashagn, -slash-all-grace-notes
11        Add a slash to all grace notes
12
13     4:
14        -sluragn, -slur-all-grace-notes
15        Add a slur to all grace notes
16
17     5:
18        -beamagn, -beam-all-grace-notes
19        Add a beam to all grace notes

```

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

17.15 Insider versus regular handlers

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

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

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

class `oahRegularHandler` relies on the corresponding insider handler:

```

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

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

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

```

1  void xml2xmlRegularHandler::createInformationsRegularGroup ()
2  {
3      // group
4
5      S_oahGroup
6      group =
7          oahGroup::create (
8              "Informations group",
9              "help-informations-group", "hinfos-group",
10             "",
11             oahElementVisibilityKind::kElementVisibilityWhole);
12     appendGroupToRegularHandler (group);
13
14     // subgroup
15
16     S_oahSubGroup
17     subGroup =
18         oahSubGroup::create (
19             "Informations",
20             "help-informations", "hinfos",
21             "",
22             oahElementVisibilityKind::kElementVisibilityWhole,
23             group);
24     group->
25         appendSubGroupToGroup (subGroup);
26
27     // atoms from the insider handler
28
29     registerAtomInRegularSubgroup ("about", subGroup);
30     registerAtomInRegularSubgroup ("version", subGroup);
31     registerAtomInRegularSubgroup ("version-full", subGroup);
32     registerAtomInRegularSubgroup ("history", subGroup);
33     registerAtomInRegularSubgroup ("mf-version", subGroup);
34     registerAtomInRegularSubgroup ("mf-history", subGroup);
35     registerAtomInRegularSubgroup ("contact", subGroup);
36 }
```

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

```

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

```

17.16 Deciphering the options and arguments

17.16.1 Options and arguments multi-pass analysis

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

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

```

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

```

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

```

1  EXP mfMusicformatsErrorKind musicxmlfile2musicxml (
2  const char*          fileName,
3  mfOptionsAndArguments& handlerOptionsAndArguments,
4  std::ostream&         out,
5  std::ostream&         err)
6  {
7  SXMLFile

```



```

8      sxmlfile =
9          createXMLFileFromFile (
10              fileName,
11              mfPassIDKind::kMfPassID_1,
12              "Create an MXSR from a MusicXML file");
13
14      if (sxmlfile) {
15          return
16              xmlFile2musicxmlWithOptionsAndArguments (
17                  sxmlfile,
18                  handlerOptionsAndArguments,
19                  out,
20                  err);
21      }
22
23      return mfMusicformatsErrorKind::kMusicformatsErrorInvalidFile;
24  }

```

This is done using a two-pass scheme:

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

class `oahHandler` contains:

```

1      // elements uses
2      std::list<S_oahElementUse> fElementUsesList;
3
4      // atoms waiting for a value
5      S_oahValueFittedAtom
6          fPendingArgvValueFittedAtom;
7      std::string
9          fNameUsedForPendingArgvValueFittedAtom;

```

17.16.2 Pure help runs

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

```

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

```

17.16.3 Applying options

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

```

1      oahElementHelpOnlyKind oahHandler::applyOptionsFromElementUsesList ()

```

The heart of it is:

```

1 oahElementHelpOnlyKind oahHandler::applyOptionsFromElementUsesList ()
2 {
3     // ... ..
4
5     // the heart of it
6     if (
7         // group?
8         S_oahGroup
9         group =
10            dynamic_cast<oahGroup*>(&(*elementUsed))
11    ) {
12        group->
13            applyValueLessAtom (
14                gOutput);
15    }
16
17    else if (
18        // subgroup?
19        S_oahSubGroup
20        subGroup =
21            dynamic_cast<oahSubGroup*>(&(*elementUsed))
22    ) {
23        subGroup->
24            applyValueLessAtom (
25                gOutput);
26    }
27
28    else {
29        // this is an atom
30
31        S_oahAtom
32        atom =
33            dynamic_cast<oahAtom*>(&(*elementUsed));
34
35        oahElementValueKind
36        atomValueKind =
37            atom->
38                getElementValueKind ();
39
40        if (
41            // value-fitted atom?
42            S_oahValueFittedAtom
43            valueFittedAtom =
44                dynamic_cast<oahValueFittedAtom*>(&(*elementUsed))
45        ) {
46            switch (atomValueKind) {
47                case oahElementValueKind::kElementValueLess:
48                    {
49                        std::stringstream ss;
50
51                        ss <<
52                            "Atom with value " <<
53                            valueFittedAtom->fetchNamesBetweenQuotes () <<
54                            " has been registered as without value";
55
56                        oahInternalError (ss.str ());
57                    }
58                    break;
59
60                case oahElementValueKind::kElementValueImplicit:
61                %
62                    valueFittedAtom->
63                    applyAtomWithDefaultValue (
64                        gOutput);
65                    break;
66
67                case oahElementValueKind::kElementValueFitted:

```

```

67         if (valueUsed.size ()) {
68             valueFittedAtom->
69                 applyAtomWithValue (
70                     valueUsed,
71                     gOutput);
72         }
73         else {
74             std::stringstream ss;
75
76             ss <<
77                 "Value-fitted atom " <<
78                 valueFittedAtom->fetchNamesBetweenQuotes () <<
79                 " needs a non-empty value";
80
81             oahInternalError (ss.str ());
82         }
83         break;
84
85     case oahElementValueKind::kElementValueDefaulted:
86         if (valueUsed.size ()) {
87             valueFittedAtom->
88                 applyAtomWithValue (
89                     valueUsed,
90                     gOutput);
91         }
92         else {
93             valueFittedAtom->
94                 applyAtomWithDefaultValue (
95                     gOutput);
96         }
97         break;
98     } // switch
99 }
100
101     else {
102 #ifdef MF_TRACE_IS_ENABLED
103         if (gEarlyOptions.getTraceEarlyOptions ()) {
104             gLog <<
105                 "**** Handling value-less atom:" <<
106                 std::endl;
107
108             ++gIndenter;
109
110             gLog <<
111                 atom <<
112                 std::endl;
113
114             --gIndenter;
115         }
116 #endif // MF_TRACE_IS_ENABLED
117
118         atom->
119             applyValueLessAtom (
120                 gOutput);
121     }
122 }
123
124 // has a help-only been applied?
125 switch (elementUsed->getElementHelpOnlyKind ()) {
126     case oahElementHelpOnlyKind::kElementHelpOnlyYes:
127         // a help option has been applied
128         this->
129             setOahHandlerFoundAHelpOption (
130                 elementUsed->
131                     fetchNamesBetweenQuotes ());
132         break;
133     case oahElementHelpOnlyKind::kElementHelpOnlyNo:

```

```

134         break;
135     } // switch
136 }
137
138 else {
139     std::stringstream ss;
140
141     ss <<
142         "Element from the from the atom uses list for \"" <<
143         nameUsed <<
144         "\" is null";
145
146     oahInternalError (ss.str ());
147
148     // ... ..
149 }

```

17.16.4 Early handling of some options

Debugging OAH needs the trace handling trace 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 std::string K_INSIDER_OPTION_LONG_NAME;
5 EXP extern const std::string K_INSIDER_OPTION_SHORT_NAME;
6
7 // ... ..
8
9 class EXP oahEarlyOptions
10 {
11     // ... ..
12
13     public:
14
15         // set and get
16         // -----
17
18         void                setEarlyInsiderOption ();
19         Bool                getEarlyInsiderOption () const
20                             { return fEarlyInsiderOption; }
21
22     // ... ..
23
24     private:
25
26         // fields
27         // -----
28

```

```

29     Bool                                fEarlyInsiderOption;
30
31     // ... ..
32 }

```

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

```

1 // ... ..
2
3 const std::string K_INSIDER_OPTION_LONG_NAME  = "insider";
4 const std::string K_INSIDER_OPTION_SHORT_NAME = "ins";
5
6 void oahEarlyOptions::setEarlyInsiderOption ()
7 {
8     if (fTraceEarlyOptions) {
9         std::stringstream ss;
10
11         ss <<
12             "Setting fEarlyInsiderOption" <<
13             std::endl;
14     }
15
16     fEarlyInsiderOption = true;
17 }
18
19 // ... ..

```

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

```

1 void oahEarlyOptions::applyEarlyOptionIfRelevant (
2     const std::string& argumentWithoutDashToBeUsed,
3     const std::string& optionValue)
4 {
5     // this is OAH handling pass 1
6     if (
7         isEarlyOptionRecognized (
8             argumentWithoutDashToBeUsed, K_INSIDER_OPTION_LONG_NAME)
9         ||
10        isEarlyOptionRecognized (
11            argumentWithoutDashToBeUsed, K_INSIDER_OPTION_SHORT_NAME)
12    ) {
13        setEarlyInsiderOption ();
14    }
15
16    // ... ..
17 }

```

17.17 Implementing the -find option

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

```

1 class EXP oahFindableElement : public smartable
2 {
3     public:
4
5         // creation from MusicXML
6         // -----
7
8         /* this class is purely virtual
9            static SMARTP<oahFindableElement> create ();
10        */

```

```

11
12 // ... ..
13
14 public:
15
16 // public services
17 // -----
18
19 virtual Bool      findStringInFindableElement (
20                     const std::string&        lowerCaseString,
21                     std::list<S_oahFindStringMatch>& foundMatchesList,
22                     std::ostream&              os) const = 0;
23
24 public:
25
26 // print
27 // -----
28
29 virtual std::string  asString () const = 0;
30
31 virtual void          print (std::ostream& os) const = 0;
32
33 virtual const std::string  containingFindableElementAsString () const = 0;
34
35 private:
36
37 // private fields
38 // -----
39 };

```

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

```

1 class  oahFindStringMatch : public smartable
2 {
3     // ... ..
4
5     protected:
6
7         // protected fields
8         // -----
9
10        std::string      fFoundString;
11        std::string      fContainingFindableElementInfo;
12 };

```

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

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

```

1 Bool oahHandler::findStringInFindableElement (
2     const std::string&        lowerCaseString,
3     std::list<S_oahFindStringMatch>& foundMatchesList,
4     std::ostream&              os) const
5 {
6     Bool result;
7
8     // .. ..
9
10    // does this handler's header match?
11    Bool headerMatches =
12        mfStringToLowerCase (fHandlerHeader).find (lowerCaseString) != std::string::npos;
13
14    // does this handler's description match?

```

```

15 Bool descriptionMatches =
16     mfStringToLowerCase (fHandlerDescription).find (lowerCaseString) != std::string::npos;
17
18 // does this handler's usage match?
19 Bool usageMatches =
20     mfStringToLowerCase (fHandlerUsage).find (lowerCaseString) != std::string::npos;
21
22 if (headerMatches || descriptionMatches || usageMatches) {
23     std::stringstream ss;
24
25     ss <<
26         fHandlerHeader <<
27         ' ' <<
28         fHandlerDescription <<
29         ' ' <<
30         fHandlerUsage;
31
32     // append the match to foundStringsList
33     foundMatchesList.push_back (
34         oahFindStringMatch::create (
35             ss.str (),
36             containingFindableElementAsString ());
37
38     result = true;
39 }
40
41 // do this handler's prefixes match?
42 if (fHandlerPrefixesMap.size ()) {
43     ++gIndenter;
44
45     for (
46         std::map<std::string, S_oahPrefix>::const_iterator i =
47             fHandlerPrefixesMap.begin ();
48         i != fHandlerPrefixesMap.end ();
49         ++i
50     ) {
51         S_oahPrefix
52             prefix = (*i).second;
53
54         // does the prefix match?
55         prefix->
56             findStringInFindableElement (
57                 lowerCaseString,
58                 foundMatchesList,
59                 os);
60     } // for
61
62     --gIndenter;
63 }
64
65 // do this handler's groups match?
66 if (fHandlerGroupsList.size ()) {
67     ++gIndenter;
68
69     for (S_oahGroup group : fHandlerGroupsList) {
70         group->
71             findStringInGroup (
72                 lowerCaseString,
73                 foundMatchesList,
74                 os);
75     } // for
76
77     --gIndenter;
78 }
79
80 return result;
81 }

```

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

17.18 Checking options consistency

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

For example:

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

17.19 Adding new options

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

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

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

17.19.1 'Representations' vs. 'passes' options

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

For example, the trace of `<backup/>` and `<forward/>` is used by both `mxsr2msrOah` and `msr2mxsrOah`. The corresponding options are thus placed in `src/formats/mxsr/mxsrOah.h/.cpp`:

17.19.2 Using an existing OAH atom class

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

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

```

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

```

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

```

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

```

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

```

1 void oahOahGroup::printOahOahValues (int valueFieldWidth)
2 {
3     gLog <<
4         "The basic options are:" <<
5         std::endl;
6
7     // ... ..
8
9     // options and help display
10    // -----
11
12    gLog << std::left <<
13        std::setw (valueFieldWidth) << "Options trace and display:" <<
14        std::endl;
15
16    ++gIndenter;

```

```

17
18 gLog << std::left <<
19     std::setw (valueFieldWidth) << "fReverseNamesDisplayOrder" << ": " <<
20     fReverseNamesDisplayOrder <<
21     std::endl <<
22
23 // ... ..

```

- then tests of the use of option `-reverse-names-display-order`, `-rndo` were added in `src/oah/oahElements.h` such as in method `oahElement::fetchNames ()`:

```

1 std::string oahElement::fetchNames () const
2 {
3     std::stringstream ss;
4
5     if (
6         fShortName.size ()
7         &&
8         fLongName.size ()
9     ) {
10        if (gOahOahGroup->getReverseNamesDisplayOrder ()) {
11            ss <<
12                '- ' << fShortName <<
13                ", " <<
14                '- ' << fLongName;
15        }
16        else {
17            ss <<
18                '- ' << fLongName <<
19                ", " <<
20                '- ' << fShortName;
21        }
22    }
23
24    else {
25        if (fShortName.size ()) {
26            ss <<
27                '- ' << fShortName;
28        }
29        if (fLongName.size ()) {
30            ss <<
31                '- ' << fLongName;
32        }
33    }
34
35    return ss.str ();
36 }

```

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

```

1 void msdl2brailleRegularHandler::createOahRegularGroup ()
2 {
3     // ... ..
4
5     registerAtomInRegularSubgroup ("reverse-names-display-order", subGroup);
6
7     // ... ..
8 }

```

17.19.3 Creating a new OAH atom class without a value

When class `oahHistoryAtom` was added to OAH, the first thing has been to add a `printHistory ()` in class `mfcMultiComponent` in `src/mfutilities/mfcBasicTypes.h`:

```

1 class mfcMultiComponent : public smartable
2 {
3     // ... ..
4
5     public:
6
7         // print
8         // -----
9
10        void                print (std::ostream& os) const;
11        void                printHistory (std::ostream& os) const;
12
13        // ... ..
14 };

```

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

```

1 // -----
2 class EXP oahHistoryAtom : public oahPureHelpValueLessAtom
3 {
4     // ... ..
5
6     public:
7
8         // public services
9         // -----
10
11        void                applyValueLessAtom (std::ostream& os) override;
12
13        public:
14
15        // visitors
16        // -----
17
18        void                acceptIn (basevisitor* v) override;
19        void                acceptOut (basevisitor* v) override;
20
21        void                browseData (basevisitor* v) override;
22
23        // print
24        // -----
25
26        void                print (std::ostream& os) const override;
27        void                printHistory (std::ostream& os) const;
28 };
29

```

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

```

1 void oahHistoryAtom::applyValueLessAtom (std::ostream& os)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gEarlyOptions.getTraceEarlyOptions ()) {
5             std::stringstream ss;
6
7             ss <<
8                 "==" option ' ' << fetchNames () << " is a oahHistoryAtom";
9
10            gWaeHandler->waeTraceWithoutInputLocation (
11                __FILE__, __LINE__,
12                ss.str ());

```

```

13 }
14 #endif // MF_TRACE_IS_ENABLED
15
16 int saveIndent = gIndenter.getIndent ();
17
18 gIndenter.resetToZero ();
19
20 printHistory (os);
21
22 gIndenter.setIndent (saveIndent);
23
24 fSelected = true;
25 }

```

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

```

1 void oahHistoryAtom::printHistory (std::ostream& os) const
2 {
3     // get the handler history
4     S_mfcMultiComponent
5     handlerMultiComponent =
6     fetchAtomUpLinkToHandler ()->
7     getHandlerMultiComponent ();
8
9 #ifdef MF_SANITY_CHECKS_ARE_ENABLED
10    // sanity check
11    mfAssert (
12        __FILE__, __LINE__,
13        handlerMultiComponent != nullptr,
14        "handlerMultiComponent is null");
15 #endif // MF_SANITY_CHECKS_ARE_ENABLED
16
17    handlerMultiComponent->
18    printHistory (os);
19 }

```

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

```

1 // history
2
3 subGroup->
4     appendAtomToSubGroup (
5         oahHistoryAtom::create (
6             "hist", "history",
7             regex_replace (
8 R"(Display EXECUTABLE_NAME's history.)",
9             std::regex ("EXECUTABLE_NAME"),
10             serviceName),
11             serviceName));

```

And the new option long name `version` has been added to all regular OAH handlers that already contained the option `-version`, `-v`, such as in method `xml2lyRegularHandler::createInformationsRegularGroup ()`, alongside the existing option `-version`, `-v`:

```

1 registerAtomInRegularSubgroup ("version", subGroup);
2 registerAtomInRegularSubgroup ("version-full", subGroup);
3 registerAtomInRegularSubgroup ("history", subGroup);
4 registerAtomInRegularSubgroup ("mf-version", subGroup);
5 registerAtomInRegularSubgroup ("mf-history", subGroup);

```

17.19.4 Creating a new OAH value-fitted atom class

Let's look at how class `oahLengthAtom` is implemented.

class `msrLength` is defined in `src/formats/msr/msrBasicTypes.h/.cpp`:

```

1 class EXP msrLength : public smartable
2 {
3     // ... ..
4
5     // public services
6     // -----
7
8     Bool                operator== (const msrLength& other) const
9     {
10         // JMI convert to same length unit kind before comparing
11
12         return
13             fLengthUnitKind == other.fLengthUnitKind
14             &&
15             fLengthValue == other.fLengthValue;
16     }
17
18     Bool                operator!= (const msrLength& other) const
19     { return ! ((*this) == other); }
20
21     void                convertToLengthUnit (
22         msrLengthUnitKind lengthUnitKind);
23
24     // ... ..
25
26 private:
27     // private fields
28     // -----
29
30     msrLengthUnitKind    fLengthUnitKind;
31     float                fLengthValue;
32 };

```

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

```

1 enum class msrLengthUnitKind {
2     kUnitInch, kUnitCentimeter, kUnitMillimeter
3 };

```

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

```

1 class EXP oahLengthAtom : public oahAtomStoringAValue
2 {
3     /*
4     an atom controlling a length variable
5     */
6
7     // ... ..
8
9     public:
10
11     // set and get
12     // -----
13
14     void                setLengthVariable (
15         msrLength value)
16     {
17         fLengthVariable = value;
18     }

```

```

19
20     msrLength          getLengthVariable () const
21                         { return fLengthVariable; }
22
23 public:
24
25     // public services
26     // -----
27
28     void                applyAtomWithValue (
29                         const std::string& theString,
30                         std::ostream&      os) override;
31
32     // ... ..
33
34 public:
35
36     // print
37     // -----
38
39     // ... ..
40
41     void                displayAtomWithVariableOptionsValues (
42                         std::ostream& os,
43                         int          valueFieldWidth) const override;
44
45 private:
46
47     // private fields
48     // -----
49
50     msrLength&          fLengthVariable;
51 };

```

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

```

1 void oahLengthAtom::applyAtomWithValue (
2     const std::string& theString,
3     std::ostream&      os)
4 {
5     // ... ..
6
7     std::regex e (regularExpression);
8     std::smatch sm;
9
10    regex_match (theString, sm, e);
11
12    unsigned int smSize = sm.size ();
13
14    // ... ..
15
16    if (smSize == 4) {
17        // leave the low level details to the STL...
18        float floatValue;
19        {
20            std::stringstream ss;
21            // concatenate the integer and decimal parts
22            ss << sm [ 1 ] << sm [ 2 ];
23            ss >> floatValue;
24        }
25
26        std::string lengthUnitName = sm [ 3 ];
27
28        // is lengthUnitName known in the length unit names map?
29        std::map<std::string, msrLengthUnitKind>::const_iterator

```

```

30     it =
31         gGlobalMsrLengthUnitKindsMap.find (
32             lengthUnitName);
33
34     if (it == gGlobalMsrLengthUnitKindsMap.end ()) {
35         // no, length unit name is unknown in the map
36
37         std::stringstream ss;
38
39         ss <<
40             "length unit name \"" << lengthUnitName <<
41             "\" is unknown" <<
42             std::endl <<
43             "The " <<
44             gGlobalMsrLengthUnitKindsMap.size () <<
45             " known length unit names are:" <<
46             std::endl;
47
48         ++gIndenter;
49
50         ss <<
51             availableMsrLengthUnitKinds (K_MF_NAMES_LIST_MAX_LENGTH);
52
53         --gIndenter;
54
55         oahError (ss.str ());
56     }
57
58     setLengthVariable (
59         msrLength (
60             (*it).second,
61             floatValue));
62 }
63
64 else {
65     std::stringstream ss;
66
67     ss <<
68         "length value \"" << theString <<
69         "\" for option '" << fetchNames () <<
70         "' is ill-formed";
71
72     oahError (ss.str ());
73 }
74
75 fSelected = true;
76 }

```

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

```

1 void oahLengthAtom::displayAtomWithVariableOptionsValues (
2     std::ostream& os,
3     int valueFieldWidth) const
4 {
5     os << std::left <<
6         std::setw (valueFieldWidth) <<
7         fVariableName <<
8         " : " <<
9         fLengthVariable.asString ();
10    switch (fEarlyOptionKind) {
11        case oahEarlyOptionKind::kEarlyOptionNo:
12            break;
13        case oahEarlyOptionKind::kEarlyOptionYes:
14            os <<
15                ", early";

```

```

16     break;
17 } // switch
18 if (fSelected) {
19     os << ", selected";
20 }
21 os << std::endl;
22 }

```

Then an option to set the LilyPond paper height can be added to the relevant OAH options group in method `lpsrOahGroup::initializeLpsrPaperOptions ()` in `src/formats/lpsr//lpsrOah.cpp` by:

```

1 // paper height
2
3 fPaperHeight.setLengthUnitKind (msrLengthUnitKind::kUnitMillimeter);
4 fPaperHeight.setLengthValue (297);
5
6 fPaperHeightAtom =
7     oahLengthAtom::create (
8         "paper-height", "",
9         R"(Set the LilyPond 'paper-height' paper variable to HEIGHT in the LilyPond code.
10        HEIGHT should be a positive floating point or integer number,
11        immediately followed by a unit name, i.e. 'in', 'mm' or 'cm'.
12        By default, LilyPond uses 297 mm (A4 format).)",
13         "HEIGHT",
14         "paperHeight",
15         fPaperHeight);
16
17 subGroup->
18     appendAtomToSubGroup (
19         fPaperHeightAtom);

```

17.20 Extra options

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

```

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

```

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

```

1 // create the extra OAH group
2 appendGroupToHandler (
3     createGlobalHarmoniesExtraOahGroup ());

```

Macro `MF_HARMONIES_EXTRA_IS_ENABLED` is defined or not in `src/oah/mfEnableHarmoniesExtraSetting.h`:

```

1 // comment the following definition if no extra options are wanted
2
3 #ifndef MF_HARMONIES_EXTRA_IS_ENABLED
4     #define MF_HARMONIES_EXTRA_IS_ENABLED

```


17.21 man pages generation

MusicFormats can create man pages for its command line services by browsing their OAH hierarchy. This has not been finalized yet.

17.22 Specific global OAH groups

Some informations need to be available globally in the MusicFormats library, such as the conversion date and command line. They are grouped in `src/oah/generalOah.h/.cpp`:

```

1 class EXP generalOahGroup : public oahGroup
2 {
3     // ... ..
4
5     private:
6
7         // translation date
8         // -----
9
10        std::string          fTranslationDateFull;
11        std::string          fTranslationDateYYYYMMDD;
12
13        // warning and error handling
14        // -----
15
16        Bool                 fQuiet;
17
18        Bool                 fDontShowErrors;
19        Bool                 fDontQuitOnErrors;
20
21        Bool                 fDisplaySourceCodePositions;
22
23        // CPU usage
24        // -----
25
26        Bool                 fDisplayCPUUsage;
27 };

```

There are also harmonies-specific options grouped in `src/oah/harmoniesExtraOah.h/.cpp`. They are available as icing on the cake independently of any conversion activity:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class oahHarmoniesExtraOah.h
2 class EXP extraDisplayAllHarmoniesStructuresAtom : public oahAtomStoringAValue
3 class EXP extraDisplayAllHarmoniesContentsAtom : public oahAtomStoringAValue
4 class EXP extraDisplayHarmonyDetailsAtom : public oahAtomStoringAValue
5 class EXP extraDisplayHarmonyAnalysisAtom : public oahAtomStoringAValue
6 class EXP harmoniesExtraOahGroup : public oahGroup

```

```

1 class EXP harmoniesExtraOahGroup : public oahGroup
2 {
3     // ... ..
4
5     public:
6
7         // fields
8         // -----
9
10        std::string          fHarmoniesRootAsString;
11 };

```

17.23 Visiting OAH groups

As an internal representation, OAH can be browsed with the two-phase visitors. This is useful:

- to produce man pages automatically from the options available;
- to create the code that proposes the options to the user in a Web site, also automatically.

Part VII

Representations

Chapter 18

Representations general principles

18.1 Trees vs graphs, denormalization

In databases, denormalization means that some data is present in several places. This is usually done for speed, at the cost of making updates more complex, since no such place should be ignored in an update.

A music score can be represented as a tree of elements, but performing conversions of such representations needs shortcuts to be more practical. MusicFormats used the term `link` for that.

18.2 Denormalization

18.2.1 Descriptions sharing

MSR uses denormalization explicitly, with smart pointers to class instances being stored in other instances.

In particular, class `msrChord` contains elements that are actually detained by the notes it is composed of:

```
1  // articulations
2  std::list<S_msrArticulation>
3      fChordArticulations;
4
5  // spanners
6  std::list<S_msrSpanner>
7      fChordSpanners;
8
9  // single tremolo
10 S_msrSingleTremolo    fChordSingleTremolo;
```

This is to avoid having to browse the chord's components to obtain the corresponding information each time it is needed.

All such denormalization is done in MSR internally: the code using MSR does not have to denormalize itself. It can use whichever occurrence of any given denormalized data safely, though.

18.2.2 Multiple voices

Another aspect to account for is that of dynamics, lyrics, harmonies and figured bass.

LilyPond supplies specific support to have them outside of notes, chords and other sound-producing score elements. This provides flexibility when combining a score's staves and voices in various ways depending on the needs .

MusicFormats has explicit voice kinds for this, declared in `src/formats/msr/msrVoices.h`:

```
1 enum class msrVoiceKind {
2     kVoiceKindRegular,
3     kVoiceKindDynamics,
4     kVoiceKindHarmonies, // for MusicXML <harmony/>, LilyPond ChordNames
5     kVoiceKindFiguredBass // for MusicXML <figured-bass/>, LilyPond FiguredBass
6 };
```

In MSR, for example, a harmony is both attached to a note in a regular voice and an element of a harmony voice:

```
1 std::list<S_msrHarmony>
2     fNoteHarmoniesList;
```

An `msrNote` instance will thus be browsed twice, when those two voices are. The ones attached to a note are browsed this way:

```
1 void msrNote::browseData (basevisitor* v)
2 {
3     // ... ..
4
5     // browse the harmonies if any
6     if (fNoteHarmoniesList.size ()) {
7         ++gIndenter;
8         for (S_msrHarmony harmony : fNoteHarmoniesList) {
9             // browse the harmony
10            msrBrowser<msrHarmony> browser (v);
11            browser.browse (*harmony);
12        } // for
13        --gIndenter;
14    }
15
16    // ... ..
17};
```

18.3 Newborn clones

The multi-pass structure of the converters build with `musicformat` leads to a question: should an existing description, such as that of a `barLine` or a `note`, be used as is, or should it be built again?

Depending of the kind of description, both possibilities are used:

- the description is used as is if it is *shallow*, i.e. it doesn't contain smart-pointers to data – it is *self-contained*;
- otherwise, a new description is built, sharing some smart-pointers fieds with the existing one if needed. This newborn clone is then populated when it is inserted in the representation being built.

For example, in `src/passes/msr2lpsr/`, the `S_msrBarLine` values found in the MSR data are used also in the LPSR data:

```

1 void msr2lpsrTranslator::visitStart (S_msrBarLine& elt)
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     int inputLineNumber =
5         elt->getInputStartLineNumber ();
6 #endif // MF_TRACE_IS_ENABLED
7
8 #ifdef MF_TRACE_IS_ENABLED
9     if (gMsrOahGroup->getTraceMsrVisitors ()) {
10         std::stringstream ss;
11
12         ss <<
13             "--> Start visiting msrBarLine" <<
14             ", line " << inputLineNumber;
15
16         gWaeHandler->waeTraceWithoutInputLocation (
17             __FILE__, __LINE__,
18             ss.str ());
19     }
20 #endif // MF_TRACE_IS_ENABLED
21
22     // ... ..
23
24     // append the barLine to the current voice clone
25     fCurrentVoiceClone->
26         appendBarLineToVoice (elt);
27 }

```

On the opposite, a new `S_msrVoice` description is built for use by LPSR: this is how the LilyPond #34 issue is circumvented, adding skip notes where needed in the voices that don't have grace notes at their beginning.

Such new descriptions are created by `*NewbornClone ()` methods, such as:

```

1 S_msrTuplet msrTuplet::createTupletNewbornClone ()
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     if (gTraceOahGroup->getTraceTuplets ()) {
5         std::stringstream ss;
6
7         ss <<
8             "Creating a newborn clone of tuplet " <<
9             asString ();
10
11         gWaeHandler->waeTraceWithoutInputLocation (
12             __FILE__, __LINE__,
13             ss.str ());
14     }
15 #endif // MF_TRACE_IS_ENABLED
16
17     S_msrTuplet
18     newbornClone =
19         msrTuplet::create (
20             fInputStartLineNumber,
21             fBarLineUpLinkToMeasure->getMeasureNumber (),
22             fTupletNumber,
23             fTupletBracketKind,
24             fTupletLineShapeKind,
25             fTupletShowNumberKind,
26             fTupletShowTypeKind,
27             fTupletFactor,
28             fMemberNotesSoundingWholeNotes,
29             fMemberNotesDisplayWholeNotes);
30
31     return newbornClone;

```

32 }

Such a newborn clone is created and used this way in method `:: ()`:

```

1 void msr2lpsrTranslator::visitStart (S_msrTuplet& elt)
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     if (gMsrOahGroup->getTraceMsrVisitors ()) {
5         std::stringstream ss;
6
7         ss <<
8             "--> Start visiting msrTuplet" <<
9             ", line " << elt->getInputStartLineNumber ();
10
11         gWaeHandler->waeTraceWithoutInputLocation (
12             __FILE__, __LINE__,
13             ss.str ());
14     }
15 #endif // MF_TRACE_IS_ENABLED
16
17     // create the tuple clone
18     S_msrTuplet
19     tupleClone =
20         elt->createTupletNewbornClone ();
21
22     // register it in this visitor
23 #ifdef MF_TRACE_IS_ENABLED
24     if (gTraceOahGroup->getTraceTuplets ()) {
25         std::stringstream ss;
26
27         ss <<
28             "+> pushing tuplet '" <<
29             tupleClone->asString () <<
30             "' to tuplets stack";
31
32         gWaeHandler->waeTraceWithoutInputLocation (
33             __FILE__, __LINE__,
34             ss.str ());
35     }
36 #endif // MF_TRACE_IS_ENABLED
37
38     fTupletClonesStack.push_front (tupleClone);
39
40     // is Scheme support needed?
41     switch (elt->getTupletLineShapeKind ()) {
42     case msrTupletLineShapeKind::kTupletLineShapeStraight:
43         break;
44     case msrTupletLineShapeKind::kTupletLineShapeCurved:
45         fResultingLpsr->
46             // this score needs the 'tuplets curved brackets' Scheme function
47             setTupletsCurvedBracketsSchemeFunctionIsNeeded ();
48         break;
49     } // switch
50 }

```

18.4 Deep clones

Some classes in MusicFormats, such as class `msrVoice` in `src/formats/msr/msrVoices.h/.cpp`, have a `*DeepClone ()` method:

```

1 SMARTP<msrVoice> createVoiceDeepClone (
2         int          inputLineNumber,
3         msrVoiceKind voiceKind,
4         int          voiceNumber,
5         const S_msrStaff& containingStaff);

```

Deep copies of the MSR data is not used currently. This can be changed should the need arise in the future.

18.5 Inheritance

18.5.1 Single inheritance

Many classes in MusicFormats use single inheritance. For example, in [src/formats/msr/msrTimeSignature.h](#):

```

1 class EXP msrTimeSignature : public msrMeasureElement
2 {
3     public:
4
5         // creation from MusicXML
6         // -----
7
8         static SMARTP<msrTimeSignature> create (
9             int          inputLineNumber,
10            const S_msrMeasure& upLinkToMeasure,
11            msrTimeSignatureSymbolKind
12                timeSignatureSymbolKind);
13
14        // creation from the applications
15        // -----
16
17        static SMARTP<msrTimeSignature> createTwoEightsTime (
18            int inputLineNumber);
19
20        // ... ..
21
22        // creation from the applications
23        // -----
24
25        static SMARTP<msrTimeSignature> createTimeFromString (
26            int          inputLineNumber,
27            std::string timeString);
28
29        // ... ..

```

The definitions in in [src/formats/msr/msrTimeSignature.cpp](#) are:

```

1 S_msrTimeSignature msrTimeSignature::create (
2     int          inputLineNumber,
3     S_msrMeasure upLinkToMeasure,
4     msrTimeSignatureSymbolKind
5         timeSignatureSymbolKind)
6 {
7     msrTimeSignature* obj =
8         new msrTimeSignature (
9             inputLineNumber,
10            upLinkToMeasure,
11            timeSignatureSymbolKind);
12     assert (obj != nullptr);
13     return obj;
14 }
15
16 msrTimeSignature::msrTimeSignature (

```



```

17  int          inputLineNumber,
18  S_msrMeasure upLinkToMeasure,
19  msrTimeSignatureSymbolKind
20          timeSignatureSymbolKind)
21  : msrMeasureElement (
22      inputLineNumber)
23  {
24      fTimeSignatureSymbolKind = timeSignatureSymbolKind;
25
26      fTimeIsCompound = false;
27  }

```

18.5.2 Single inheritance for smart pointers

All classes for which smart pointers are needed should inherit from class `smartable`, such as in `src/formats/msdl/msd`

```

1  class  msdlScanner : public smartable
2  {
3      public:
4
5          // creation
6          // -----
7
8          static SMARTP<msdlScanner> create (std::istream& inputStream);
9
10     public:
11
12         // constructors/destructor
13         // -----
14
15         msdlScanner (std::istream& inputStream);
16
17         // ... ..
18 };

```

This leads to the following in `src/formats/msdl/msdlScanner.cpp`:

```

1  S_msdlScanner msdlScanner::create (std::istream& inputStream)
2  {
3      msdlScanner* obj =
4          new msdlScanner (inputStream);
5      assert (obj != nullptr);
6      return obj;
7  }
8
9  msdlScanner::msdlScanner (std::istream& inputStream)
10     : fInputStream (
11         inputStream),
12       fCurrentToken (
13         ),
14       fCurrentTokenKind (
15         fCurrentToken.getTokenKindNonConst ()),
16       fCurrentTokenDescription (
17         fCurrentToken.getTokenDescriptionNonConst ())
18  {
19      // trace
20      #ifdef MF_TRACE_IS_ENABLED
21          fTraceTokens      = gGlobalMsd12msr0ahGroup->getTraceTokens ();
22          fTraceTokensDetails = gGlobalMsd12msr0ahGroup->getTraceTokensDetails ();
23      #endif // MF_TRACE_IS_ENABLED
24
25      // ... ..
26  }

```

18.5.3 Multiple inheritance for visitors

Multiple inheritance is used extensively in visitors, which is the way to specify what elements are it seen by the visitor. For example, in `src/formats/msr/msr2msrTranslator.h`, there is:

```

1 class EXP msr2msrTranslator :
2
3     public visitor<S_msrScore>,
4
5     // rights
6
7     public visitor<S_msrIdentification>,
8
9     public visitor<S_msrCredit>,
10    public visitor<S_msrCreditWords>,
11
12    // ... ..
13 };

```

Then there are `visitStart ()` and/or `visitEnd ()` methods to handle the corresponding elements:

```

1 void msr2msrTranslator::visitStart (S_msrIdentification& elt)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gMsrOahGroup->getTraceMsrVisitors ()) {
5             std::stringstream ss;
6
7             ss <<
8                 "--> Start visiting msrIdentification" <<
9                 ", line " << elt->getInputStartLineNumber ();
10
11             gWaeHandler->waeTraceWithoutInputLocation (
12                 __FILE__, __LINE__,
13                 ss.str ());
14         }
15     #endif // MF_TRACE_IS_ENABLED
16
17     ++gIndenter;
18
19     // set the current identification
20     fCurrentIdentification = elt;
21
22     // store it in the resulting MSR score
23     fResultingNewMsrScore->
24         setIdentification (
25             fCurrentIdentification);
26
27     fOnGoingIdentification = true;
28 }

```

```

1 void msr2msrTranslator::visitEnd (S_msrIdentification& elt)
2 {
3     fOnGoingIdentification = false;
4
5     --gIndenter;
6
7     #ifdef MF_TRACE_IS_ENABLED
8         if (gMsrOahGroup->getTraceMsrVisitors ()) {
9             std::stringstream ss;
10
11             ss <<
12                 "--> End visiting msrIdentification" <<
13                 ", line " << elt->getInputStartLineNumber ();
14
15             gWaeHandler->waeTraceWithoutInputLocation (

```

```

16     __FILE__, __LINE__,
17     ss.str ());
18 }
19 #endif // MF_TRACE_IS_ENABLED
20 }

```

Forgetting to define those `visit* ()` methods causes *no error message whatsoever*: the corresponding elements are just not handled by the visitor.

The visitors trace options are useful to detect such cases:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/musicxml > xml2ly -find visitors
2 3 occurrences of string "visitors" have been found:
3 1:
4   -tmxmltvis, -trace-mxsr-visitors
5   Write a trace of the MusicXML tree visiting activity to standard error.
6 2:
7   -tmsrvis, -trace-msr-visitors
8   Write a trace of the MSR graphs visiting activity to standard error.
9 3:
10  -tlpsrvis, -trace-lpsr-visitors
11  Write a trace of the LPSR graphs visiting activity to standard error.

```

18.5.4 Multiple inheritance in other classes

The only such case is class `mfIndentedOstream` in `src/utilities/mfIndentedTextOutput.cpp.h/.cpp`:

```

1 class EXP mfIndentedOstream: public std::ostream, public smartable
2 {
3     /*
4     Reference for this class:
5     https://stackoverflow.com/questions/2212776/overload-handling-of-stdendl
6
7     Usage:
8     mfIndentedOstream myStream (std::cout);
9
10    myStream <<
11    1 << 2 << 3 << std::endl <<
12    5 << 6 << std::endl <<
13    7 << 8 << std::endl;
14    */
15
16    public:
17
18        // creation
19        // -----
20
21        static SMARTP<mfIndentedOstream> create (
22            std::ostream& theOStream,
23            mfOutputIndenter& theIndenter)
24        {
25            mfIndentedOstream* obj = new mfIndentedOstream (
26                theOStream,
27                theIndenter);
28            assert (obj != nullptr);
29
30            return obj;
31        }
32
33        // constructors/destructor
34        // -----
35
36        mfIndentedOstream (
37            std::ostream& theOStream,

```

```

38         mfOutputIndenter& theIndenter)
39         : std::ostream (
40             & fIndentedStreamBuf),
41         fIndentedStreamBuf (
42             theOStream,
43             theIndenter)
44         {}
45
46     virtual ~mfIndentedOstream () {};
47
48 public:
49
50     // public services
51     // -----
52
53     // flush
54     void flush ()
55         { fIndentedStreamBuf.flush (); }
56
57     // indentation
58     mfOutputIndenter& getIndenter () const
59         { return fIndentedStreamBuf.getOutputIndenter (); }
60
61     void incrIndentation ()
62         { ++ (fIndentedStreamBuf.getOutputIndenter ()); }
63
64     void decrIndentation ()
65         { -- (fIndentedStreamBuf.getOutputIndenter ()); }
66
67 private:
68
69     // private fields
70     // -----
71
72     // mfIndentedOstream just uses an mfIndentedOStreamBuf
73     mfIndentedOStreamBuf fIndentedStreamBuf;
74
75 };
76 typedef SMARTP<mfIndentedOstream> S_mfIndentedOstream;

```

18.5.5 Reversibility

All formats in MusicFormats that can be obtained by a conversion from another one should be convertible back in the latter, without information loss.

Thus:

- MXSR contains nearly everything that can be described in MusicXML data. The main `std::exception` at the time of this writing is the MIDI information, see subsection [26.1.1](#) [MusicXML coverage], page [237](#);
- 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 19

Displaying formats

MusicFormats is equipped with option `-display*` options as a help to the maintainer.

19.1 Display categories

19.2 Displaying in practise

```
1 %void lpsr2lilypondTranslator::generateCodeForNoteInMeasure (
2 %   const S_msrNote& note)
3 %{
4 %   int inputLineNumber =
5 %       note->getInputStartLineNumber ();
6 %
7 %#ifdef MF_TRACE_IS_ENABLED
8 %   if (gTraceOahGroup->getTraceNotes ()) {
9 %       std::stringstream ss;
10 %
11 %       ss <<
12 %           std::endl <<
13 %           "% --> generating code for noteRegularInMeasure " <<
14 %           note->asString () <<
15 %           ", line " << inputLineNumber <<
16 %           std::endl;
17 %
18 %       gLog          << ss.str ();
19 %       fLilypondCodeStream << ss.str ();
20 %   }
21 %#endif // MF_TRACE_IS_ENABLED
22 %
23 %
```

Chapter 20

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 21 [MSR time-oriented representation], page 218, for a presentation of how it handles time-oriented concerns.

Some of the data in MSR are supplied by the code that uses MSR, as in class `msrSlur`:

```
1  static SMARTP<msrSlur> create (
2      int                inputLineNumber ,
3      int                slurNumber ,
4      msrSlurTypeKind    slurTypeKind ,
5      msrLineTypeKind    slurLineTypeKind ,
6      msrPlacementKind    slurPlacementKind);
7
8  // ... ..
9
10 // private fields
11 // -----
12
13
14 int                fSlurNumber;
15
16 msrSlurTypeKind    fSlurTypeKind;
17
18 msrLineTypeKind    fSlurLineTypeKind;
19
20 msrPlacementKind    fSlurPlacementKind;
```

Other data are computed by the MSR private methods. For example, in `src/formats/msr/msrVoices.h`:

```
1  // there can only be 4 regular voices in a staff
2  // (those that can contain beamed notes)
3  // and we need a number for the orientation of beams
4  int                fRegularVoiceStaffSequentialNumber;
5
6  // ... ..
7
8  // fVoiceShortestNoteWholeNotes and fVoiceShortestNoteTupletFactor
9  // are used to compute a number of divisions per quarter note
10 // if needed, such as when generating MusicXML from MSR
11 mFRational          fVoiceShortestNoteWholeNotes;
12 msrTupletFactor      fVoiceShortestNoteTupletFactor;
```

There are also data that varies during the lifetime of the object, while it is being populated for example. One such case is class `msrMeasure`:

```

1      mfRational          fCurrentMeasureCurrentAccumulatedWholeNotesDuration;
2                          // this increases when musical elements
3                          // are appended to the measure

```

MSR has been designed to be as general as possible, leading it to contain informations fitted to the various textual formats that can be converted to it or output from it by MusicFormats services.

It is a *very fine-grained* representation of scores:

- some informations it contains are present as such in the textual formats;
- others are computed when the representation is populated, such as, in `src/formats/msr/msrVoices.h`:

```

1      mfRational          fVoiceShortestNoteWholeNotes;

```

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.

20.1 MSR basic types

Some types used throughout MSR are defined in `src/formats/msr/msrBasicTypes.h/.cpp`:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > egrep -rIn '^// '
   msrBasicTypes.h
2 msrBasicTypes.h:29:// input line numbers
3 msrBasicTypes.h:34:// names lists max length
4 msrBasicTypes.h:35:// -----
5 msrBasicTypes.h:39:// XMLLang
6 msrBasicTypes.h:52:// diatonic pitches
7 msrBasicTypes.h:69:// alterations
8 msrBasicTypes.h:90:// accidentals
9 msrBasicTypes.h:124:// editorial accidentals
10 ... ..
11 msrBasicTypes.h:1840:// moments
12 msrBasicTypes.h:1938:// tuplet factors
13 msrBasicTypes.h:2024:// harmonies intervals
14 msrBasicTypes.h:2134:// harmonies structure
15 msrBasicTypes.h:2231:// harmonies contents
16 msrBasicTypes.h:2320:// harmonies details and analysis
17 msrBasicTypes.h:2333:// RGB colors
18 msrBasicTypes.h:2391:// AlphaRGB colors
19 msrBasicTypes.h:2444:// score notation kinds
20 msrBasicTypes.h:2455:// global variables
21 msrBasicTypes.h:2500:// initialization

```

20.2 Data matching across formats

Choices have to be made regarding the way we represent music scores elements, since this varies across formats.

In particular, the way MusicXML structures the elements is not what MSR does. For example, class `msrIdentification` in `src/formats/msr/msrIdentification.h` contains:

```

1 class EXP msrIdentification : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // work
11        std::string          fWorkNumber;
12
13        // ... ..
14
15        // creators
16
17        // ... ..
18
19        std::list<std::string>
20            fSoftwaresList;
21
22        // ... ..
23 };

```

This information is stored in distinct elements in MusicXML:

```

1 <score-partwise>
2   <work>
3     <work-number>K. 331</work-number>
4     <work-title>Piano Sonata in A Major</work-title>
5   </work>
6   <identification>
7     <creator type="composer">Wolfgang Amadeus Mozart</creator>
8     <rights>Copyright © 2003 Recordare LLC</rights>
9     <encoding>
10      <software>Finale 2003 for Windows</software>
11      <software>Dolet for Finale 1.3</software>
12      <encoding-date>2003-03-14</encoding-date>
13    </encoding>

```

The same occurs for MusicXML's `<direction/>` elements, that contain distinct subelements `<words/>` and `<metronome/>`:

```

1   <direction>
2     <direction-type>
3       <words>Adagio</words>
4     </direction-type>
5     <direction-type>
6       <metronome>
7         <beat-unit>long</beat-unit>
8         <per-minute>100</per-minute>
9       </metronome>
10    </direction-type>
11  </direction>

```


Note that two `<direction-type/>` elements are needed, since only one of `<words/>` and `<metronome/>` can be present in a given instance, as stated in `direction.mod`:

```
1 <!ELEMENT direction-type (rehearsalMark+ | segno+ | coda+ |
2 (words | symbol)+ | wedge | dynamics+ | dashes |
3 bracket | pedal | metronome | octave-shift | harp-pedals |
4 damp | damp-all | eyeglasses | std::string-mute |
5 scordatura | image | principal-voice | percussion+ |
6 accordion-registration | staff-divide | other-direction)>
```

This is not a problem in GUI applications, since all those elements are simply *drawn*. MSR stores this in a single class `msrTempo` class in `src/formats/msr/msrTempos.h/.cpp`, since musicians use tempo indications as a whole. See chapter 42 [Tempos handling], page 313 and section 20.17 [Tempos], page 193 for more details.

20.3 Lengths

There are several cases where a length is used in MSR, hence:

```
1 enum class msrLengthUnitKind {
2     kUnitInch, kUnitCentimeter, kUnitMillimeter
3 };
```

```
1 class EXP msrLength : public smartable
2 {
3     // ... ..
4
5     msrLengthUnitKind    fLengthUnitKind;
6     float                fLengthValue;
```

20.4 Sounding and displayed durations

All durations are represented by `mfRational` numbers whose denominators are powers of 2, such as `mfRational` (3, 16, and relative to the duration of a whole note).

This information is a field of class `msrMeasureElement`:

```
1     mfRational                fSoundingWholeNotes;
```

In a tuplet, the sounding durations are different than the written durations, so we store the written duration in class `msrNote`:

```
1     // whole notes
2     mfRational                fNoteDisplayWholeNotes;
3
4     int                        fNoteDotsNumber;
5
6     msrNotesDurationKind      fNoteGraphicNotesDurationKind;
7
8     msrTupletFactor            fNoteTupletFactor;
9
10    msrQuarterTonesPitchKind
11    mfRational                fNoteQuarterTonesDisplayPitchKind;
12    msrOctaveKind              fNoteDisplayOctaveKind;
13                                // for unpitched notes
14                                // and pitched rests
```

Enumeration type `msrNotesDurationKind` is declared in `src/formats/msr/msrNotesDurations.h`:

```

1 enum class msrNotesDurationKind {
2     kNotesDuration_UNKNOWN_,
3
4     // from longest to shortest for the algorithms
5     kNotesDurationMaxima, kNotesDurationLonga, kNotesDurationBreve,
6     kNotesDurationWhole, kNotesDurationHalf,
7     kNotesDurationQuarter,
8     kNotesDurationEighth, kNotesDuration16th, kNotesDuration32nd, kNotesDuration64th,
9     kNotesDuration128th, kNotesDuration256th, kNotesDuration512th, kNotesDuration1024th
10 };

```

20.5 Measure positions and moments

Measure positions are represented by `mfRational` numbers such as 3/8, 1/1 being a whole note.

Measure positions are stored in field `msrMeasureElement::fMeasurePosition` in class `msrMeasureElement`, defined in `src/formats/msr/msrMeasureElement.h/.cpp`:

```

1 class EXP msrMeasureElement : public msrElement
2 {
3     // ... ..
4
5     protected:
6
7         // protected fields
8         // -----
9
10        /*
11         The uplink to measure is declared in the sub-classes,
12         to allow for separate *.h files, C++ constraint
13        */
14
15        mfRational          fSoundingWholeNotes;
16
17        std::string         fBarLineUpLinkToMeasure->getMeasureNumber ();
18
19        mfRational          fMeasurePosition;
20        mfRational          fVoicePosition;
21 };

```

LilyPond represents grace notes positions with a so-called *moment*, that complement the measure position with a relative offset. Grace notes durations are not accounted for in the whole notes duration of measures.

Class `msrMoment` stores a position in a measure, with a relative offset since harmonies can be placed on a note during its sounding time:

```

1     mfRational          fWrittenPositionInMeseasure;
2     mfRational          fSoundingRelativeOffset;

```

20.6 Rests and skips

A skip is an invisible rest, i.e. the meaning is the same as that in LilyPond. Skips are created to fill the holes between notes wherever needed, in order for all voices to be notes/rests/skips sequences.

Skips are also created in `src/passes/msr2lpsr/` to circumvent the LilyPond #34 issue.

20.7 Solo notes and rests

A solo note or rest is characterized as sounding alone in its multi-voice staff for its whole duration.

In the case of a solo rests, such detection allows for better output, in particular when LilyPond code is generated.

An example is at figure 20.1 [The solo rests problem], page 186 : the eighth rests in the second measure of voice 1 of the first staff should be placed on the middle line of the staff, as MuseScore does.



Figure 20.1: The solo rests problem

20.8 Linear versus time-oriented representation

Most music scoring GUI applications handle music as containing voices, which are made of sequences of notes, chords, tuplets and such. This is a horizontal, linear view of the music in the score.

Another view of the music is time-oriented, i.e., what are notes being played at a given moment in time? This is a vertical view of the music, which is highlighted in piano roll views.

MSR stores descriptions of so-called 'measures slice' through class `msrMeasuresSlice`, defined in `src/formats/msr/msrMeasuresSlice.h`. Then a time-oriented view of a voice, staff or part is a sequence of such measure slices, defined in class `msrMeasuresSlicesSequence`.

An class `msrMeasuresSlice` contains basically a slice measures vector:

```
1 // the measures in the slice
2 std::vector<S_msrMeasure> fSliceMeasuresVector;
```

From this, the following other descriptions are derived:

```
1 // notes flat list
2 std::list<S_msrNote> fSliceNotesFlatList;
3
4 // note events list
5 std::list<S_msrNoteEvent>
6     fSliceNoteEventsList;
7
8 // simultaneous notes chunks list
9 std::list<S_msrSimultaneousNotesChunk>
10     fSliceSimultaneousNotesChunksList;
```

Note events are distinguished with enumeration type `msrNoteEventKind`:

```
1 // -----
2 enum class msrNoteEventKind {
3     kNoteEventStart,
4     kNoteEventStop
5 };
```

Class `msrNoteEvent` contains:

```
1 mfrational          fNoteEventMeasurePosition;
2 S_msrNote          fNoteEventNote;
3 msrNoteEventKind    fNoteEventKind;
```

20.9 Spanners

A spanner... spans from one note or rest to another one. A choice to be made about when to use spanners: should wedges < and > be handled as spanners, or simply as being attached to notes? It has been selected to use spanners only for ligatures apart from true spanners.

MusicXML uses "start", "start" and "start" attributes, which need to be present in MSR for MusicXML generation. They are reflected in MSR as enumeration type enumeration type `msrSpannerTypeKind`, defined this way:

```
1 // spanner types
2 // -----
3 enum class msrSpannerTypeKind {
4     kSpannerType_UNKNOWN_,
5
6     kSpannerTypeStart, kSpannerTypeContinue, kSpannerTypeStop
7 };
```

20.10 Uplinks, direct uplinks and sidelinks

An uplink is a direct pointer from one class instance to one that contains it. Some are a link to the containing class instance, whilst others are shortcut links higher in the graph for speed. For example, class `msrNote` contains:

```
1 // upLinks
2 // -----
3
4 S_msrMeasure          fNoteUpLinkToMeasure;
5
6 S_msrChord            fNoteShortcutUpLinkToChord;
7
8 S_msrGraceNotesGroup  fNoteShortcutUpLinkToGraceNotesGroup;
9
10 S_msrTuplet           fNoteShortcutUpLinkToTuplet;
```

A sidelink is used in ligatures and spanners, so that each end of the structure can reference the other one.

For example, MusicFormats defines enumeration type `msrLigatureKind` in `src/formats/msr/msrLigatures.h`:

```
1 enum class msrLigatureKind {
2     kLigatureNone,
3     kLigatureStart, kLigatureContinue, kLigatureStop
4 };
```

Class `msrLigature` contains:

```

1 private:
2
3     // sideLinks
4     // -----
5     S_msrLigature          fLigatureSideLinkToOtherEnd; // two-way

```

Enumeration type `is` is declared in `src/formats/msr/msrSpanners.h`:

```

1 enum class msrSpannerKind {
2     kSpannerDashes, kSpannerWavyLine
3 };

```

20.11 Printing descriptions

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

```

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

```

There are also more specific methods such as:

```

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

```

Note that:

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

Most classes in `MusicFormats` can be printed with the `<<` operator:

```

1 std::ostream& operator << (std::ostream& os, const S_msrElement& elt)
2 {
3     if (elt) {
4         elt->print (os);
5     }
6     else {
7         os << "[NULL]" << std::endl;
8     }
9
10    return os;
11 }

```

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

```

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

```

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

```

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

A typical sequence to produce indented output is:

```

1 void msrTransposition::print (std::ostream& os) const
2 {
3     const int fieldWidth = 22;
4
5     os <<
6     "Transposition" <<
7     ", line " << fInputStartLineNumber <<
8     std::endl;
9
10    ++gIndenter;
11
12    os << std::left <<
13    std::setw (fieldWidth) <<
14    "fTranspositionDiatonic" << ": " << fTranspositionDiatonic <<
15    std::endl <<
16    std::setw (fieldWidth) <<
17    "fTranspositionChromatic" << ": " << fTranspositionChromatic <<
18    std::endl <<
19    std::setw (fieldWidth) <<
20    "fTranspositionOctaveChange" << ": " << fTranspositionOctaveChange <<
21    std::endl <<
22    std::setw (fieldWidth) <<
23    "fTranspositionDouble" << ": " << fTranspositionDouble <<
24    std::endl << std::endl;
25
26    --gIndenter;
27 }
```

The main indented output streams are:

```

1 #define gOutput *gGlobalOutputIndentedOstream
2 #define gLog     *gGlobalLogIndentedOstream
```

20.12 Pitches

MSR handle diatonic, semitone and quarter tone pitches, defined in `src/formats/msr/msrBasicTypes.h` as shown below. All pitches data is represented internally as quarter tones pitches, and conversions are done wherever needed.

```

1 // diatonic pitches
2 // -----
3 enum class msrDiatonicPitchKind {
4     kDiatonicPitch_UNKNOWN_,
5
6     // starting at C for LilyPond relative octave calculations
7     kDiatonicPitchC,
8     kDiatonicPitchD, kDiatonicPitchE, kDiatonicPitchF,
9     kDiatonicPitchG, kDiatonicPitchA, kDiatonicPitchB
10 };

```

```

1 // semi tones pitches
2 // -----
3 enum class msrSemiTonesPitchKind {
4     kSTP_NoSemiTonesPitch,
5
6     kSTP_C_TripleFlat,
7     kSTP_C_DoubleFlat, kSTP_C_Flat,
8     kSTP_C_Natural,
9     kSTP_C_Sharp, kSTP_C_DoubleSharp,
10    kSTP_C_TripleSharp,
11
12    kSTP_D_TripleFlat,
13    kSTP_D_DoubleFlat, kSTP_D_Flat,
14    kSTP_D_Natural,
15    kSTP_D_Sharp, kSTP_D_DoubleSharp,
16    kSTP_D_TripleSharp,
17
18    kSTP_E_TripleFlat,
19    kSTP_E_DoubleFlat, kSTP_E_Flat,
20    kSTP_E_Natural,
21    kSTP_E_Sharp, kSTP_E_DoubleSharp,
22    kSTP_E_TripleSharp,
23
24    kSTP_F_TripleFlat,
25    kSTP_F_DoubleFlat, kSTP_F_Flat,
26    kSTP_F_Natural,
27    kSTP_F_Sharp, kSTP_F_DoubleSharp,
28    kSTP_F_TripleSharp,
29
30    kSTP_G_TripleFlat,
31    kSTP_G_DoubleFlat, kSTP_G_Flat,
32    kSTP_G_Natural,
33    kSTP_G_Sharp, kSTP_G_DoubleSharp,
34    kSTP_G_TripleSharp,
35
36    kSTP_A_TripleFlat,
37    kSTP_A_DoubleFlat, kSTP_A_Flat,
38    kSTP_A_Natural,
39    kSTP_A_Sharp, kSTP_A_DoubleSharp,
40    kSTP_A_TripleSharp,
41
42    kSTP_B_TripleFlat,
43    kSTP_B_DoubleFlat, kSTP_B_Flat,
44    kSTP_B_Natural,
45    kSTP_B_Sharp, kSTP_B_DoubleSharp,
46    kSTP_B_TripleSharp
47 };

```

```

1 // quarter tones pitches
2 // -----
3 enum class msrQuarterTonesPitchKind {
4     kQTP_UNKNOWN_,
5
6     kQTP_Rest, kQTP_Skip,

```

```

7
8   kQTP_A_TripleFlat,
9   kQTP_A_DoubleFlat, kQTP_A_SesquiFlat, kQTP_A_Flat, kQTP_A_SemiFlat,
10  kQTP_A_Natural,
11  kQTP_A_SemiSharp, kQTP_A_Sharp, kQTP_A_SesquiSharp, kQTP_A_DoubleSharp,
12  kQTP_A_TripleSharp,
13
14  kQTP_B_TripleFlat,
15  kQTP_B_DoubleFlat, kQTP_B_SesquiFlat, kQTP_B_Flat, kQTP_B_SemiFlat,
16  kQTP_B_Natural,
17  kQTP_B_SemiSharp, kQTP_B_Sharp, kQTP_B_SesquiSharp, kQTP_B_DoubleSharp,
18  kQTP_B_TripleSharp,
19
20  kQTP_C_TripleFlat,
21  kQTP_C_DoubleFlat, kQTP_C_SesquiFlat, kQTP_C_Flat, kQTP_C_SemiFlat,
22  kQTP_C_Natural,
23  kQTP_C_SemiSharp, kQTP_C_Sharp, kQTP_C_SesquiSharp, kQTP_C_DoubleSharp,
24  kQTP_C_TripleSharp,
25
26  kQTP_D_TripleFlat,
27  kQTP_D_DoubleFlat, kQTP_D_SesquiFlat, kQTP_D_Flat, kQTP_D_SemiFlat,
28  kQTP_D_Natural,
29  kQTP_D_SemiSharp, kQTP_D_Sharp, kQTP_D_SesquiSharp, kQTP_D_DoubleSharp,
30  kQTP_D_TripleSharp,
31
32  kQTP_E_TripleFlat,
33  kQTP_E_DoubleFlat, kQTP_E_SesquiFlat, kQTP_E_Flat, kQTP_E_SemiFlat,
34  kQTP_E_Natural,
35  kQTP_E_SemiSharp, kQTP_E_Sharp, kQTP_E_SesquiSharp, kQTP_E_DoubleSharp,
36  kQTP_E_TripleSharp,
37
38  kQTP_F_TripleFlat,
39  kQTP_F_DoubleFlat, kQTP_F_SesquiFlat, kQTP_F_Flat, kQTP_F_SemiFlat,
40  kQTP_F_Natural,
41  kQTP_F_SemiSharp, kQTP_F_Sharp, kQTP_F_SesquiSharp, kQTP_F_DoubleSharp,
42  kQTP_F_TripleSharp,
43
44  kQTP_G_TripleFlat,
45  kQTP_G_DoubleFlat, kQTP_G_SesquiFlat, kQTP_G_Flat, kQTP_G_SemiFlat,
46  kQTP_G_Natural,
47  kQTP_G_SemiSharp, kQTP_G_Sharp, kQTP_G_SesquiSharp, kQTP_G_DoubleSharp,
48  kQTP_G_TripleSharp
49 };

```

20.13 Octaves

They are represented with enumeration type :

```

1  // octaves
2  // -----
3  enum class msrOctaveKind {
4      kOctave_UNKNOWN_,
5
6      kOctave0, kOctave1, kOctave2, kOctave3,
7      kOctave4, // that of middle C
8      kOctave5, kOctave6, kOctave7, kOctave8, kOctave9
9  };

```

For the needs of LilyPond and MSDL, MSR also contains a description of how to enter octaves:

```

1  // octave entry
2  // -----
3  enum class msrOctaveEntryKind {

```



```

4   kOctaveEntryRelative,
5   kOctaveEntryAbsolute,
6   kOctaveEntryFixed
7 };

```

20.14 NotesDurations

They are represented in MSR with the enumeration type `msrNotesDurationKind` enumeration type, defined in `src/formats/msr/msrBasicTypes.h`:

```

1 // durations
2 // -----
3 enum class msrNotesDurationKind {
4     kNotesDuration_UNKNOWN_,
5
6     // from longest to shortest for the algorithms
7     kNotesDurationMaxima, kNotesDurationLonga, kNotesDurationBreve, kNotesDurationWhole,
8     kNotesDurationHalf,
9     kNotesDurationQuarter,
10    kNotesDurationEighth, kNotesDuration16th, kNotesDuration32nd, kNotesDuration64th,
11    kNotesDuration128th, kNotesDuration256th, kNotesDuration512th, kNotesDuration1024th
12 };

```

Class `msrDottedNotesDuration` contains:

```

1     msrNotesDurationKind  fNotesDurationKind;
2     int                   fDotsNumber;

```

20.15 Alterations

```

1 // alterations
2 // -----
3 enum class msrAlterationKind {
4     kAlteration_UNKNOWN_,
5
6     kAlterationTripleFlat, kAlterationDoubleFlat, kAlterationSesquiFlat,
7     kAlterationFlat, kAlterationSemiFlat,
8     kAlterationNatural,
9     kAlterationSemiSharp, kAlterationSharp, kAlterationSesquiSharp,
10    kAlterationDoubleSharp, kAlterationTripleSharp

```

20.16 Accidentals

```

1 // accidentals
2 // -----
3 enum class msrAccidentalKind {
4     kAccidentalNone,
5
6     kAccidentalSharp, kAccidentalNatural,
7     kAccidentalFlat, kAccidentalDoubleSharp,
8     kAccidentalSharpSharp,
9     kAccidentalFlatFlat, kAccidentalNaturalSharp,
10    kAccidentalNaturalFlat, kAccidentalQuarterFlat,
11    kAccidentalQuarterSharp, kAccidentalThreeQuartersFlat,
12    kAccidentalThreeQuartersSharp,
13 };

```

```

14 kAccidentalSharpDown, kAccidentalSharpUp,
15 kAccidentalNaturalDown, kAccidentalNaturalUp,
16 kAccidentalFlatDown, kAccidentalFlatUp,
17 kAccidentalTripleSharp, kAccidentalTripleFlat,
18 kAccidentalSlashQuarterSharp, kAccidentalSlashSharp,
19 kAccidentalSlashFlat, kAccidentalDoubleSlashFlat,
20 kAccidentalSharp_1, kAccidentalSharp_2,
21 kAccidentalSharp_3, kAccidentalSharp_5,
22 kAccidentalFlat_1, kAccidentalFlat_2,
23 kAccidentalFlat_3, kAccidentalFlat_4,
24 kAccidentalSori, kAccidentalKoron,
25
26 kAccidentalOther
27 };

```

20.17 Tempos

There are thus several kinds of tempos in MSR, with variants represented by enumeration type `msrTempoKBeatUnitsKind` in `src/formats/msr/msrTempos.h`:

```

1 class EXP msrTempo : public msrMeasureElement
2 {
3     public:
4
5     // data types
6     // -----
7
8     enum class msrTempoKBeatUnitsKind {
9         kTempoBeatUnits_UNKNOWN_,
10        kTempoBeatUnitsWordsOnly,
11        kTempoBeatUnitsPerMinute,
12        kTempoBeatUnitsEquivalence,
13        kTempoNotesRelationship
14    };
15
16    // ... ..
17
18    enum class msrTempoParenthesizedKind {
19        kTempoParenthesizedYes, kTempoParenthesizedNo
20    };
21
22    // ... ..
23
24    enum class msrTempoNotesRelationshipKind {
25        kTempoNotesRelationshipNone, kTempoNotesRelationshipEquals
26    };
27
28    // ... ..
29 };

```

20.17.1 Tempos notes

A tempo indication can contain a note a notes in a tuplet. Such notes are described by class `msrTempoNote`:

```

1 class EXP msrTempoNote : public msrElement
2 {
3     public:
4
5     // creation from MusicXML
6     // -----
7

```

```

8      static SMARTP<msrTempoNote> create (
9          int                inputLineNumber ,
10         const mFRational& tempoNoteWholeNotes ,
11         Bool                tempoNoteBelongsToATuplet);
12
13 protected:
14
15     // constructors/destructor
16     // -----
17
18         msrTempoNote (
19             int                inputLineNumber ,
20             const mFRational& tempoNoteWholeNotes ,
21             Bool                tempoNoteBelongsToATuplet);
22
23     // ... ..
24
25 private:
26
27     // private fields
28     // -----
29
30     mFRational                fTempoNoteWholeNotes;
31
32     std::list<S_msrBeam>      fTempoNoteBeams;
33
34     Bool                      fTempoNoteBelongsToATuplet;
35 };

```

20.17.2 Tempos tuplets

A tuplet in a tempo representation is described by class `msrTempoTuplet`:

```

1 // -----
2 class EXP msrTempoTuplet : public msrElement
3 {
4     public:
5
6     // data types
7     // -----
8
9     enum class msrTempoTupletTypeKind {
10         kTempoTupletTypeNone ,
11         kTempoTupletTypeStart , kTempoTupletTypeStop
12     };
13
14     // ... ..
15
16     enum class msrTempoTupletBracketKind {
17         kTempoTupletBracketYes , kTempoTupletBracketNo
18     };
19
20     // ... ..
21
22     enum class msrTempoTupletShowNumberKind {
23         kTempoTupletShowNumberActual ,
24         kTempoTupletShowNumberBoth ,
25         kTempoTupletShowNumberNone
26     };
27
28     // ... ..
29
30     // creation from MusicXML
31     // -----

```

```

32
33     static SMARTP<msrTempoTuplet> create (
34         int                inputLineNumber ,
35         int                tempoTupletNumber ,
36         msrTempoTupletBracketKind  tempoTupletBracketKind ,
37         msrTempoTupletShowNumberKind  tempoTupletShowNumberKind ,
38         msrTupletFactor        tempoTupletFactor ,
39         msrWholeNotes          memberNotesDisplayWholeNotes);
40
41 protected:
42
43     // constructors/destructor
44     // -----
45
46     msrTempoTuplet (
47         int                inputLineNumber ,
48         int                tempoTupletNumber ,
49         msrTempoTupletBracketKind  tempoTupletBracketKind ,
50         msrTempoTupletShowNumberKind  tempoTupletShowNumberKind ,
51         msrTupletFactor        tempoTupletFactor ,
52         msrWholeNotes          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     mfrational            fMemberNotesDisplayWholeNotes;
72
73     mfrational            fTempoTupletDisplayWholeNotes;
74
75     std::list<S_msrElement>
76         fTempoTupletElements;
77 };

```

20.17.3 Tempos description

The private fields in class `msrTempo` are:

```

1 class EXP msrTempo : public msrMeasureElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        msrTempoKBeatUnitsKind
11            fTempoKind;
12
13        std::list<S_msrWords> fTempoWordsList;

```

```

14
15     msrDottedNotesDuration      fTempoBeatUnit;
16
17     std::string                  fTempoPerMinute; // '90' or '132-156' for example
18     msrDottedNotesDuration      fTempoEquivalentBeatUnit;
19
20     S_msrTempoNotesRelationshipElements
21         fTempoNotesRelationshipLeftElements;
22     msrTempoNotesRelationshipKind
23         fTempoNotesRelationshipKind;
24     S_msrTempoNotesRelationshipElements
25         fTempoNotesRelationshipRightElements;
26
27     msrTempoParenthesizedKind
28         fTempoParenthesizedKind;
29
30     msrPlacementKind            fTempoPlacementKind;
31 };

```

Among these fields:

- field `msrTempo::fTempoKind` denotes the variant;
- field `msrTempo::fTempoWordsList` contains the words that can be present, such as 'adagio molto';
- field `msrTempo::fTempoBeatUnit` is a dotted duration, as in '4.';
- field `msrTempo::fTempoPerMinute` is a `std::string`, since it can contain ranges indication as in '4. = 60-66';
- field `msrTempo::fTempoEquivalentBeatUnit` is a dotted duration;
- field `msrTempo::fTempoNotesRelationshipLeftElements`, field `msrTempo::fTempoNotesRelationshipKind` and field `msrTempo::fTempoNotesRelationshipRightElements` are used when a relationship is present, such as '2. = 1', in which case field `msrTempo::fTempoNotesRelationshipKind` contains field `msrTempo::kTempoNotesRelationshipKind`;
- field `msrTempo::fTempoParenthesizedKind` indicates whether the tempo indication is parenthesized;
- field `msrTempo::fTempoPlacementKind` tells whether the tempo is to be placed above or below the staff, constant `msrPlacementKind::kPlacementAbove` by default.

20.18 Clefs

Clefs are distinguished using enumeration type `msrClefKind`:

```

1 // clefs
2 // -----
3
4 enum class msrClefKind {
5     kClef_UNKNOWN_,
6
7     kClef_Treble,
8     kClef_Soprano, kClef_MezzoSoprano, kClef_Alto, kClef_Tenor, kClef_Baritone, kClef_Bass,
9     kClef_TrebleLine1,
10    kClef_TrebleMinus15, kClef_TrebleMinus8, kClef_TreblePlus8, kClef_TreblePlus15,
11
12    kClef_BassMinus15, kClef_BassMinus8, kClef_BassPlus8, kClef_BassPlus15,
13
14    kClef_Varbaritone,
15
16    kClef_Tablature4, kClef_Tablature5, kClef_Tablature6, kClef_Tablature7,
17

```

```

18   kClefPercussion ,
19
20   kClefJianpu
21 };

```

Class `msrClef` contains:

```

1   msrClefKind      fClefKind;
2   int              fClefStaffNumber;

```

20.19 Keys

MSR, as MusicXML, supports Humdrum-Scot keys as well as traditional key such as C and 6/8.

A Humdrum-Scot key is composed of items represented by class `msrHumdrumScotKeyItem`, each containing:

```

1   msrDiatonicPitchKind  fKeyDiatonicPitchKind;
2   msrAlterationKind     fKeyAlterationKind;
3   msrOctaveKind         fKeyOctaveKind;

```

An example is at figure 20.2 [Humdrum-Scot keys], page 197. It has been produced by:

```

1 xml2ly -auto-output-file-name keys/HumdrumScotKeys.xml

```

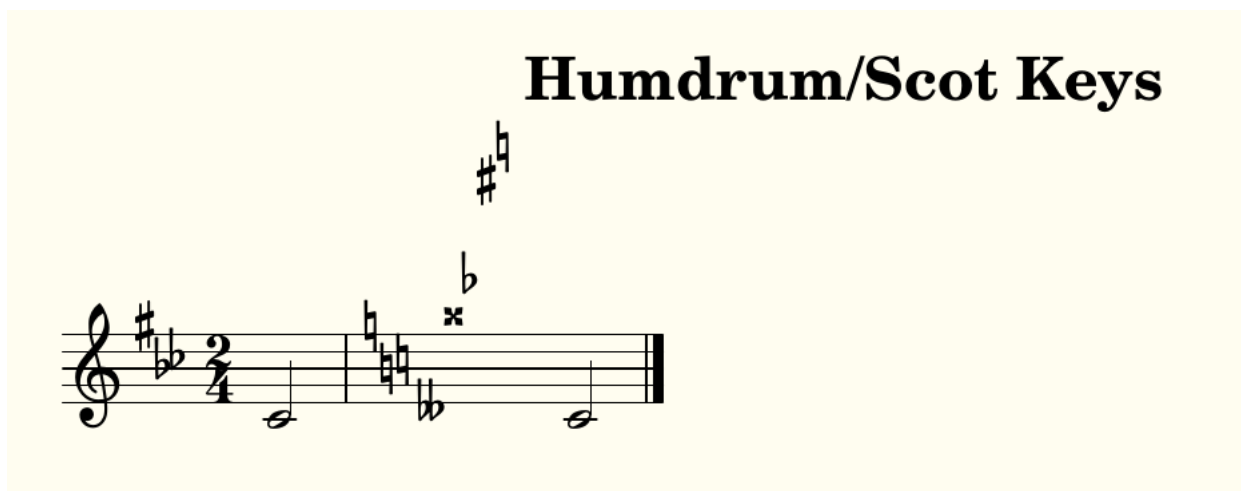


Figure 20.2: Humdrum-Scot keys

Class `msrKey` thus contains:

```

1   // private fields
2   // -----
3
4   msrKeyKind      fKeyKind;
5
6   // traditional keys
7
8   msrQuarterTonesPitchKind
9   fKeyTonicQuarterTonesPitchKind;
10  msrModeKind      fModeKind;
11  int              fKeyCancel;
12
13  // Humdrum/Scot keys
14
15  std::vector<S_msHumdrumScotKeyItem>

```

```

16         fHumdrumScotKeyItemsVector;
17     Bool        fKeyItemsOctavesAreSpecified;

```

20.20 Time signatures

The variants in time signatures are distinguished by enumeration type enumeration type `msrTimeSignatureSymbolKind`

```

1 // time symbols
2 // -----
3 enum class msrTimeSignatureSymbolKind {
4     kTimeSignatureSymbolNone,
5     kTimeSignatureSymbolCommon,
6     kTimeSignatureSymbolCut,
7     kTimeSignatureSymbolNote,
8     kTimeSignatureSymbolDottedNote,
9     kTimeSignatureSymbolSingleNumber,
10    kTimeSignatureSymbolSenzaMisura
11 };

```

A time signature can also be structured, and this is described by those two types:

```

1 enum class msrTimeSignatureSeparatorKind {
2     kTimeSignatureSeparatorNone,
3     kTimeSignatureSeparatorHorizontal,
4     kTimeSignatureSeparatorDiagonal,
5     kTimeSignatureSeparatorVertical,
6     kTimeSignatureSeparatorAdjacent
7 };

```

```

1 enum class msrTimeSignatureRelationKind {
2     kTimeSignatureRelationNone,
3     kTimeSignatureRelationParentheses,
4     kTimeSignatureRelationBracket,
5     kTimeSignatureRelationEquals,
6     kTimeSignatureRelationSlash,
7     kTimeSignatureRelationSpace,
8     kTimeSignatureRelationHyphen
9 };

```

A brick that can be used in class `msrTimeSignature` is `msrTimeSignatureItem`, whose private fields are:

```

1     std::vector<int>        fTimeSignatureBeatsNumbersVector; // 5+3+1 is possible
2     int                    fTimeSignatureBeatValue;

```

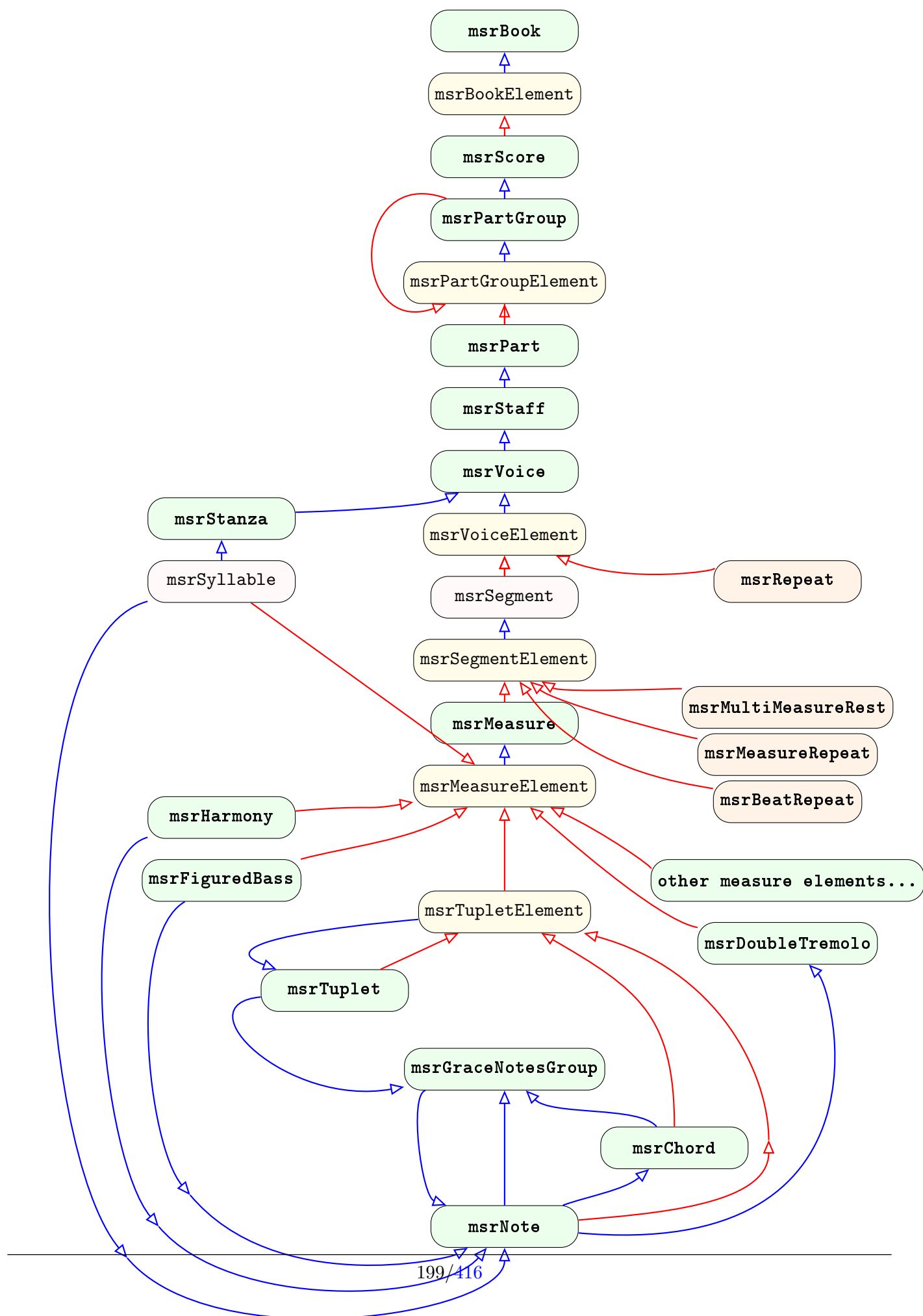
Class `msrTimeSignature` contains:

```

1     msrTimeSignatureSymbolKind
2         fTimeSignatureSymbolKind;
3
4     std::vector<S_msTimeSignatureItem>
5         fTimeSignatureItemsVector;
6
7     // a time is compound if it contains several items
8     // or if the only one has several beats numbers
9     // i.e. 3/4 is not, (3+4)/8 is, and 2/4+3/4 is too
10    Bool        fTimeIsCompound;

```

Figure 20.3: The MSR classes hierarchy



20.21 MSR classes inheritance

The picture at figure 20.3 [The MSR classes hierarchy], page 199, shows the hierarchy of the main MSR classes. The colors are used as follows:

The background colors are used as follows:

- **green**: a score element that is expected to be found in a score representation, such as class `msrStaff` and class `msrChord`;
- **pink**: a element needed in MSR to structure the representation, such as class `msrSegment` and class `msrSyllable`;
- **yellow**: a base class with name class `msr*Element` for elements that can be used in another class, such as class `msrVoiceElement`;

The arrows colors have the following meaning:

- **red**: a link from a class to its base class. For example, class `msrPart` is derived from class `msrPartGroupElement`, class `msrPartGroup` is derived from class `msrPartGroupElement`, and class `msrChord` is derived from class `msrTupletElement`;
- **blue**: one or more fields of a class are smart pointers to instances of another. For example, an class `msrChords` instance may be an element of a class `msrGraceNotesGroup` instance.

When not shown for clarity, the common base class of all these classes is class `msrElement`, that contains an integer input line number.

The `otherMeasureElements` classes are:

- bars:
 - class `msrBarCheck`
 - class `msrBarNumberCheck`
 - class `msrBarLine`
 - class `msrHiddenMeasureAndBarLine`
- breaks:
 - class `msrLineBreak`
 - class `msrPageBreak`
- notes:
 - class `msrVoiceStaffChange`
 - class `msrOctaveShift`
- clefs, keys, times, tempo:
 - class `msrClef`
 - class `msrKey`
 - class `msrTime`
 - class `msrTempo`

- instruments:
 - class `msrStaffDetails`
 - class `msrScordatura`
 - class `msrAccordionRegistration`
 - class `msrHarpPedalsTuning`
 - class `msrPedal`
 - class `msrDamp`
 - class `msrDampAll`
- lyrics:
 - class `msrSyllable`
- rehearsals, segno and coda:
 - class `msrRehearsalMark`
 - class `msrSegno`
 - class `msrDalSegno`
 - class `msrCoda`
- others:
 - class `msrPrintLayout`
 - class `msrEyeGlasses`
 - class `msrStaffLevelElement`
 - class `msrTransposition`
 - class `msrTupletElement`

20.22 Books

Books handling is presented at section [54](#) [Books handling], page [338](#).

LilyPond handles `\book {...}` by placing the scores one after the other in the resulting PDF or SVG files. It will also generate separate MIDI files if a `\markup {...}` block is used.

There is no such concept in MusicXML, but MSR uses it for completeness, creating an implicit class `msrBook` instance if needed.

An class `msrBook` contains a list and a set of `S_msrBookElement`:

```

1 // book elements
2 std::set<S_msrBookElement> fBookElementsSet;
3
4 std::list<S_msrBookElement> fBookElementsList;
```

Currently, the only book element used is the class `msrScore`, but others might come, such as texts, which LilyPond allows as `\markup {...}`:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrBook' *
2 formats/msr/msrScores.h: class EXP msrScore : public msrBookElement
```

20.23 Scores

Scores handling is presented at section ?? [Scores handling], page ??.

A score in MSR is the usual music score concept. It contains a set and a list of `S_msrPartGroup`:

```
1 // part groups
2 std::set<S_msrPartGroup>    fScorePartGroupsSet;
3
4 std::list<S_msrPartGroup>    fPartGroupsList;
```

20.24 Part groups

Part groups handling is presented at section 52 [Part groups handling], page 335.

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 outer-most part group exists in MSR if the score does not contain explicit part groups.

An class `msrPartGroup` thus contains parts and part groups in any order, as is found in symphonic music scores:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrPartGroupElement' *
2 formats/msr/msrParts.h:class EXP msrPart : public msrPartGroupElement
3 formats/msr/msrPartGroups.h:class EXP msrPartGroup : public msrPartGroupElement
```

which are stored in a list:

```
1 // allowing for both parts and nested part groups as elements
2 std::list<S_msrPartGroupElement>
3                               fPartGroupElementsList;
```

20.25 Parts

Parts handling is presented at section 51 [Parts handling], page 333.

A part in MSR is composed of voices, stored in:

```
1 // staves
2
3 std::map<int, S_msrStaff>
4                               getPartStaveNumbersToStavesMap;
5 std::list<S_msrStaff>    fPartAllStavesList;
6
7 // harmonies
8
9 S_msrStaff                fPartHarmoniesStaff;
10 S_msrVoice               fPartHarmoniesVoice;
11
12 // figured bass
13
14 S_msrStaff                fPartFiguredBassStaff;
15 S_msrVoice               fPartFiguredBassVoice;
16
17 // voices
18
19 std::list<S_msrVoice>    fPartAllVoicesList;
```

20.26 Staves

Staves handling is presented at section ?? [Staves handling], page ??.

A staff contains at most 4 numbered voices, stored in:

```

1  // the mapping of all the voices in the staff,
2  // including harmonies and figured bass voices
3  std::map<int, S_msrVoice>
4      fStaffVoiceNumbersToAllVoicesMap;
5
6  // the mapping of voice numbers to regular voices
7  std::map<int, S_msrVoice>
8      fStaffVoiceNumbersToRegularVoicesMap;
9
10 // we need to handle the regular voice specifically
11 // to assign them sequencing numbers from 1 to gMaxStaffVoices,
12 // needed to set the beams orientation (up or down)
13 int      fStaffRegularVoicesCounter;
14
15 // harmonies and figured bass elements should be placed %%JMI
16 // in the first regular voice of the staff, hence:
17 std::list<S_msrVoice> fStaffRegularVoicesList;
18
19 // we need to sort the voices by increasing voice numbers,
20 // but with harmonies voices right before the corresponding regular voices
21 std::list<S_msrVoice> fStaffAllVoicesList;

```

20.27 Voice elements

Voices contain instances of class `msrVoiceElement`, defined in `src/formats/msr/msrVoiceElements.h/.cpp`:

```

1  // -----
2  /*
3   Various elements can found in voices,
4   hence class  msrVoiceElement
5  */
6
7  class EXP msrVoiceElement : public msrElement
8  {
9  public:
10
11     // creation from MusicXML
12     // -----
13
14     // cloning
15     // -----
16
17  protected:
18
19      msrVoiceElement (
20          int inputLineNumber);
21
22      virtual          ~msrVoiceElement ();
23
24  /*
25   The voice uplink is declared in the sub-classes,
26   to allow for separate *.h files, C++ constraint
27  */
28  };

```

The classes derived from class `msrVoiceElement` are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'public msrVoiceElement
  ' *.h
2 msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
3 msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
4 msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
5 msrMultiMeasureRests.h:class EXP msrMultiMeasureRest : public msrVoiceElement
6 msrSegments.h:class EXP msrSegment : public msrVoiceElement

```

They are describes in specific sections below.

20.28 Voices

Voices handling is presented at section 49 [Voices handling], page 331.

A voice is conceptually a sequence of `S_msVoiceElement`, that may be:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrVoiceElement' *
2 formats/msr/msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
3 formats/msr/msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
4 formats/msr/msrMultiMeasureRests.h:class EXP msrMultiMeasureRest : public msrVoiceElement
5 formats/msr/msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
6 formats/msr/msrSegments.h:class EXP msrSegment : public msrVoiceElement

```

More precisely and for technical reasons, an class `msrVoice` contains:

```

1 // voice initial elements list
2
3 std::list<S_msVoiceElement>
4     fVoiceInitialElementsList;
5
6 // voice first and last segments
7
8 // fVoiceLastSegment contains the music
9 // not yet stored in fVoiceInitialElementsList,
10 // it is thus logically the end of the latter,
11 // and is created implicitly for every voice.
12 // It is needed 'outside' of the 'list<S_msElement>'
13 // because it is not a mere S_msElement, but a S_msSegment
14 S_msSegment     fVoiceLastSegment;
15
16 // fVoiceFirstSegment is used to work around LilyPond issue #34
17 S_msSegment     fVoiceFirstSegment;

```

Each voice is described by a field of enumeration type `msrVoiceKind`, defined in `src/formats/msr/msrBasicTypes.h`:

```

1 enum class msrVoiceKind {
2     kVoiceKindRegular,
3     kVoiceKindDynamics,
4     kVoiceKindHarmonies, // for MusicXML <harmony/>, LilyPond ChordNames
5     kVoiceKindFiguredBass // for MusicXML <figured-bass/>, LilyPond FiguredBass
6 };

```

As stated in the comment above, `fVoiceLastSegment` is used because it because `fVoiceInitialElementsList` can contain any class `msrVoiceElement`, whereas all MSR elements appended to the voice are to be placed in a segment.

An class `msrSegment` instance should thus be created and stored in `fVoiceLastSegment` before class `msrVoiceElement` instances can be appended to the voice.

When repeats are handled, an class `msrRepeat` instance is created. Then the contents of field `msrVoice::fVoiceLastS` is moved into it and a new segment is created, see section 20.35 [Repeats], page 210.

Whether the last segment should be created right when the voice is created is controlled with enumeration type `msrVoiceCreateInitialLastSegmentKind`, defined in `src/formats/msr/msrVoices.h`:

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

20.29 Measures

Measures handling is presented at section 40 [Measures handling], page 288.

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/`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'public
   msrMeasureElement' *.h
2 msrBars.h:class EXP msrBarCheck : public msrMeasureElement
3 msrBars.h:class EXP msrBarNumberCheck : public msrMeasureElement
4 msrBars.h:class EXP msrBarLine : public msrMeasureElement
5 msrBreaks.h:class EXP msrLineBreak : public msrMeasureElement
6 msrBreaks.h:class EXP msrPageBreak : public msrMeasureElement
7 msrClefs.h:class EXP msrClef : public msrMeasureElement
8 msrCodas.h:class EXP msrCoda : public msrMeasureElement
9 msrDoubleTremolos.h:class EXP msrDoubleTremolo : public msrMeasureElement
10 msrEyeGlasses.h:class EXP msrEyeGlasses : public msrMeasureElement
11 msrFiguredBasses.h:class EXP msrFiguredBass : public msrMeasureElement
12 msrHarmonies.h:class EXP msrHarmony : public msrMeasureElement
13 msrHiddenMeasureAndBarLines.h:class EXP msrHiddenMeasureAndBarLine : public
   msrMeasureElement
14 msrInstruments.h:class EXP msrScordatura : public msrMeasureElement
15 msrInstruments.h:class EXP msrAccordionRegistration : public msrMeasureElement
16 msrInstruments.h:class EXP msrHarpPedalsTuning : public msrMeasureElement
17 msrInstruments.h:class EXP msrPedal : public msrMeasureElement
18 msrInstruments.h:class EXP msrDamp : public msrMeasureElement
19 msrInstruments.h:class EXP msrDampAll : public msrMeasureElement
20 msrKeys.h:class EXP msrKey : public msrMeasureElement
21 msrLyrics.h:class EXP msrSyllable : public msrMeasureElement
22 msrMusicXMLSpecifics.h:class EXP msrPrintLayout : public msrMeasureElement
23 msrRehearsalMarks.h:class EXP msrRehearsalMark : public msrMeasureElement
24 msrSegnos.h:class EXP msrSegno : public msrMeasureElement
25 msrDalSegnos.h:class EXP msrDalSegno : public msrMeasureElement
26 msrStavesDetails.h:class EXP msrStaffDetails : public msrMeasureElement
27 msrTempos.h:class EXP msrTempo : public msrMeasureElement
28 msrTimeSignatures.h:class EXP msrTimeSignature : public msrMeasureElement
29 msrTranspositions.h:class EXP msrOctaveShift : public msrMeasureElement
30 msrTranspositions.h:class EXP msrTransposition : public msrMeasureElement
31 msrVoiceStaffChanges.h:class EXP msrVoiceStaffChange : public msrMeasureElement
```

In order to perform a time-wise analysis of the scores, MSR contains class `msrmeasure` linear flat lists, without the class `msrRepeat` and such being represented.

This is used when identifying rest notes that are not 'heard' simultaneously with other notes or rests: this way, the rest can ignore the current voice number and be placed in the vertical middle of the staff.

Apart from the cloning methods, only one method creates measures, namely method `msrSegment::createAMeasureAndAppendItToSegment ()`, defined in `src/formats/msr/msrSegments.h/.cpp`:

```

1 S_msrMeasure msrSegment::createAMeasureAndAppendItToSegment (
2     int     inputLineNumber,
3     std::string measureNumber,
4     msrMeasureImplicitKind
5         measureImplicitKind)
6 {
7     // ... ..
8
9     ++gIndenter;
10
11    // determine new measure 'first in segment' kind
12    msrMeasureFirstInSegmentKind
13        measureFirstInSegmentKind;
14
15    if (fSegmentElementsList.size () == 0) {
16        // this is the first measure in the segment
17        measureFirstInSegmentKind =
18            msrMeasureFirstInSegmentKind::kMeasureFirstInSegmentKindYes;
19    }
20    else {
21        // this is not the first measure in the segment
22        measureFirstInSegmentKind =
23            msrMeasureFirstInSegmentKind::kMeasureFirstInSegmentKindNo;
24    }
25
26    // create a measure
27    // ... ..
28
29    S_msrMeasure
30        result =
31        msrMeasure::create (
32            inputLineNumber,
33            measureNumber,
34            this);
35
36    // set result's ordinal number
37    result->
38        setMeasureOrdinalNumberInVoice (
39            fSegmentUpLinkToVoice->
40            incrementVoiceCurrentMeasureOrdinalNumber ());
41
42    // append result to the segment
43    appendMeasureToSegment (result);
44
45    --gIndenter;
46
47    return result;
48 }

```

20.30 Repeats patterns and replicas

MSR represents repeated beats and measures this way:

- a pattern describes what is repeated;
- there are as many replicas of the music as needed.

This leads to:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep Pattern *.h | grep
  class
2 msrBeatRepeats.h:class EXP msrBeatRepeatPattern : public msrElement
3 msrMeasureRepeats.h:class EXP msrMeasureRepeatPattern : public msrElement
4 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep Replicas *.h | grep
  class
5 msrBeatRepeats.h:class EXP msrBeatRepeatReplicas : public msrElement
6 msrMeasureRepeats.h:class EXP msrMeasureRepeatReplicas : public msrElement

```

These two repeat cases are described in the sections below.

20.31 Beat repeats

Beat repeats handling is presented at section 45 [Beat repeats handling], page 321.

Class `msrBeatRepeat`, defined in `src/formats/msr/msrBeatRepeats.h/.cpp`, contains a pattern and replicas:

```

1 class EXP msrBeatRepeat : public msrVoiceElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrVoice          fUpLinkToBeatRepeatToVoice;
12
13        // numbers
14        int                 fBeatRepeatMeasuresNumber;
15        int                 fBeatRepeatSlashesNumber;
16
17        // measures repeat pattern
18        S_msrBeatRepeatPattern
19                            fBeatRepeatPattern;
20
21        // measures repeat replicas
22        S_msrBeatRepeatReplicas
23                            fBeatRepeatReplicas;
24
25        // measures repeat build phase, used when building the measures repeat
26        msrBeatRepeatBuildPhaseKind
27                            fCurrentBeatRepeatBuildPhaseKind; // unused??? JMI
28 };

```

Class `msrBeatRepeatPattern` contains a segment and an uplink:

```

1 class EXP msrBeatRepeatPattern : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrBeatRepeat      fUpLinkToBeatRepeat;
12
13        // segment

```



```

14     S_msrSegment                fBeatRepeatPatternSegment;
15 };

```

Class `msrBeatRepeatReplicas` contains a segment and an uplink:

```

1  class EXP msrBeatRepeatReplicas : public msrElement
2  {
3      // ... ..
4
5      private:
6
7          // private fields
8          // -----
9
10         // upLinks
11         S_msrBeatRepeat        fUpLinkToBeatRepeat;
12
13         // segment
14         S_msrSegment           fBeatRepeatReplicasSegment;
15 };

```

20.32 Measure repeats

Measure repeats handling is presented at section 46 [Measure repeats handling], page 322.

Class `msrMeasureRepeat`, defined in `src/formats/msr/msrMeasureRepeat.h/.cpp`, contains a pattern and replicas:

```

1  class EXP msrMeasureRepeat : public msrVoiceElement
2  {
3      // ... ..
4
5      private:
6
7          // private fields
8          // -----
9
10         // upLinks
11         S_msrVoice                fUpLinkToMeasureRepeatToVoice;
12
13         // numbers
14         int                       fMeasureRepeatMeasuresNumber;
15         int                       fMeasureRepeatSlashesNumber;
16
17         // measures repeat pattern
18         S_msrMeasureRepeatPattern fMeasureRepeatPattern;
19
20
21         // measures repeat replicas
22         S_msrMeasureRepeatReplicas fMeasureRepeatReplicas;
23
24
25         // measures repeat build phase, used when building the measures repeat
26         msrMeasureRepeatBuildPhaseKind fCurrentMeasureRepeatBuildPhaseKind;
27 };
28

```

Class `msrMeasureRepeatPattern` contains a segment and an uplink:

```

1 class EXP msrMeasureRepeatPattern : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrMeasureRepeat    fUpLinkToMeasureRepeat;
12
13        // segment
14        S_msrSegment          fMeasureRepeatPatternSegment;
15 };

```

Class `msrMeasureRepeatReplicas` contain a segment and an uplink:

```

1 class EXP msrMeasureRepeatReplicas : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrMeasureRepeat    fUpLinkToMeasureRepeat;
12
13        // segment
14        S_msrSegment          fMeasureRepeatReplicasSegment;
15 };

```

20.33 Multi-measure rests

Multi-measure rests handling is presented at section 47 [Multi-measure rests handling], page 323.

Class `msrMultiMeasureRest`, defined in `src/formats/msr/msrMultiMeasureRest.h/.cpp`, essentially contains a list of class `_msrMeasure` instances and a multi-measure rests number:

```

1 class EXP msrMultiMeasureRest : public msrSegmentElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        S_msrSegment          fMultiMeasureRestUpLinkToSegment;
11
12        int                   fMeasureRestsNumber; // supplied by MusicXML
13        std::list<S_msrMeasure> fMeasureRestsList;
14
15
16        int                   fLastMeasureRestPuristNumber;
17
18        std::string           fNextMeasureNumber;
19 };

```

20.34 Barlines

20.35 Repeats

Repeats handling is presented at section 48 [Repeats handling], page 324.

Contrary to MusicXML, MusicFormats represents the full structure of repeated music, not just barlines.

The following classes are defined in `src/formats/msr/msrRepeats.h/.cpp`, contains:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep class    msrRepeats.h
2 class    msrRepeat;
3 class    msrMultiMeasureRest;
4 class    msrMeasureRepeat;
5 class    msrNote;
6 class EXP msrRepeatCommonPart : public msrElement
7 class EXP msrRepeatEnding : public msrElement
8 class EXP msrRepeat : public msrVoiceElement
9 class EXP msrRepeatDescr : public smartable
10 class EXP msrRepeatElement : public msrElement
```

Class `msrRepeat`, defined in `msrBothmsrRepeats`, contains an class `msrRepeatCommonPart`, followed by zero or more instances of class `msrRepeatEnding`:

```
1 class EXP msrRepeat : public msrVoiceElement
2 {
3     public:
4
5         // data types
6         // -----
7
8         enum class msrRepeatExplicitStartKind {
9             kRepeatExplicitStartNo,
10            kRepeatExplicitStartYes
11        };
12
13        // ... ..
14
15        // common part
16        S_msRepeatCommonPart fRepeatCommonPart;
17
18        // repeat endings
19        std::vector<S_msRepeatEnding>
20            fRepeatEndings;
21        int fRepeatEndingsInternalCounter;
22
23        // immediately preceding and following repeats
24        // detecting several repeats in a row helps LilyPond code generation
25        // depending on the options JMI
26        S_msRepeat fImmediatelyPrecedingRepeat;
27        S_msRepeat fImmediatelyFollowingRepeat;
28    };
```

Class `msrRepeatCommonPart` contains a list of class `msrVoiceElement`:

```
1     private:
2
3         // private fields
4         // -----
5
6         // upLinks
7         S_msRepeat fRepeatCommonPartUpLinkToRepeat;
8
9         // elements list
```

```

10 std::list<S_msrVoiceElement>
11     fRepeatCommonPartElementsList;

```

Enumeration type `msrRepeatEndingKind` is used to distinguish hooked and hookless repeat endings: hookless when the ending is simply overlined, and hooked when there a vertical line at the end of the ending's overline:

```

1 enum class msrRepeatEndingKind {
2     kRepeatEndingHooked,
3     kRepeatEndingHookless
4 };

```

Class `msrRepeatEnding` contains a list of class `msrVoiceElement` too, as well as a enumeration type `msrRepeatEndingKind` field:

```

1 private:
2
3     // private fields
4     // -----
5
6     // upLinks
7     S_msrRepeat          fRepeatEndingUpLinkToRepeat;
8
9     // numbers
10    std::string           fRepeatEndingNumber; // may be "1, 2"
11    int                   fRepeatEndingInternalNumber; // internally assigned
12
13    // kind
14    msrRepeatEndingKind   fRepeatEndingKind;
15
16    // elements list
17    std::list<S_msrVoiceElement>
18        fRepeatEndingElementsList;

```

20.36 Segments

Segments handling is presented at section 44 [Segments handling], page 315.

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 20.4 [Three segments in a voice], page 212, 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 20.4: Three segments in a voice

20.37 Notes and rests

Class `msrNote` is complex class: it handles many variants, but using classes to represent the variants would be too cumbersome. As shown at figure 20.3 [The MSR classes hierarchy], page 199:

- 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 unpitched.

class `msrNote` thus uses enumeration type enumeration type `msrNoteKind`, defined in `src/formats/msr/msrBasicType` to distinguish them:

```

1  enum class msrNoteKind {
2      kNote_UNKNOWN_,
3
4      // in measures
5      kNoteRegularInMeasure,
6      kNoteRestInMeasure,
7      kNoteSkipInMeasure, // an invisible rest
8      kNoteUnpitchedInMeasure,
9
10     // in chords
11     kNoteRegularInChord,
12
13     // in triplets
14     kNoteRegularInTriplet,
15     kNoteRestInTriplet,
16     kNoteUnpitchedInTriplet,
17
18     // in grace notes groups
19     kNoteRegularInGraceNotesGroup,
20     kNoteSkipInGraceNotesGroup, // used to circumvent LilyPond issue #34
21
22     // in chords in grace notes groups
23     kNoteInChordInGraceNotesGroup,
24
25     // in triplets in grace notes groups
26     kNoteInTripletInGraceNotesGroup,
27
28     // in double-tremolos
29     kNoteInDoubleTremolo
30 };

```

20.38 Grace notes groups

Grace notes groups handling is presented at section 60 [Grace notes groups handling], page 344.

20.39 Chords

A chord contains notes only, and can occur in measures, tuplets and grace notes groups, hence:

```

1 // chords
2 // -----
3
4 enum class msrChordInKind {
5     kChordIn_UNKNOWN_,
6
7     kChordInMeasure,
8     kChordInTuplet,
9     kChordInGraceNotesGroup
10 };

```

20.40 Tuplets

Tuplets handling is presented at section 62 [Tuplets handling], page 346.

A tuplet can contain:

- notes and rests;
- chords;
- other tuplets.

Tuplets can occur in measures and other tuplets, hence enumeration type `msrTupletInKind`:

```

1 enum class msrTupletInKind {
2     kTupletIn_UNKNOWN_,
3
4     kTupletInMeasure,
5     kTupletInTuplet
6 };

```

Tuplets factors are represented by class `msrTupletFactor`, defined in `src/formats/msr/msrBasicTypes.h/.cpp`.

```

1 class EXP msrTupletFactor
2 {
3     // ... ..
4
5     public:
6
7     // public services
8     // -----
9
10    Bool                isEqualToOne () const
11    {
12        return
13            fTupletActualNotes == fTupletNormalNotes;
14    }
15

```

```

16      mfrational          asRational () const
17      {
18          return
19              mfrational (
20                  fTupletActualNotes ,
21                  fTupletNormalNotes);
22      }
23
24      // ... ..
25
26      private:
27
28          // private fields
29          // -----
30
31          int              fTupletActualNotes;
32          int              fTupletNormalNotes;
33  };

```

20.41 Harmonies and figured bass similarities

Harmonies and figured bass handling is presented at section 63 [Harmonies handling], page 347 and section 64 [Figured bass elements handling], page 365, respectively.

In MusicXML, harmonies and figured bass occur at the measure level:

```

1      <harmony print-frame="no">
2          <root>
3              <root-step>C</root-step>
4          </root>
5          <kind text="m">minor</kind>
6      </harmony>
7      <note default-x="75.17" default-y="-35.00">
8          <pitch>
9              <step>F</step>
10             <octave>4</octave>
11         </pitch>
12         <duration>2</duration>
13         <voice>1</voice>
14         <type>quarter</type>
15         <stem>up</stem>
16     </note>

```

```

1      <harmony>
2          <root>
3              <root-step>F</root-step>
4              <root-alter>1</root-alter>
5          </root>
6          <kind>major</kind>
7          <inversion>2</inversion>
8      </harmony>
9      <note>
10         <pitch>
11             <step>C</step>
12             <octave>4</octave>
13         </pitch>
14         <duration>4</duration>
15         <type>whole</type>
16     </note>

```

In MSR, the instances of class `msrHarmony` and class `msrFiguredBass` are present twice:

- each class `msrNote` instance contains the harmonies and figured bass attached to it:

```

1 class EXP msrNote : public msrTupletElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // harmonies
11        // -----
12
13        std::list<S_msrHarmony>
14            fNoteHarmoniesList;
15
16        // figured bass
17        // -----
18
19        std::list<S_msrFiguredBass>
20            fNoteFiguredBassesList;
21
22        // ... ..
23 };

```

- each class `msrPart` instance contains a harmonies staff and voice, as well as a figured bass staff and voice:

```

1 class EXP msrPart : public msrPartGroupElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9         // harmonies
10
11        S_msrStaff          fPartHarmoniesStaff;
12        S_msrVoice         fPartHarmoniesVoice;
13
14        // figured bass
15
16        S_msrStaff          fPartFiguredBassStaff;
17        S_msrVoice         fPartFiguredBassVoice;
18
19        // ... ..
20 };

```

The way harmonies and figured bass elements are represented in MusicFormats is presented in the next two sections.

20.42 Harmonies

Harmonies handling is presented at section [63](#) [Harmonies handling], page [347](#).

20.43 Figured bass

Figured bass elements handling is presented at section [64](#) [Figured bass elements handling], page [365](#).

20.44 Lyrics

Lyrics handling is presented at section [65](#) [Lyrics handling], page [381](#).

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;
- they can be several lyrics stanzas associated to a given staff;
- the syllables in lyrics can apply to more than 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 `msrSyllableKind`:

```

1  enum class msrSyllableKind {
2      kSyllableNone,
3      kSyllableSingle,
4      kSyllableBegin, kSyllableMiddle, kSyllableEnd,
5
6      kSyllableOnRestNote,
7      kSyllableSkipOnRestNote,
8      kSyllableSkipOnNonRestNote,
9
10     kSyllableMeasureEnd,
11     kSyllableLineBreak, kSyllablePageBreak
12 };

```

Extensions are described by enumeration type :

```

1  enum class msrSyllableExtendKind {
2      kSyllableExtendNone,
3      kSyllableExtendEmpty,
4      kSyllableExtendSingle,
5      kSyllableExtendStart, kSyllableExtendContinue, kSyllableExtendStop
6  };

```

Class `msrSyllable` contains:

```

1  // syllable kind
2  msrSyllableKind      fSyllableKind;
3
4  // texts list
5  std::list<std::string>
6      fSyllableElementsList;
7
8  // extend kind
9  msrSyllableExtendKind fSyllableExtendKind;
10
11 // stanza number, may contain non-digits
12 std::string          fSyllableStanzaNumber;
13
14 // syllable whole notes
15 mfrational           fSyllableWholeNotes;
16
17 // syllable tuplet factor
18 msrTupletFactor      fSyllableTupletFactor;

```

Syllables are one case where the data in MSR is denormalized: a given class `msrSyllable` instance belongs both to an class `msrNote` instance and to a lyrics instance of class `msrVoice`.

At the higher level, syllables are organized as instances of class `msrStanza`, which contains:

```
1 // contents
2 std::vector<S_msrSyllable> fSyllables;
3
4 Bool fStanzaTextPresent;
```

20.45 MIDI

MIDI handling is presented at section [66](#) [MIDI handling], page [382](#).

Chapter 21

MSR time-oriented representation

In order to represent the music according to simultaneous sounding time, MSR builds:

- a flat list of measures at the voice and staff levels;
- from this, a vector of measures slices at the voice, staff, part, part group and score levels.

The source files are in `src/formats/msr/msrMeasuresSlices.h/.cpp`.

21.1 Note events

Notes start and stop are represented by enumeration type `msrNoteEventKind`:

```
1 enum class msrNoteEventKind {  
2     kNoteEventStart ,  
3     kNoteEventStop  
4 };
```

A note event is described in class :

```
1 class    msrNoteEvent : public smartable  
2 {  
3     // ... ..  
4  
5     private:  
6  
7         // private fields  
8         // -----  
9  
10        mfRational                fNoteEventMeasurePosition;  
11        S_msrNote                 fNoteEventNote;  
12        msrNoteEventKind          fNoteEventKind;  
13};
```

21.2 Simultaneous notes chunks

Such a chunk is a set of notes or rests played simultaneously, i.e. that start and stop at the same time. The set is stored as a list actually:

```

1 class    msrSimultaneousNotesChunk : public smartable
2 {
3     // ... ..
4
5     private:
6
7     // private fields
8     // -----
9
10    mfrational          fChunkMeasurePosition;
11    std::list<S_msrNote> fChunkNotesList;
12    mfrational          fChunkNotesDurationWholeNotes;
13 };

```

21.3 Measures slices

A measures slice, described by class `msrMeasuresSlice`, is a 'vertical' cut in the score across voices: it contains all the measures starting at the same time, one per voice:

```

1 class EXP msrMeasuresSlice : public smartable
2 {
3     // ... ..
4
5     protected:
6
7     // protected fields
8     // -----
9
10    int          fSlicePuristMeasureNumber;
11    std::string  fSliceMeasureNumber;
12
13    // the measures in the slice
14    std::vector<S_msrMeasure> fSliceMeasuresVector;
15
16    // notes flat list
17    std::list<S_msrNote> fSliceNotesFlatList;
18
19    // note events list
20    std::list<S_msrNoteEvent>
21        fSliceNoteEventsList;
22
23    // simultaneous notes chunks list
24    std::list<S_msrSimultaneousNotesChunk>
25        fSliceSimultaneousNotesChunksList;
26 };

```

21.4 Measures slices sequences

A class `msrMeasuresSlicesSequence` contains a vector of `S_msrMeasuresSlice` instances:

```

1 class EXP msrMeasuresSlicesSequence : public smartable
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        std::string          fMeasuresOrigin;
11
12        std::vector<S_msrMeasuresSlice>
13                                fMeasuresSlicesVector;
14 };

```

A smart pointer to an `msrMeasuresSlicesSequence` instance is stored in `msrVoice`, `msrStaff`, `msrPart`, `msrPartGroup` and `msrScore`.

21.5 Building the measures slices

21.5.1 Part measures slices

At the part level, this is done in method `msrPart::collectPartMeasuresSlices ()`:

```

1 void msrPart::collectPartMeasuresSlices (
2     int inputLineNumber)
3 {
4     // ... ..
5
6     // create the part measures slices sequence
7     fPartMeasuresSlicesSequence =
8         msrMeasuresSlicesSequence::create (
9             fPartName); // origin
10
11    // populate it
12    for (S_msrStaff staff : fPartAllStavesList) {
13        // ... ..
14
15        ++gIndenter;
16
17        S_msrMeasuresSlicesSequence
18            staffMeasuresSlicesSequence =
19            staff->
20                getStaffMeasuresSlicesSequence ();
21
22        if (! staffMeasuresSlicesSequence) {
23            std::stringstream ss;
24
25            ss <<
26                "The staffMeasuresSlicesSequence of staff \"" <<
27                staff->getStaffName () <<
28                "\" is null";
29
30            musicxmlWarning (
31                gServiceRunData->getInputSourceName (),
32                inputLineNumber,
33                ss.str ());
34        }
35        else {
36            fPartMeasuresSlicesSequence->
37                mergeWithMeasuresSlicesSequence (
38                    inputLineNumber,

```

```

39         getPartCombinedName (),
40         staffMeasuresSlicesSequence);
41     }
42
43     --gIndenter;
44 } // for
45
46 // ... ..
47 }

```

21.5.2 Staff measures slices

Method `msrStaff::collectStaffMeasuresSlices ()` builds them:

```

1 void msrStaff::collectStaffMeasuresSlices (
2     int inputLineNumber)
3 {
4     // ... ..
5
6     // create the staff measures slices sequence
7     fStaffMeasuresSlicesSequence =
8         msrMeasuresSlicesSequence::create (
9             fStaffName); // origin
10
11     // populate it
12     for (const S_msrVoice& voice : fStaffAllVoicesList) {
13         // ... ..
14
15         // get the voice measures slices sequence
16         S_msrMeasuresSlicesSequence
17             voiceMeasuresSlicesSequence =
18             voice->
19                 getVoiceMeasuresSlicesSequence ();
20
21         // merge it with the voice measures slices sequence
22         if (voiceMeasuresSlicesSequence) { // JMI
23             fStaffMeasuresSlicesSequence =
24                 fStaffMeasuresSlicesSequence->
25                     mergeWithMeasuresSlicesSequence (
26                         inputLineNumber,
27                         fStaffName,
28                         voiceMeasuresSlicesSequence);
29         }
30
31         // identify the solo notes and rests in the staff
32         fStaffMeasuresSlicesSequence->
33             identifySoloNotesAndRests ();
34
35         --gIndenter;
36     } // for
37
38     // ... ..
39 }

```

21.6 Solo notes and rests

A solo note or rest is one that occurs alone at some point in time for its whole duration, without any other note being played at the same time.

Identifying such solo notes or rests is done in method `msrMeasuresSlicesSequence::identifySoloNotesAndRests` using the measures slices of the staff they occur in, called method `msrStaff::collectStaffMeasuresSlices` () as shown above:

```
1 void msrMeasuresSlicesSequence::identifySoloNotesAndRests ()
2 {
3     // ... ..
4
5     // collect the notes from the sequence's measures slices
6     for (
7         std::vector<S_msrMeasuresSlice>::const_iterator i =
8             fMeasuresSlicesVector.begin ();
9         i != fMeasuresSlicesVector.end ();
10        ++i
11    ) {
12        S_msrMeasuresSlice measuresSlice = (*i);
13
14        measuresSlice->
15            collectNonSkipNotesFromMeasuresSliceMeasures ();
16    } // for
17 }
```

21.7 A measures slices example

Chapter 22

Path to voice

`src/formats/msr/msrPathToVoice.h.h/.cpp` defines class `msrPathToVoice`, used to create partial clones of class `msrBook` retaining only certain staves and/or voices, or to create new class `msrScore` instances containing each of them only:

```
1 class EXP msrPathToVoice : public smartable
2 {
3     // ... ..
4
5     public:
6
7         // public services
8         // -----
9
10        void                appendPartGroup (const S_msrPartGroup& partGroup)
11                                {
12                                    fPartGroupsList.push_back (partGroup);
13                                }
14
15        // ... ..
16
17        private:
18
19            // private fields
20            // -----
21
22            S_msrBook        fBook;
23
24            S_msrScore        fScore;
25
26            // part groups can be nested
27            std::list<S_msrPartGroup>  fPartGroupsList;
28
29            S_msrPart        fPart;
30
31            S_msrStaff        fStaff;
32
33            S_msrVoice        fVoice;
34 };
```


Chapter 23

LilyPond Scores Representation (LPSR)

An LPSR description contains two components:

- the first one is an MSR, thus the whole music score description;
- the second one is a description of the structure of the score mirroring LilyPond's specific blocks such as `\book {...}` and `\layout {...}`.

Class `lpsrScore` thus contains:

```
1  // MSR data
2  S_msrScore          fMsrScore;
3
4  // ... ..
5
6  // LilyPond stuff
7  S_lpsrHeader        fScoreHeader;
8  S_lpsrPaper         fScorePaper;
9  S_lpsrLayout        fScoreLayout;
10
11 // variables, voices and stanzas
12 std::list<S_msrElement>
13         fScoreElementsList;
14
15 // score LPSR book blocks list
16 std::list<S_lpsrBookBlock> fScoreBookBlocksList;
17 S_lpsrScoreBlock        fScoreScoreBlock; // JMI ???
```

23.1 LPSR basic types

Some types used throughout LSPR are defined in `src/formats/lpsr/lpsrEnumTypes.h/.cpp`:

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/lpsr >  egrep -rIn  '^// '
    lpsrEnumTypes.h
2  lpsrEnumTypes.h:28://  score output kinds
3  lpsrEnumTypes.h:50://  accidental styles
4  lpsrEnumTypes.h:53://  JMI there are new ones in LilyPond 2.22
5  lpsrEnumTypes.h:87://  chords languages
6  lpsrEnumTypes.h:107:// whole notes
7  lpsrEnumTypes.h:118:// dotted durations
8  lpsrEnumTypes.h:129:// rests measures
9  lpsrEnumTypes.h:135:// texts lists
10 lpsrEnumTypes.h:141:// pitches and octaves
11 lpsrEnumTypes.h:151:// lyrics durations
12 lpsrEnumTypes.h:168:// initialization
```

23.2 Adapting LilyPond code generation to the target version number

As of version 2.22, `compressMultiMeasureRests` has been replaced by `compressMeasureRests` for clarity.

Such is done specific methods:



Chapter 24

Braille Scores Representation (BSR)

BSR represents braille scores as composed of lines of 6-dot cells.

24.1 BSR basic types

Some types used throughout BSR are defined in [src/formats/bsr/bsrEnumTypes.h/.cpp](#):

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/bsr > egrep -rIn '~/' '
    bsrEnumTypes.h
2 bsrEnumTypes.h:23:// cell kinds
3 bsrEnumTypes.h:107:// lower case letters
4 bsrEnumTypes.h:139:// capitals
5 bsrEnumTypes.h:143:// kCellCapitalsSequenceSign, // { kCellCapitalsSign,
    kCellCapitalsSign };
6 bsrEnumTypes.h:145:// decimal digits
7 bsrEnumTypes.h:160:// lower decimal digits
8 bsrEnumTypes.h:174:// alterations
9 bsrEnumTypes.h:181:// augmentation dots
10 bsrEnumTypes.h:186:// arithmetic operators
11 bsrEnumTypes.h:195:// words
12 bsrEnumTypes.h:205:// braille cells
13 bsrEnumTypes.h:212:// braille output kinds
14 bsrEnumTypes.h:231:// chords languages
15 bsrEnumTypes.h:251:// brailling numbers
16 bsrEnumTypes.h:255:// brailling characters and strings
17 bsrEnumTypes.h:261:// writing UTF-16 to ostream
18 bsrEnumTypes.h:273:// initialization
19 bsrEnumTypes.h:971:// constants
20 bsrEnumTypes.h:975:// computations
```

24.2 Representing cells

This is done basically with enumeration type enumeration type `bsrCellKind`, defined in [src/formats/bsr/bsrEnumTy](#)

```
1 // cell kinds
2 // -----
3 enum class bsrCellKind {
4     kCellUnknown,
5
6     // non 6dots values
7     kCellEOL      , // L'\u000a'
8     kCellEOP      , // L'\u000c'
9
10    // 6dots values for Braille
```

```

11 kDotsNone      , // L'\u2800'
12 kDots1         , // L'\u2801'
13 kDots2         , // L'\u2802'
14
15 // ... ..
16
17 kDots23456      , // L'\u283e'
18 kDots123456     // L'\u283f'
19 };

```

Enumeration type `bsrCellKind` constants are not used throughout the code base: instead, there are enumeration type `bsrCellKind` constants to provide context-specific names for the cells kinds.

Lower-case letters:

```

1 // lower case letters
2 // -----
3 const bsrCellKind
4     kCellA = bsrCellKind::kDots1,
5     kCellB = bsrCellKind::kDots12,
6
7
8     kCellY = bsrCellKind::kDots13456,
9     kCellZ = bsrCellKind::kDots1356;

```

Capital sign:

```

1 // capitals
2 // -----
3 const bsrCellKind
4     kCellCapitalsSign = bsrCellKind::kDots46;

```

Decimal digits:

```

1 // decimal digits
2 // -----
3 const bsrCellKind
4     kCellNumberSign = bsrCellKind::kDots3456,
5     kCell11 = kCellA,
6     kCell12 = kCellB,
7     kCell13 = kCellC,
8     kCell14 = kCellD,
9     kCell15 = kCellE,
10    kCell16 = kCellF,
11    kCell17 = kCellG,
12    kCell18 = kCellH,
13    kCell19 = kCellI,
14    kCell10 = kCellJ;

```

Alterations:

```

1 // alterations
2 // -----
3 const bsrCellKind
4     kCellFlat      = bsrCellKind::kDots126,
5     kCellNatural   = bsrCellKind::kDots16,
6     kCellSharp     = bsrCellKind::kDots146;

```

Augmentation dots:

```

1 // augmentation dots
2 // -----
3 const bsrCellKind
4     kCellAugmentationDot = bsrCellKind::kDots3;

```

Arithmetic operators:

```
1 // arithmetic operators
2 // -----
3 const bsrCellKind
4     kCell_ac_plus      = bsrCellKind::kDots235,
5     kCell_ac_minus     = bsrCellKind::kDots36,
6     kCell_ac_times     = bsrCellKind::kDots35,
7     kCell_ac_dividedBy = bsrCellKind::kDots25,
8     kCell_ac_equals    = bsrCellKind::kDots2356;
```

Words:

```
1 // words
2 // -----
3 const bsrCellKind
4     kCellWordSign      = bsrCellKind::kDots345,
5
6     kCellWordApostrophe = bsrCellKind::kDots6,
7
8     kCellParenthesis    = bsrCellKind::kDots2356,
9     kCellQuestionMark   = bsrCellKind::kDots26;
```

Chapter 25

MusicXML Scores Representation (MXSR)

This format is provided by `libmusicxml2`, even though Dominique Fober didn't give it that name. It is a tree of class `mxmlelement` nodes, mapped one to one to the MusicXML markups.

The files in `libmusicxml/src`.

A set of interface functions is contained in `src/formats/mxsr/mxsr.h/.cpp`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/mxsr/
2 total 48
3  0 drwxr-xr-x   6 jacquesmenu  staff    192 May 26 08:20:55 2021 ./
4  0 drwxr-xr-x  10 jacquesmenu  staff    320 Jun 25 05:39:49 2021 ../
5  8 -rw-r--r--@   1 jacquesmenu  staff  3292 Jun  6 06:35:19 2021 mxsr.cpp
6  8 -rw-r--r--@   1 jacquesmenu  staff  1555 Jun  6 06:35:19 2021 mxsrGeneration.h
7 16 -rw-r--r--@   1 jacquesmenu  staff  7781 Jun  6 06:35:19 2021 mxsr0ah.cpp
8 16 -rw-r--r--@   1 jacquesmenu  staff  4829 Jun  6 06:35:19 2021 mxsr0ah.h
```

25.1 MusicXML elements and attributes

MusicXML data contains so-called elements, written as `<... />` markups, that can be nested:

```
1 <system-margins>
2   <left-margin>15</left-margin>
3   <right-margin>0</right-margin>
4 </system-margins>
```

In the example above, the values of the two margins are 15 and 0, respectively.

MusicXML elements can have attributes, such as `version` below:

```
1 <score-partwise version="3.1">
```

The values of the elements and attributes are strings.

There are two special elements at the beginning of MusicXML data:

- a `<?xml/>` element indicating the characters encoding used;
- a `<“!”DOCTYPE/>` element telling that the contents is in 'score-partwise' mode and containing the URL of the DTD.

An exemple is:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 2.0 Partwise//EN"
3 "http://www.musicxml.org/dtds/partwise.dtd">

```

25.2 The xmlelement and xmlattribute types

xmlelementxmlattribute

These two classes are defined in `libmusicxml/src/elements!xml.h/.cpp`:

```

1 class xmlelement;
2 class xmlattribute;
3
4 typedef SMARTP<xmlattribute> Sxmlattribute;
5 typedef SMARTP<xmlelement> Sxmlelement;

```

class xmlattribute contains:

```

1 //-----
2 class EXP xmlattribute : public smartable {
3     //! the attribute name
4     std::string fName;
5     //! the attribute value
6     std::string fValue;
7
8     // ... ..
9
10    //! returns the attribute value as a int
11    operator int () const;
12    //! returns the attribute value as a long
13    operator long () const;
14    //! returns the attribute value as a float
15    operator float () const;

```

class xmlelement contains:

```

1 class EXP xmlelement : public ctree<xmlelement>, public visitable
2 {
3     private:
4         //! the element name
5         std::string fName;
6         //! the element value
7         std::string fValue;
8         //! list of the element attributes
9         std::vector<Sxmlattribute> fAttributes;
10
11     protected:
12         // the element type
13         int fType;
14         // the input line number for messages to the user
15         int fInputStartLineNumber;
16
17         // ... ..
18
19         //! returns the element value as a long
20         operator long () const;
21         //! returns the element value as a int
22         operator int () const;
23         //! returns the element value as a float
24         operator float () const;
25         //! elements comparison

```

```

26 Bool operator ==(const xmlelement& elt) const;
27 Bool operator !=(const xmlelement& elt) const { return !(*this == elt); }
28
29 ///! adds an attribute to the element
30 long add (const Sxmlattribute& attr);
31
32 // ... ..
33 };

```

Type Sxmlelement is a smart pointer to an xmlelement, so it is an xmlelement tree, since xmlelement is a recursive type.

fInputStartLineNumber 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.

25.3 Enumeration types for xmlelement's fType

xmlelement

libmusicxml2 uses elements/templates/elements.bash, a Bash script, to generate the enumeration type constants and classes source code from the MusicXML DTD. This is not done in the Makefile, since it is to be run by hand only once.

The DTD files we use as reference are in libmusicxml/dtds/3.1/schema;

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/libmusicxml/dtds/3.1/schema > ls -sal *.mod
2 40 -rwxr-xr-x@ 1 jacquesmenu staff 20238 Apr 22 15:49 attributes.mod
3 16 -rwxr-xr-x 1 jacquesmenu staff 4943 Apr 22 15:49 barLine.mod
4 80 -rwxr-xr-x@ 1 jacquesmenu staff 37932 Apr 22 15:49 common.mod
5 88 -rwxr-xr-x@ 1 jacquesmenu staff 41960 Apr 22 15:49 direction.mod
6 16 -rwxr-xr-x@ 1 jacquesmenu staff 4097 Apr 22 15:49 identity.mod
7 24 -rwxr-xr-x@ 1 jacquesmenu staff 10266 Apr 22 15:49 layout.mod
8 8 -rwxr-xr-x@ 1 jacquesmenu staff 2833 Apr 22 15:49 link.mod
9 104 -rwxr-xr-x@ 1 jacquesmenu staff 51384 Apr 22 15:49 note.mod
10 32 -rwxr-xr-x@ 1 jacquesmenu staff 15476 Apr 22 15:49 score.mod

```

The first result of running libmusicxml/src/elements/templates!elements.bash is an anonymous enumeration type defined in libmusicxml/src/elements!elements.h:

```

1 enum class {
2     kNoElement,
3     kComment,
4     kProcessingInstruction,
5     k_accent,
6     k_accidental,
7     k_accidental_mark,
8     k_accidental_text,
9
10    // ... ..
11
12     k_work,
13     k_work_number,
14     k_work_title,
15     kEndElement
16 };

```

The constants kNoElement, kComment and kProcessingInstruction are added by elements.bash.

25.4 Classes for the xmlelements

All the MusicXML classes are instantiated from the `musicxml` template class, defined in `libmusicxml/src/elements!`. This is where `fType` gets its value:

```
1 template <int elt> class musicxml : public xmlelement
2 {
3     protected:
4         musicxml (int inputLineNumber) : xmlelement (inputLineNumber) { fType = elt; }
5 };
```

The smart pointer `s` to the various elements are defined in `libmusicxml/src/elements!typedef.h`, using an anonymous enumeration type :

```
1 typedef SMARTP<musicxml<kComment> > S_comment;
2 typedef SMARTP<musicxml<kProcessingInstruction> > S_processing_instruction;
3
4 typedef SMARTP<musicxml<k_accent> > S_accent;
5 typedef SMARTP<musicxml<k_accidental> > S_accidental;
6 typedef SMARTP<musicxml<k_accidental_mark> > S_accidental_mark;
7 typedef SMARTP<musicxml<k_accidental_text> > S_accidental_text;
8
9 // ... ..
10
11 typedef SMARTP<musicxml<k_work> > S_work;
12 typedef SMARTP<musicxml<k_work_number> > S_work_number;
13 typedef SMARTP<musicxml<k_work_title> > S_work_title;
```

The two-way correspondance of MusicXML elements names to type `Sxmlelement` is stored `fMap` and `fType2Name`, defined in `libmusicxml/src/elements!factory.h`:

```
1 class EXP factory : public singleton<factory>{
2
3     std::map<std::string, functor<Sxmlelement>*> fMap;
4     std::map<int, const char*> fType2Name;
5
6     // ... ..
7 };
```

Those two maps are initialized in `libmusicxml/samples/elements/factory.cpp`:

```
1 factory::factory()
2 {
3     fMap["comment"] = new newElementFunctor<kComment>;
4     fMap["pi"] = new newElementFunctor<kProcessingInstruction>;
5     fType2Name[kComment] = "comment";
6     fType2Name[kProcessingInstruction] = "pi";
7
8     fMap["accent"] = new newElementFunctor<k_accent>;
9     fMap["accidental"] = new newElementFunctor<k_accidental>;
10    fMap["accidental-mark"] = new newElementFunctor<k_accidental_mark>;
11    fMap["accidental-text"] = new newElementFunctor<k_accidental_text>;
12
13    // ... ..
14
15    fMap["work"] = new newElementFunctor<k_work>;
16    fMap["work-number"] = new newElementFunctor<k_work_number>;
17    fMap["work-title"] = new newElementFunctor<k_work_title>;
18
19    fType2Name[k_accent] = "accent";
20    fType2Name[k_accidental] = "accidental";
21    fType2Name[k_accidental_mark] = "accidental-mark";
22    fType2Name[k_accidental_text] = "accidental-text";
23 }
```

```

24 // ... ..
25
26 fType2Name[k_work] = "work";
27 fType2Name[k_work_number] = "work-number";
28 fType2Name[k_work_title] = "work-title";
29 }

```

class `newElementFunctor` is defined in `to` to provide call operator as:

```

1 template<int elt>
2 class newElementFunctor : public functor<Sxmlelement>
3 {
4     public:
5
6     Sxmlelement operator ()()
7     { return musicxml<elt>::new_musicxml (libmxmllineneno); }
8 };

```

25.5 xmlelement trees

This section describes features supplied by `libmusicxml2`.

An `xmlelement` is the basic brick to represent a MusicXML element.

Smart pointer type `SXMLFile` is defined in `libmusicxml/src/xmlfile.h`:

```

1 // -----
2 class EXP TXMLFile : public smartable
3 {
4     private:
5         TXMLDecl*      fXMLDecl;
6         TDocType*      fDocType;
7         Sxmlelement    fXMLTree;
8
9     protected:
10         TXMLFile () : fXMLDecl(0), fDocType(0) {}
11         virtual ~TXMLFile () { delete fXMLDecl; delete fDocType; }
12
13     public:
14         static SMARTP<TXMLFile> create();
15
16     public:
17         TXMLDecl*      getXMLDecl ()      { return fXMLDecl; }
18         TDocType*      getDocType ()      { return fDocType; }
19         Sxmlelement    elements ()        { return fXMLTree; }
20
21         void           set (Sxmlelement root) { fXMLTree = root; }
22         void           set (TXMLDecl * dec) { fXMLDecl = dec; }
23         void           set (TDocType * dt) { fDocType = dt; }
24
25         void           print (std::ostream& s);
26 };
27 typedef SMARTP<TXMLFile> SXMLFile;

```

25.5.1 Creating xmlelement trees from textual data

Reading MusicXML data creates instances of `xmlelement`. This is done by an instance of `xmlreader`, defined in `libmusicxml/src/xmlreader.h/.cpp`, which provides methods:

```
1 SXMLFile readbuff(const char* file);
2 SXMLFile read(const char* file);
3 SXMLFile read(FILE* file);
```

These three functions are defined this way:

```
1 // -----
2 SXMLFile xmlreader::readbuff(const char* buffer)
3 {
4     fFile = TXMLFile::create();
5     debug("read buffer", '-');
6     return readbuffer (buffer, this) ? fFile : 0;
7 }
8
9 // -----
10 SXMLFile xmlreader::read(const char* file)
11 {
12     fFile = TXMLFile::create();
13     debug("read", file);
14     return readfile (file, this) ? fFile : 0;
15 }
16
17 // -----
18 SXMLFile xmlreader::read(FILE* file)
19 {
20     fFile = TXMLFile::create();
21     return readstream (file, this) ? fFile : 0;
22 }
```

25.5.2 Printing xmlelement trees

An `xmlelement` can be printed by function `printMxsr ()`, defined in `src/formats/mxsr/mxsr.h/.cpp`:

```
1 void printMxsr (const Sxmlelement theMxsr, std::ostream& os)
2 {
3     xmlvisitor v (os);
4     tree_browser<xmlelement> browser (&v);
5     browser.browse (*theMxsr);
6 }
```

This how MusicXML and Guido output are generated.

25.6 The SXMLFile type

`SXMLFile` is defined in `libmusicxml/src/factory!musicxmlfactory.h` as a smart pointer to class `TXMLFile`:

```
1 // -----
2 class EXP TXMLFile : public smartable
3 {
4     private:
5         TXMLDecl*          fXMLDecl;
6         TDocType*          fDocType;
7         Sxmlelement        fXMLTree;
8
9     protected:
```

```

10     TXMLFile () : fXMLDecl(0), fDocType(0) {}
11     virtual ~TXMLFile () { delete fXMLDecl; delete fDocType; }
12
13     public:
14         static SMARTP<TXMLFile> create();
15
16     public:
17         TXMLDecl*      getXMLDecl ()      { return fXMLDecl; }
18         TDocType*      getDocType ()      { return fDocType; }
19         Sxmlelement    elements ()        { return fXMLTree; }
20
21         void            set (Sxmlelement root) { fXMLTree = root; }
22         void            set (TXMLDecl * dec)  { fXMLDecl = dec; }
23         void            set (TDocType * dt)   { fDocType = dt; }
24
25         void            print (std::ostream& s);
26 };
27 typedef SMARTP<TXMLFile> SXMLFile;

```

fXMLDecl describes the <?xml/> element and fDocType contains the <"!DOCTYPE"/> element.

Part VIII

Passes

Chapter 26

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.

26.1 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.

26.1.1 MusicXML coverage

`src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.h/.cpp` and `src/passes/msr2mxsr/msr2mxsrTranslator.h/.cpp` handle most of the MusicXML version 3.1 elements. Some of them are handled by both sub-passes, such as `S_voice`, `S_measure` and `S_harmony`.

Among the elements that MusicFormats does not handle are the ones for which there is no occurrence in the corpus in folder `files/musicxml`, such as `beat-unit-tied` and `metronome-tied`.

The elements that are new in MusicXML version 4.0 are not known nor handled yet as of this writing.

26.1.2 Backup and forward handling

These markups move the 'MusicXML drawing point' left and right. Class `mxsr2msrTranslator` does this this way:

26.2 Translating an MXSR to an MSR

This is done by class `mxsr2msrTranslator`.

26.3 Translating an MSR to an MXSR

26.4 Translating an MSR to another MSR

Such translation is meant to offer an opportunity to modify the score's description depending on options.

26.5 Translating an MSR to an LPSR

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

26.6 Translating an LPSR to LilyPond code

There are two visiting trace options for the generation of LilyPond code, one for its MSR component, and the other one for its LPSR own part:

```
1 // %%JMI      Bool      fGenerateMsrVisitingInformation;  
2      Bool      fGenerateLpsrVisitingInformation;
```

26.7 Translating an MSR to an BSR

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

26.8 Translating a BSR to another BSR

26.9 Translating an MXSR to Guido

Chapter 27

LilyPond code generation

LilyPond code is produced on standard output, unless options option `-output-file-name`, `-obj` or option `-auto-output-file-name`, `-aofn` are used.

27.1 Basic principle

Lilypond generation is done in `src/passes/lpsr2lilypond/lpsr2lilypondTranslator.h/.cpp`.

Class `lpsr2lilypondTranslator` provides:

```
1 public:
2
3         lpsr2lilypondTranslator (
4             const S_msrOahGroup& msrOpts,
5             const S_lpsrOahGroup& lpsrOpts,
6             std::ostream&         lilypondCodeStream);
7
8     virtual          ~lpsr2lilypondTranslator ();
9
10    void              translateLpsrToLilypondCode (
11        const S_lpsrScore& lpsrScore);
```

It contains these fields among others:

```
1 private:
2
3     // options
4     // -----
5
6     S_msrOahGroup      fMsrOahGroup;
7     S_lpsrOahGroup     fLpsrOahGroup;
8
9     // the LPSR score we're visiting
10    // -----
11    S_lpsrScore         fVisitedLpsrScore;
12
13    // the output stream
14    // -----
15
16    std::ostream&       fLilypondCodeStream;
```


27.2 Generating Scheme functions in the LilyPond output

xml2ly can generate Scheme code that is used by the LilyPond code it generates. This is described in class `lpsrScore` by a number of `*IsNeeded` fields, such as:

```
1 // files includes
2 Bool                fJianpuFileIncludeIsNeeded;
3
4 // Scheme modules
5 Bool                fScmAndAccregSchemeModulesAreNeeded;
6
7 // Scheme functions
8 Bool                fTongueSchemeFunctionIsNeeded;
```

Chapter 28

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.

28.1 Basic principle

Lilypond generation is done in `src/passes/bsr2braille/bsr2brailleTranslator.h/.cpp`.

Class `bsr2brailleTranslator` provides:

```
1 public:
2
3         bsr2brailleTranslator (
4             const S_bsrScore&      bsrScore,
5             const S_bsr0ahGroup& bsrOpts,
6             std::ostream&          brailleOutputStream);
7
8     virtual          ~bsr2brailleTranslator ();
9
10    void              translateBsrToBraille ();
```

It contains these fields among others:

```
1 private:
2
3     // options
4     // -----
5
6     S_bsr0ahGroup      fBsr0ahGroup;
7
8     // the BSR score we're visiting
9     // -----
10
11    S_bsrScore          fVisitedBsrScore;
12
13    // the braille generator used
14    // -----
15
16    S_bsrBrailleGenerator fBrailleGenerator;
17
18    // the output stream
19    // -----
20
21    std::ostream&       fBrailleOutputStream;
```

28.2 Output files name and contents options

he contents options use the following enumeration types:

```

1 enum class bsrUTFKind {
2     kUTF8, kUTF16
3 };
4
5 enum class bsrByteOrderingKind {
6     kByteOrderingNone,
7     kByteOrderingBigEndian, kByteOrderingSmallEndian
8 };

```

xml2brl supplies a option -files options subgroup:

```

1 jacquesmenu@macmini > xml2brl -query files
2 --- Help for subgroup "files" in group "Files group" ---
3 Files group (-files-group):
4 -----
5 Files (-files):
6     -obj, -output-file-name FILENAME
7         Write Braille to file FILENAME instead of standard output.
8     -aofn, -auto-output-file-name
9         This option can only be used when reading from a file.
10        Write MusicXML code to a file in the current working directory.
11        The file name is derived from that of the input file,
12        replacing any suffix after the '.' by 'xml'
13        or adding '.xml' if none is present.
14     -bok, -braille-output-kind OUTPUT_KIND
15        Use OUTPUT_KIND to write the generated Braille to the output.
16        The 4 output kinds available are:
17        ascii, utf16, utf8 and utf8d.
18        'utf8d' leads to every line in the braille score to be generated
19        as a line of cells followed by a line of text showing the contents
20        for debug purposes.
21        The default is 'ascii'.
22     -ueifn, -use-encoding-in-file-name
23        Append a description of the encoding used
24        and the presence of a BOM if any to the file name before the '.'.
25     -bom, -byte-ordering-mark BOM_ENDIAN
26        Generate an initial BOM_ENDIAN byte ordering mark (BOM)
27        ahead of the Braille nusic code,
28        which can be one of 'big' or 'small'.
29        By default, a big endian BOM is generated.

```

28.3 Braille generators

The following classes are defined in [src/formatsgeneration/brailleGeneration/brailleGeneration.h/.cpp](#)

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formatsgeneration/brailleGeneration > grep
2     class brailleGeneration.h
3 enum class bsrUTFKind {
4 enum class bsrByteOrderingKind {
5 class EXP bsrBrailleGenerator : public smartable
6 /* this class is purely virtual
7 class EXP bsrAsciiBrailleGenerator : public bsrBrailleGenerator
8 class EXP bsrUTF8BrailleGenerator : public bsrBrailleGenerator
9 class EXP bsrUTF8DebugBrailleGenerator : public bsrUTF8BrailleGenerator
10 class EXP bsrUTF16BigEndianBrailleGenerator : public bsrBrailleGenerator
11 class EXP bsrUTF16SmallEndianBrailleGenerator : public bsrBrailleGenerator

```

The base class bsrBrailleGenerator contains:

```

1 public:
2
3     // public services
4     // -----
5
6     virtual void          generateCodeForBrailleCell (
7                             bsrCellKind cellKind) = 0;
8
9     void                generateCodeForCellsList (
10                            const S_bsrCellsList& cellsList);
11
12     virtual void          generateCodeForMusicHeading (
13                             const S_bsrMusicHeading& musicHeading);
14
15     virtual void          generateCodeForLineContents (
16                             const S_bsrLineContents& lineContents);
17
18     // ... ..
19
20 protected:
21
22     // protected fields
23     // -----
24
25     std::ostream&          fBrailleOutputStream;

```

28.4 Writing braille cells

Braille cells are output to an `std::ostream` as hexadecimal strings by virtual method `generateCodeForBrailleCell` methods in `src/passes/bsr2braille/brailleGeneration.h`, depending on the kind of output selected.

For example, ASCII braille generation is done by:

```

1 void bsrAsciiBrailleGenerator::generateCodeForBrailleCell (
2     bsrCellKind cellKind)
3 {
4     std::string stringForCell;
5
6     switch (cellKind) {
7     case bsrCellKind::kCellUnknown:
8         {
9             std::stringstream ss;
10
11             ss <<
12                 "cannot generate code for braille cell '" <<
13                 bsrCellKindAsString (cellKind) <<
14                 "'";
15             msrInternalError (
16                 gServiceRunData->getInputSourceName (),
17                 -98, // inputLineNumber, TICINO JMI
18                 __FILE__, __LINE__,
19                 ss.str ());
20         }
21         break;
22
23     case bsrCellKind::kCellEOL:      stringForCell = "\x0a"; break;
24     case bsrCellKind::kCellEOP:      stringForCell = "\x0c"; break;
25
26     case bsrCellKind::kDotsNone:      stringForCell = "\x20"; break;
27
28     case bsrCellKind::kDots1:         stringForCell = "\x41"; break;
29     case bsrCellKind::kDots2:         stringForCell = "\x31"; break;
30

```

```
31     // ... ..
32
33     case bsrCellKind::kDots23456:  stringForCell = "\x29"; break;
34     case bsrCellKind::kDots123456: stringForCell = "\x3d"; break;
35 } // switch
36
37 fBrailleOutputStream <<
38     stringForCell;
39 }
```

Chapter 29

MusicXML generation

MusicXML text is produced on the standard output stream, unless options `'-output-file-name'` or `'-auto-output-f'` are used.

29.1 Basic principle

MusicXML generation is done in two passes:

- first create and MXSR containing the data;
- then simply write this tree.

29.2 Creating an xmlelement

An simple example is:

```
1 // create a direction element
2 Sxmlelement directionElement = createMxmlelement (k_direction, "");
3
4 // set it's "placement" attribute if relevant
5 std::string
6     placementString =
7     msrPlacementKindAsMusicXMLString (placementKind);
8
9 if (placementString.size ()) {
10     directionElement->add (createMxmlAttribute ("placement", placementString));
11 }
```

This one supplies a value to the `xmlelement` it creates:

```
1 void msr2mxsrTranslator::visitStart (S_msrIdentification& elt)
2 {
3     // composers
4     const std::list<std::string>&
5     composersList =
6     elt->getComposersList ();
7
8     for (
9         std::list<std::string>::const_iterator i = composersList.begin ();
10         i != composersList.end ();
11         ++i
12     ) {
13         std::string variableValue = (*i);
```

```

14
15 // create a creator element
16 Sxmlelement creatorElement = createMxmlelement (k_creator, variableValue);
17
18 // set its "type" attribute
19 creatorElement->add (createMxmlAttribute ("type", "composer"));
20
21 // append it to the composers elements list
22 fComposersElementsList.push_back (creatorElement);
23 } // for
24
25 // ... ..
26 }

```

29.3 Creating an xmlelement tree

In , this code:

```

1 void msr2mxsrTranslator::visitStart (S_msrClef& elt)
2 {
3     // ... ..
4
5     Sxmlelement clefElement = createMxmlelement (k_clef, "");
6
7     // set clefElement's "number" attribute if relevant
8     /*
9      0 by default in MSR,
10     1 by default in MusicXML:
11     The optional number attribute refers to staff numbers within the part,
12     from top to bottom on the system.
13     A value of 1 is assumed if not present.
14     */
15
16     int clefStaffNumber =
17         elt->getClefStaffNumber ();
18
19     if (clefStaffNumber > 1) {
20         clefElement->add (
21             createMxmlIntegerAttribute ("number", clefStaffNumber));
22     }
23
24     // populate clefElement
25     switch (elt->getClefKind ()) {
26         // ... ..
27
28         case msrClefKind::kClefTrebleMinus8:
29             {
30                 clefElement->push (
31                     createMxmlelement (
32                         k_sign,
33                         "G"));
34                 clefElement->push (
35                     createMxmlIntegerElement (
36                         k_line,
37                         2));
38                 clefElement->push (
39                     createMxmlIntegerElement (
40                         k_clef_octave_change,
41                         -1));
42             }
43             break;
44
45         // ... ..
46     }

```

creates this MusicXML element depending on the value returned by method `msrClef::getClefStaffNumber ()`:

```

1 <clef number="2">
2   <sign>G</sign>
3   <line>2</line>
4   <clef-octave-change>-1</clef-octave-change>
5 </clef>

```

29.4 Browsing the visited MSR score

The creation of the tree is done in `src/passes/msr2mxsr/msr2mxsrTranslator.h/.cpp`.

Class `msr2mxsrTranslator` is defined in those files, it contains:

```

1 public:
2
3         msr2mxsrTranslator (
4             const S_msrScore& visitedMsrScore);
5
6     virtual          ~msr2mxsrTranslator ();
7
8     Sxmlelement      translateMsrToMxsr ();
9
10    // ... ..
11
12 private:
13
14     // the MSR score we're visiting
15     // -----
16     S_msrScore          fVisitedMsrScore;
17
18
19     // the MXSR we're building
20     // -----
21     Sxmlelement          fResultingMusicxmlelement;

```

The method `msr2mxsrTranslator::translateMsrToMxsr ()` method does the following:

```

1 // -----
2 Sxmlelement msr2mxsrTranslator::translateMsrToMxsr ()
3 {
4     #ifdef MF_SANITY_CHECKS_ARE_ENABLED
5         // sanity check
6         mfAssert (
7             __FILE__, __LINE__,
8             fVisitedMsrScore != nullptr,
9             "fVisitedMsrScore is null");
10    #endif // MF_SANITY_CHECKS_ARE_ENABLED
11
12    // create the current score part-wise element
13    fResultingMusicxmlelement =
14        createMxmlScorePartWiseElement ();
15
16    // create a msrScore browser
17    msrBrowser<msrScore> browser (this);
18
19    // set the parts browsing order
20    theMsrScore->
21        setStavesBrowsingOrderKind (
22            msrStavesBrowsingOrderKind::kStavesBrowsingOrderHarmoniesFiguredBassesRegulars);
23
24    // browse the visited MSR score
25    browser.browse (*fVisitedMsrScore);

```



```
26 |  
27 |     return fResultingMusicxmlelement;  
28 | }
```

29.5 Ancillary functions to create MXSR data

The function `createMxmlScorePartWiseElement ()` is defined in `src/formats/mxsr/mxsr.h/.cpp`:

```
1 | // -----  
2 | Sxmlelement createMxmlScorePartWiseElement ()  
3 | {  
4 |     Sxmlelement result = factory::instance ().create (k_score_partwise);  
5 |  
6 |     Sxmllattribute versionAttribute = createMxmlAttribute("version", "3.1");  
7 |     result->add (versionAttribute);  
8 |  
9 |     return result;  
10 | }
```

Chapter 30

Guido code generation

Guido code is produced on standard output, unless options option `-output-file-name`, `-obj` or option `-auto-output-file-name`, `-aofn` are used.

30.1 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 [29.1](#) [musicxmlGeneration], page [245](#), for more details.

Part IX

Generators

Chapter 31

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.

31.1 MusicAndHarmonies

`MusicAndHarmonies.cpp`

31.2 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 MSDDL converter.

The score produced is shown at figure 31.1 [Zoltán Kodály's Mikrokosmos III Wandering], page 251.



Figure 31.1: Zoltán Kodály's Mikrokosmos III Wandering

31.3 LilyPondIssue34


This service produces the same score as that obtained by:

```
1 xml2ly -auto-output-file-name gracenotes/LilyPondIssue34.xml
```

The resulting score is shown at figure 31.2 [The LilyPondIssue34 score], page 252.

Piano Sonata in A Major

Wolfgang A



Piano

Figure 31.2: The LilyPondIssue34 score

The name LilyPondIssue34 stems from the fact that translating this MusicXML file to LilyPond with `musicxml2ly` exhibits the famous LilyPond issue #34.

This example was written to design a LilyPond-oriented interface to LPSR, preparing the grounds for LilyPond export to other formats. This work is in progress at the time of this writing.

Part X

Converters

Chapter 32

The converters

A multi-pass converter performs a sequence of passes, i.e. a sequence of steps. For example, `xml2ly` performs the following passes:

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

32.1 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.

32.2 xml2brl

`xml2brl` is mentioned here, but not described in detail.

32.3 xml2xml**32.4** xml2gmh**32.5** msdlconverter

Part XI

Interfaces

Chapter 33

Library interfaces

Chapter 34

Representations interfaces

These interfaces are a set of functions to create formats for various needs.

34.1 MSR interfaces

The MSR interfaces are in `interfaces/msrinterfaces/`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll interfaces/msrinterfaces/  
2 total 32  
3 0 drwxr-xr-x  5 jacquesmenu  staff    160 May 26 08:20:55 2021 ./  
4 0 drwxr-xr-x  8 jacquesmenu  staff    256 Jun 25 05:59:13 2021 ../  
5 8 -rw-r--r--@ 1 jacquesmenu  staff     77 Apr 22 15:49:27 2021 README.md  
6 16 -rw-r--r--@ 1 jacquesmenu  staff   5796 Jun 24 17:47:02 2021 msrInterface.cpp  
7 8 -rw-r--r--@ 1 jacquesmenu  staff   1371 Jun 13 07:38:04 2021 msrInterface.h
```

34.2 LPSR interfaces

The LSPR interfaces are in `interfaces/lpsrinterfaces/`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll interfaces/lpsrinterfaces/  
2 total 24  
3 0 drwxr-xr-x  5 jacquesmenu  staff    160 Jun 13 07:36:53 2021 ./  
4 0 drwxr-xr-x  8 jacquesmenu  staff    256 Jun 25 05:59:13 2021 ../  
5 8 -rw-r--r--@ 1 jacquesmenu  staff     78 Jun 13 07:37:13 2021 README.md  
6 8 -rw-r--r--@ 1 jacquesmenu  staff    670 Jun 13 07:41:01 2021 lpsrInterface.cpp  
7 8 -rw-r--r--@ 1 jacquesmenu  staff   1450 Jun 13 07:39:29 2021 lpsrInterface.h
```

34.3 MSDL interfaces

The MSDL interfaces are in `interfaces/msdlinterfaces/`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > ll interfaces/msdlinterfaces/  
2 total 8  
3 0 drwxr-xr-x  3 jacquesmenu  staff     96 Jun 25 05:57:39 2021 ./  
4 0 drwxr-xr-x  8 jacquesmenu  staff    256 Jun 25 05:59:13 2021 ../  
5 8 -rw-r--r--@ 1 jacquesmenu  staff   1967 Jun  6 06:38:55 2021 libmsdl.h
```

Chapter 35

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 services, to be used in terminals and scripts.

The CLI services actually use the API functions to do their job. For example, in `main ()` function does:

```
1 int main (int argc, char* argv[])
2 // -----
3 {
4     // setup signals catching
5     // -----
6
7     catchSignals ();
8
9     // ... ..
10
11    switch (multiGenerationOutputKind) {
12        case mfMultiGenerationOutputKind::kGeneration_UNKNOWN_:
13            // should not occur, unless the run is a pure help one
14            break;
15
16            // ... ..
17
18        case mfMultiGenerationOutputKind::kGeneration_Guido:
19            err =
20                msrScore2guidoWithHandler (
21                    theMsrScore,
22                    mfPassIDKind::kMfPassID_2,
23                    "Convert the MSR score into a second MSR",
24                    mfPassIDKind::kMfPassID_3,
25                    "Convert the second MSR into an MXSR",
26                    mfPassIDKind::kMfPassID_4,
27                    "Convert the MXSR into Guido text",
28                    std::cout,
29                    std::cerr,
30                    handler);
31            break;
32
33            // ... ..
34    }
```

35.1 Translating MusicXML data to an MXSR

35.2 Translating an MXSR to an MSR

35.3 Translating an MSR to an MXSR

35.4 Translating an MSR to another MSR

Such translation is meant to offer an opportunity to modify the score's description depending on options.

35.5 Translating an MSR to an LPSR

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

35.6 Translating an LPSR to LilyPond code

35.7 Translating an MSR to an BSR

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

35.8 Translating a BSR to another BSR

35.9 Translating an MXSR to Guido

Chapter 36

Converters interfaces

These interfaces are a set of functions to run the various converters. They are placed in the corresponding subdirectories of `src/converters/`, such as `src/converters/musicxml2musicxml/musicxml2musicxmlInterface.h`.

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/converters/musicxml2musicxml > cat
   musicxml2musicxmlInterface.h
2 /*
3  MusicFormats Library
4  Copyright (C) Jacques Menu 2016-2023
5
6  This Source Code Form is subject to the terms of the Mozilla Public
7  License, v. 2.0. If a copy of the MPL was not distributed with this
8  file, you can obtain one at http://mozilla.org/MPL/2.0/.
9
10 https://github.com/jacques-menu/musicformats
11 */
12
13 #ifndef __musicxml2musicxmlInterface__
14 #define __musicxml2musicxmlInterface__
15
16 #include "mfMusicformatsErrors.h" // for mfMusicformatsErrorKind
17
18 namespace MusicFormats
19 {
20 /*
21  The API functions with an options and arguments and no handler
22  are declared in libmusicxml.h
23  */
24
25 // -----
26 EXP mfMusicformatsErrorKind convertMusicxmlFile2musicxmlWithHandler (
27     const char*   fileName,
28     std::ostream& out,
29     std::ostream& err,
30     const S_oahHandler& handler);
31
32 // -----
33 EXP mfMusicformatsErrorKind convertMusicxmlFd2musicxmlWithHandler (
34     FILE*         fd,
35     std::ostream& out,
36     std::ostream& err,
37     const S_oahHandler& handler);
38
39 // -----
40 EXP mfMusicformatsErrorKind convertMusicxmlString2musicxmlWithHandler (
41     const char*   buffer,
42     std::ostream& out,
43     std::ostream& err,
```

```
45     const S_oahHandler& handler);  
46  
47  
48 }  
49  
50  
51 #endif // __musicxml2musicxmlInterface__
```

Part XII

Releases

36.1 The GitHub repository

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

The releases Zip archives are supplied with all MusicFormats versions, i.e. the current, most recent version of MusicFormats (the default master branch in the repository), and the earlier versions such as the v0.9.65 release.

The `musicformats-ubuntu-distrib` archive has to be clicked to get downloaded, since its URL cannot be guessed by an algorithm (it contains numbers internal to GitHub).

Doing so for the three releases, we get the following, here in the `Downloads/` folder on Mac OS™, with the Zip archives are automatically uncompressed :

```
1 jacquesmenu@macmini: ~/Downloads > ls -sal musicformats-*-distrib
2 musicformats-macos-distrib:
3 total 8
4 0 drwx-----@ 5 jacquesmenu staff 160 Mar 3 09:18 .
5 0 drwx-----+ 72 jacquesmenu staff 2304 Mar 3 09:18 ..
6 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:10 MusicFormatsVersionNumber.txt
7 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Mar 3 09:18 build
8 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 documentation
9
10 musicformats-ubuntu-distrib:
11 total 8
12 0 drwx-----@ 5 jacquesmenu staff 160 Mar 3 09:18 .
13 0 drwx-----+ 72 jacquesmenu staff 2304 Mar 3 09:18 ..
14 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:31 MusicFormatsVersionNumber.txt
15 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 build
16 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 documentation
17
18 musicformats-windows-distrib:
19 total 8
20 0 drwx-----@ 5 jacquesmenu staff 160 Mar 3 09:18 .
21 0 drwx-----+ 72 jacquesmenu staff 2304 Mar 3 09:18 ..
22 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:43 MusicFormatsVersionNumber.txt
23 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 build
24 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 documentation
```

```
1 jacquesmenu@macmini: ~/Downloads > ls -sal musicformats-ubuntu-distrib/*
2 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:31 musicformats-ubuntu-distrib/
   MusicFormatsVersionNumber.txt
3
4 musicformats-ubuntu-distrib/build:
5 total 0
6 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 .
7 0 drwx-----@ 5 jacquesmenu staff 160 Mar 3 09:18 ..
8 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Mar 3 09:18 bin
9 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 lib
10
11 musicformats-ubuntu-distrib/documentation:
12 total 0
13 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 .
14 0 drwx-----@ 5 jacquesmenu staff 160 Mar 3 09:18 ..
15 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Mar 3 09:18 IntroductionToMusicXML
16 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Mar 3 09:18 mfuserguide
```

```
1 jacquesmenu@macmini: ~/Downloads > ls -sal musicformats-ubuntu-distrib/**
2 musicformats-ubuntu-distrib/build/bin:
3 total 2272
4 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Mar 3 09:18 .
5 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 ..
```

```

6 96 -rw-r--r--@ 1 jacquesmenu staff 49008 Mar 3 07:31 LilyPondIssue34
7 96 -rw-r--r--@ 1 jacquesmenu staff 49048 Mar 3 07:31 Mikrokosmos3Wandering
8 96 -rw-r--r--@ 1 jacquesmenu staff 47280 Mar 3 07:31 MusicAndHarmonies
9 96 -rw-r--r--@ 1 jacquesmenu staff 47272 Mar 3 07:31 RandomChords
10 96 -rw-r--r--@ 1 jacquesmenu staff 47272 Mar 3 07:31 RandomMusic
11 72 -rw-r--r--@ 1 jacquesmenu staff 33848 Mar 3 07:31 countnotes
12 40 -rw-r--r--@ 1 jacquesmenu staff 17648 Mar 3 07:31 displayMusicformatsHistory
13 40 -rw-r--r--@ 1 jacquesmenu staff 17648 Mar 3 07:31 displayMusicformatsVersion
14 104 -rw-r--r--@ 1 jacquesmenu staff 50400 Mar 3 07:31 msdlconverter
15 544 -rw-r--r--@ 1 jacquesmenu staff 276024 Mar 3 07:31 partsummary
16 88 -rw-r--r--@ 1 jacquesmenu staff 43768 Mar 3 07:31 readunrolled
17 80 -rw-r--r--@ 1 jacquesmenu staff 39064 Mar 3 07:31 xml2brl
18 80 -rw-r--r--@ 1 jacquesmenu staff 39104 Mar 3 07:31 xml2gmn
19 48 -rw-r--r--@ 1 jacquesmenu staff 23192 Mar 3 07:31 xml2guido
20 72 -rw-r--r--@ 1 jacquesmenu staff 34816 Mar 3 07:31 xml2ly
21 88 -rw-r--r--@ 1 jacquesmenu staff 42928 Mar 3 07:31 xml2midi
22 80 -rw-r--r--@ 1 jacquesmenu staff 39104 Mar 3 07:31 xml2xml
23 88 -rw-r--r--@ 1 jacquesmenu staff 43416 Mar 3 07:31 xmlclone
24 48 -rw-r--r--@ 1 jacquesmenu staff 22616 Mar 3 07:31 xmlfactory
25 160 -rw-r--r--@ 1 jacquesmenu staff 79440 Mar 3 07:31 xmliter
26 56 -rw-r--r--@ 1 jacquesmenu staff 28472 Mar 3 07:31 xmlread
27 64 -rw-r--r--@ 1 jacquesmenu staff 28704 Mar 3 07:31 xmltranspose
28 40 -rw-r--r--@ 1 jacquesmenu staff 17360 Mar 3 07:31 xmlversion
29
30 musicformats-ubuntu-distrib/build/lib:
31 total 158600
32 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 .
33 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 ..
34 113728 -rw-r--r--@ 1 jacquesmenu staff 58227464 Mar 3 07:31 libmusicformats.a
35 44872 -rw-r--r--@ 1 jacquesmenu staff 22971160 Mar 3 07:31 libmusicformats.so
36
37 musicformats-ubuntu-distrib/documentation/IntroductionToMusicXML:
38 total 1704
39 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Mar 3 09:18 .
40 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 ..
41 1704 -rw-r--r--@ 1 jacquesmenu staff 869211 Mar 3 07:31 IntroductionToMusicXML.pdf
42
43 musicformats-ubuntu-distrib/documentation/mfuserguide:
44 total 3000
45 0 drwxr-xr-x@ 3 jacquesmenu staff 96 Mar 3 09:18 .
46 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 ..
47 3000 -rw-r--r--@ 1 jacquesmenu staff 1532300 Mar 3 07:31 mfuserguide.pdf

```

The contents of musicformats-windows-distrib/ differs in the lib/ contents:

```

1 jacquesmenu@macmini: ~/Downloads > ls -sal musicformats-windows-distrib/build/lib/
2 total 37672
3 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 .
4 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 ..
5 14768 -rw-r--r--@ 1 jacquesmenu staff 7558913 Mar 3 07:44 musicformats.exp
6 22904 -rw-r--r--@ 1 jacquesmenu staff 11726392 Mar 3 07:44 musicformats.lib

```

For Mac OS™, there is no lib/ directory, since the executables in bin/ are self-sufficient. They can be placed anywhere on a disk except the trash. Usually, they are placed in the /Applications/ directory.

36.1.1 Creating the releases

The hierarchy in the musicformats-*-distrib/ directories comes from the MusicFormats repository untouched, which is not convenient for the users.

Their contents is thus re-structured by `devtools/MakeMusicFormatsDistributions.bash`:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > devtools/MakeMusicFormatsDistributions.bash
2 ... ..
3
4 ==> final distrib contents:
5 -----
6
7 4208 -rw-r--r-- 1 jacquesmenu staff 2153547 Mar 3 12:55 /Users/jacquesmenu/
8 musicformats-git-dev/distrib/MusicFormatsForWindows.zip
9 34576 -rw-r--r-- 1 jacquesmenu staff 17559638 Mar 3 12:55 /Users/jacquesmenu/
10 musicformats-git-dev/distrib/MusicFormatsForUbuntu.zip
11 109512 -rw-r--r-- 1 jacquesmenu staff 55888914 Mar 3 12:55 /Users/jacquesmenu/
12 musicformats-git-dev/distrib/MusicFormatsForMacOS.zip
13 1704 -rw-r--r--@ 1 jacquesmenu staff 869211 Mar 3 07:10 /Users/jacquesmenu/
14 musicformats-git-dev/distrib/IntroductionToMusicXML.pdf
15 3000 -rw-r--r--@ 1 jacquesmenu staff 1532300 Mar 3 07:10 /Users/jacquesmenu/
16 musicformats-git-dev/distrib/mfuserguide.pdf
17 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:10 /Users/jacquesmenu/
18 musicformats-git-dev/distrib/MusicFormatsVersionNumber.txt
19
20 /Users/jacquesmenu/musicformats-git-dev/distrib/MusicFormatsForWindows:
21 total 8
22 0 drwxr-xr-x 16 jacquesmenu staff 512 Mar 3 12:55 ..
23 0 drwxr-xr-x 5 jacquesmenu staff 160 Mar 3 12:55 .
24 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 lib
25 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Mar 3 09:18 bin
26 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:43 MusicFormatsVersionNumber.txt
27
28 /Users/jacquesmenu/musicformats-git-dev/distrib/MusicFormatsForUbuntu:
29 total 8
30 0 drwxr-xr-x 16 jacquesmenu staff 512 Mar 3 12:55 ..
31 0 drwxr-xr-x 5 jacquesmenu staff 160 Mar 3 12:55 .
32 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Mar 3 09:18 lib
33 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Mar 3 09:18 bin
34 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:31 MusicFormatsVersionNumber.txt
35
36 /Users/jacquesmenu/musicformats-git-dev/distrib/MusicFormatsForMacOS:
37 total 8
38 0 drwxr-xr-x 16 jacquesmenu staff 512 Mar 3 12:55 ..
39 0 drwxr-xr-x 4 jacquesmenu staff 128 Mar 3 12:55 .
40 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Mar 3 07:10 bin
41 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:10 MusicFormatsVersionNumber.txt

```

The contents of `distrib/` is now:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/distrib > ls -sal
2 total 154128
3 0 drwxr-xr-x 16 jacquesmenu staff 512 Mar 3 13:18 .
4 0 drwxr-xr-x 35 jacquesmenu staff 1120 Mar 3 07:13 ..
5 24 -rw-r--r--@ 1 jacquesmenu staff 8196 Feb 24 13:33 .DS_Store
6 1704 -rw-r--r--@ 1 jacquesmenu staff 869211 Mar 3 07:10 IntroductionToMusicXML.pdf
7 0 drwxr-xr-x 4 jacquesmenu staff 128 Mar 3 13:18 MusicFormatsForMacOS
8 109960 -rw-r--r-- 1 jacquesmenu staff 55888914 Mar 3 13:18 MusicFormatsForMacOS.zip
9 0 drwxr-xr-x 5 jacquesmenu staff 160 Mar 3 13:18 MusicFormatsForUbuntu
10 35216 -rw-r--r-- 1 jacquesmenu staff 17559638 Mar 3 13:18 MusicFormatsForUbuntu.zip
11 0 drwxr-xr-x 5 jacquesmenu staff 160 Mar 3 13:18 MusicFormatsForWindows
12 4208 -rw-r--r-- 1 jacquesmenu staff 2153547 Mar 3 13:18 MusicFormatsForWindows.zip
13 3000 -rw-r--r--@ 1 jacquesmenu staff 1532300 Mar 3 07:10 mfuserguide.pdf
14 8 -rw-r--r--@ 1 jacquesmenu staff 6 Mar 3 07:10 MusicFormatsVersionNumber.
15 txt
16 8 -rwxr-xr-x@ 1 jacquesmenu staff 95 Mar 3 12:54 doClean.bash
17 0 drwx-----@ 6 jacquesmenu staff 192 Mar 3 10:56 musicformats-macos-distrib
18 0 drwx-----@ 6 jacquesmenu staff 192 Mar 3 10:56 musicformats-ubuntu-
19 distrib
20 0 drwx-----@ 6 jacquesmenu staff 192 Mar 3 10:56 musicformats-windows-
21 distrib

```

36.1.2 Security issue in recent MacOS™ versions

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

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

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

The trouble is that these executables are in *quarantine* by default. To make them usable, they have to quit quarantine and explicitly be made executable.

This is done this way using `chmod` and `xattr` in `devtools/MakeMusicFormatsDistributions.bash`:

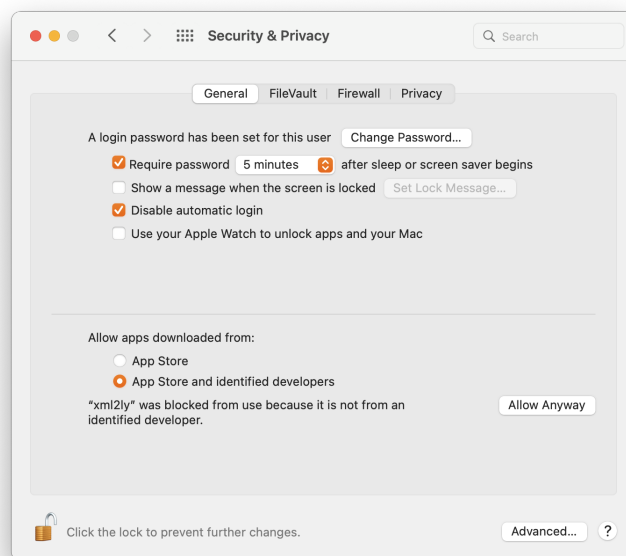
```
1 # make the executables actually executable
2 chmod +x bin/*
3 xattr -d com.apple.quarantine bin/*
```

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

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

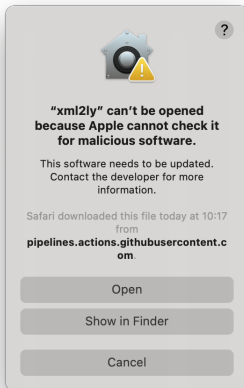
The above can be done in the GUI file by file too. Right after you got the message above:

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



- click on the *Allow Anyway* button.

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



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

36.2 GitHub actions

These actions are defined in .yaml files in .github/workflows/:

```
1 jacquesmenu@macmini:~/musicformats-git-dev/.github/workflows > ls -sal
2 total 24
3 0 drwxr-xr-x  5 jacquesmenu  staff   160 Aug 24 09:35 .
4 0 drwxr-xr-x  4 jacquesmenu  staff   128 Aug 22 08:41 ..
5 8 -rw-r--r--@ 1 jacquesmenu  staff  1366 Aug 23 07:09 build-macos-version.yml
6 8 -rw-r--r--@ 1 jacquesmenu  staff  1371 Aug 23 07:09 build-ubuntu-version.yml
7 8 -rw-r--r--@ 1 jacquesmenu  staff  1455 Aug 23 07:08 build-windows-version.yml
```

These actions are executed by default each time a `git push` is performed to the master branch. This can be disabled at will if needed.

For example, the Ubuntu action in file `build-ubuntu-version.yml` is shown below:

```
1 jacquesmenu@macstudio-1:/Volumes/JMI_Volume/JMI_Developpement/musicformats-git-dev/.github
  /workflows > cat build-ubuntu-artifact.yml
2 # This is a workflow to build MusicFormats and create an artifact of it
3
4 name: Build Ubuntu Version
5
6 # Controls when the action will run.
7 on:
8   # Triggers the workflow on push or pull request events but only for the master branch
9   push:
10     branches: [ master ]
11   pull_request:
12     branches: [ master ]
13
14 # Allows you to run this workflow manually from the Actions tab
15 workflow_dispatch:
16
17 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
18 jobs:
19   # This workflow contains a single job called "build"
20   build:
21     # The type of runner that the job will run on
22     runs-on: ubuntu-latest
23
24     # Steps represent a sequence of tasks that will be executed as part of the job
```

```

25  steps:
26      # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
27      - uses: actions/checkout@v3
28      - uses: actions/setup-node@v3
29        with:
30          node-version: '16'
31
32      - name: Build MusicFormats for Ubuntu
33        run: make -C build
34
35      - name: Upload libraries and executables for Ubuntu
36        uses: actions/upload-artifact@v3
37        with:
38          name: musicformats-ubuntu-artifact
39          path: |
40            MusicFormatsVersionNumber.txt
41            MusicFormatsVersionDate.txt
42            build/bin
43            build/lib
44            include
45            documentation/IntroductionToMusicXML/IntroductionToMusicXML.pdf
46            documentation/mfuserguide/mfuserguide.pdf
47            documentation/MusicFormatsFigures/MusicFormatsFigures.pdf
48            documentation/mfmaintainanceguide/mfmaintainanceguide.pdf
49            #uncomment to have the full libdir in the .zip archive
50            #build/libdir

```

After a push to the master branch:

```

1  jacquesmenu@macmini: ~/musicformats-git-dev > git commit -m "Switched from C++11 to C++17
   for <filesystem>" -a[master 77d3d29] Switched from C++11 to C++17 for <filesystem>
2  7 files changed, 10 insertions(+), 10 deletions(-)
3
4  jacquesmenu@macmini: ~/musicformats-git-dev > git push
5  Enumerating objects: 33, done.
6  Counting objects: 100% (33/33), done.
7  Delta compression using up to 8 threads
8  Compressing objects: 100% (16/16), done.
9  Writing objects: 100% (17/17), 1.47 KiB | 1.47 MiB/s, done.
10 Total 17 (delta 14), reused 0 (delta 0), pack-reused 0
11 remote: Resolving deltas: 100% (14/14), completed with 13 local objects.
12 To https://github.com/jacques-menu/musicformats.git
13  a880063..77d3d29  master -> master

```

we get for example:

github.com/jacques-menu/musicformats/actions

Bienvenue sur e-PJ MusicFormats Qt C++ MusicXML Brother Mac OS 12 LilyPond Mac OS 11 Mac OS X JavaScript Sax Soprano

Actions · jacques-menu/musicformats

Search or jump to...

Pull requests Issues Marketplace Explore

jacques-menu / musicformats Public

Pin Unwatch 2 Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Workflows

New workflow

All workflows

MacOS

Node.js Package

Ubuntu

Windows

All workflows

Showing runs from all workflows

Filter workflow runs

777 workflow runs

Event Status Branch Actor

Switched from C++11 to C++17 for <filesystem>

Ubuntu #8: Commit 38b584f pushed by jacques-menu

master

4 hours ago 37m 47s

Switched from C++11 to C++17 for <filesystem>

Windows #8: Commit 38b584f pushed by jacques-menu

master

4 hours ago 46m 25s

Switched from C++11 to C++17 for <filesystem>

MacOS #8: Commit 38b584f pushed by jacques-menu

master

4 hours ago 20m 22s

Then clicking on the link leads to:

270/416

github.com/jacques-menu/musicformats/actions/runs/1921985888

Bienvenue sur e-PJ

MusicFormats

Qt

C++

MusicXML

Brother

Mac OS 12

LilyPond

Mac OS 11

Mac OS X

JavaScript

Sax Soprano

Switched from C++11 to C++17 for <filesystem> - jacques-menu/musicformats@38b584f

Search or jump to...

Pull requests

Issues

Marketplace

Explore

Pin

Unwatch

2

Fork

0

Star

0

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

Switched from C++11 to C++17 for <filesystem> Ubuntu #8

Re-run all jobs

Summary

Jobs

build

Triggered via push 19 hours ago

jacques-menu pushed 38b584f master

Status

Total duration

Artifacts

Success

37m 47s

1

ubuntu-check.yml

on: push

build

37m 35s

Artifacts

Produced during runtime

Name	Size
musicformats-ubuntu-distrib	79.3 MB

271/416

Chapter 37

MusicFormats releases

The MusicFormats repository contains a single, master branch, that contains the current evolution of the code base, examples and documentation.

When a `git push` is performed, the `musicformats-**-artifact` archives are created, but they cannot be added to the MusicFormats repository by GitHub because they are too large.

In order to create a new release, one should:

1. (re-)create the up-to-date documentation with:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > devtools/CreateDocumentationPDFs.bash
```

2. add all new and/or modified files to the local repository. The `addAll` function is defined for this:

```
1 jacquesmenu@macstudio:/Volumes/JMI_Volume/JMI_Developpement/musicformats-git-dev >
  typ addAll
2 addAll is a shell function from /Volumes/JMI_VOLUME/JMI_ShellSettings/
  zsh_20_MusicFormats
3 --
4 addAll () {
5     git add -f ${MUSIC_FORMATS_DEV}/MusicFormatsVersionNumber.txt
6     git add -f ${MUSIC_FORMATS_DEV}/MusicFormatsVersionDate.txt
7     git add -f ${MUSIC_FORMATS_DEV}/src/MusicFormatsVersionNumber.h
8     git add -f ${MUSIC_FORMATS_DEV}/src/MusicFormatsVersionDate.h
9     addSrc
10    addBuild
11    addDevtools
12    addInclude
13    addDoc
14    addFXML
15    addFmfs1
16 }
```

3. commit to the local repository clone with the release version number:

```
1 git commit -m "v0.9.67-alpha" -a
```

4. push this to the MusicFormats repo with:

```
1 git push
```

5. the actions in the MusicFormats repository then perform a build on Linux, Windows™ and Mac OS™. Check that they were executed successfully at <https://github.com/jacques-menu/musicformats/actions>:

📄 jacques-menu / musicformats Public

🔔 Notifications

🍴 Fork 0

★ Star 0

<> Code Issues Pull requests **Actions** Projects Wiki Security Insights

Workflows

All workflows

Create Ubuntu Release

MacOS

Ubuntu

Windows

All workflows

Showing runs from all workflows

🔍 Filter workflow runs

873 workflow runs

Event ▾

Status ▾

Branch ▾

Actor ▾

✓ **Pre v0.9.63**

Windows #40: Commit b4b09f8 pushed by jacques-menu

master

📅 2 hours ago

⌚ 1h 0m 14s

✓ **Pre v0.9.63**

Ubuntu #40: Commit b4b09f8 pushed by jacques-menu

master

📅 2 hours ago

⌚ 34m 14s

✓ **Pre v0.9.63**

MacOS #40: Commit b4b09f8 pushed by jacques-menu

master

📅 2 hours ago

⌚ 16m 49s

6. when that is the case, download each of the three resulting `musicformats-*-distrib` archives locally in turn:

📄 jacques-menu / musicformats Public

🔔 Notifications

🍴 Fork 0

★ Star 0

<> Code Issues Pull requests **Actions** Projects Wiki Security Insights

✓ **Pre v0.9.63 Ubuntu #40**

Sign in to view logs

Summary

Jobs

✓ build

Triggered via push 2 hours ago

👤 jacques-menu pushed → b4b09f8 master

Status

Success

Total duration

34m 14s

Artifacts

1

ubuntu-check.yml

on: push

✓ build

34m 3s

🔍 - +

Artifacts

Produced during runtime

Name	Size
📦 musicformats-ubuntu-distrib	83.7 MB

On this authors's machine, they go to `${HOME}/Downloads`:

```

1 jacquesmenu@macstudio: ~/JMI_Downloads > ls -sal musicformats-*artifact
2 musicformats-macos-artifact:
3 total 32
4 0 drwx-----@ 8 jacquesmenu staff 256 Mar 13 14:26 .
5 0 drwxr-xr-x@ 65 jacquesmenu staff 2080 Mar 13 14:29 ..
6 16 -rw-r--r--@ 1 jacquesmenu staff 6148 Mar 13 14:27 .DS_Store
7 8 -rw-r--r--@ 1 jacquesmenu staff 15 Mar 13 12:49 MusicFormatsVersionDate.txt
8 8 -rw-r--r--@ 1 jacquesmenu staff 12 Mar 13 12:49 MusicFormatsVersionNumber.txt
9 0 drwxr-xr-x@ 5 jacquesmenu staff 160 Mar 13 14:26 build
10 0 drwxr-xr-x@ 7 jacquesmenu staff 224 Mar 13 14:26 documentation
11 0 drwxr-xr-x@ 18 jacquesmenu staff 576 Mar 13 14:26 include
12
13 musicformats-ubuntu-artifact:
14 total 32
15 0 drwx-----@ 8 jacquesmenu staff 256 Mar 13 14:26 .
16 0 drwxr-xr-x@ 65 jacquesmenu staff 2080 Mar 13 14:29 ..
17 16 -rw-r--r--@ 1 jacquesmenu staff 6148 Mar 13 14:27 .DS_Store
18 8 -rw-r--r--@ 1 jacquesmenu staff 15 Mar 13 12:41 MusicFormatsVersionDate.txt
19 8 -rw-r--r--@ 1 jacquesmenu staff 12 Mar 13 12:41 MusicFormatsVersionNumber.txt
20 0 drwxr-xr-x@ 5 jacquesmenu staff 160 Mar 13 14:26 build
21 0 drwxr-xr-x@ 7 jacquesmenu staff 224 Mar 13 14:26 documentation
22 0 drwxr-xr-x@ 18 jacquesmenu staff 576 Mar 13 14:26 include
23
24 musicformats-windows-artifact:
25 total 32
26 0 drwx-----@ 8 jacquesmenu staff 256 Mar 13 14:26 .
27 0 drwxr-xr-x@ 65 jacquesmenu staff 2080 Mar 13 14:29 ..
28 16 -rw-r--r--@ 1 jacquesmenu staff 6148 Mar 13 14:27 .DS_Store
29 8 -rw-r--r--@ 1 jacquesmenu staff 16 Mar 13 12:59 MusicFormatsVersionDate.txt
30 8 -rw-r--r--@ 1 jacquesmenu staff 12 Mar 13 12:59 MusicFormatsVersionNumber.txt
31 0 drwxr-xr-x@ 5 jacquesmenu staff 160 Mar 13 14:26 build
32 0 drwxr-xr-x@ 7 jacquesmenu staff 224 Mar 13 14:26 documentation
33 0 drwxr-xr-x@ 18 jacquesmenu staff 576 Mar 13 14:26 include

```

7. create the releases in the local MusicFormats repository clone:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > devtools/MakeMusicFormatsDistributions.
2 bash
3 ... ..
4 ==> final distrib contents:
5 -----
6
7 4368 -rw-r--r-- 1 jacquesmenu staff 2234811 Jun 9 11:47 /Users/jacquesmenu/
8 musicformats-git-dev/releases/MusicFormatsForWindows.zip
9 35312 -rw-r--r-- 1 jacquesmenu staff 18076799 Jun 9 11:47 /Users/jacquesmenu/
10 musicformats-git-dev/releases/MusicFormatsForUbuntu.zip
11 127824 -rw-r--r-- 1 jacquesmenu staff 65442854 Jun 9 11:47 /Users/jacquesmenu/
12 musicformats-git-dev/releases/MusicFormatsForMacOS.zip
13 1712 -rw-r--r--@ 1 jacquesmenu staff 872863 Jun 9 07:40 /Users/jacquesmenu/
14 musicformats-git-dev/releases/IntroductionToMusicXML.pdf
15 5328 -rw-r--r--@ 1 jacquesmenu staff 2724130 Jun 9 07:40 /Users/jacquesmenu/
16 musicformats-git-dev/releases/mfuserguide.pdf
17 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 07:40 /Users/jacquesmenu/
18 musicformats-git-dev/releases/MusicFormatsVersionNumber.txt
19
20 /Users/jacquesmenu/musicformats-git-dev/releases/MusicFormatsForWindows:
21 total 8
22 0 drwxr-xr-x 14 jacquesmenu staff 448 Jun 9 11:47 ..
23 0 drwxr-xr-x 5 jacquesmenu staff 160 Jun 9 11:47 .
24 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Jun 9 11:43 lib
25 0 drwxr-xr-x@ 26 jacquesmenu staff 832 Jun 9 11:43 bin
26 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 08:14 MusicFormatsVersionNumber.txt
27
28 /Users/jacquesmenu/musicformats-git-dev/releases/MusicFormatsForUbuntu:

```

```

23 total 8
24 0 drwxr-xr-x 14 jacquesmenu staff 448 Jun 9 11:47 ..
25 0 drwxr-xr-x 5 jacquesmenu staff 160 Jun 9 11:47 .
26 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Jun 9 11:44 lib
27 0 drwxr-xr-x@ 26 jacquesmenu staff 832 Jun 9 11:44 bin
28 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 07:57 MusicFormatsVersionNumber.txt
29
30 /Users/jacquesmenu/musicformats-git-dev/releases/MusicFormatsForMacOS:
31 total 8
32 0 drwxr-xr-x 14 jacquesmenu staff 448 Jun 9 11:47 ..
33 0 drwxr-xr-x 4 jacquesmenu staff 128 Jun 9 11:47 .
34 0 drwxr-xr-x@ 26 jacquesmenu staff 832 Jun 9 07:40 bin
35 8 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 07:40 MusicFormatsVersionNumber.txt

```

Now, the **releases** folder contains the MusicFormats new releases for the three operating systems:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/releases > ls -al
2 total 174552
3 drwxr-xr-x 14 jacquesmenu staff 448 Jun 9 11:47 .
4 drwxr-xr-x 29 jacquesmenu staff 928 Jun 9 09:13 ..
5 -rw-r--r--@ 1 jacquesmenu staff 872863 Jun 9 07:40 IntroductionToMusicXML.pdf
6 drwxr-xr-x 4 jacquesmenu staff 128 Jun 9 11:47 MusicFormatsForMacOS
7 -rw-r--r-- 1 jacquesmenu staff 65442854 Jun 9 11:47 MusicFormatsForMacOS.zip
8 drwxr-xr-x 5 jacquesmenu staff 160 Jun 9 11:47 MusicFormatsForUbuntu
9 -rw-r--r-- 1 jacquesmenu staff 18076799 Jun 9 11:47 MusicFormatsForUbuntu.zip
10 drwxr-xr-x 5 jacquesmenu staff 160 Jun 9 11:47 MusicFormatsForWindows
11 -rw-r--r-- 1 jacquesmenu staff 2234811 Jun 9 11:47 MusicFormatsForWindows.zip
12 -rw-r--r--@ 1 jacquesmenu staff 2724130 Jun 9 07:40 mfuserguide.pdf
13 -rw-r--r--@ 1 jacquesmenu staff 6 Jun 9 07:40 MusicFormatsVersionNumber.
    txt
14 drwx-----@ 5 jacquesmenu staff 160 Jun 9 11:44 musicformats-macos-distrib
15 drwx-----@ 5 jacquesmenu staff 160 Jun 9 11:44 musicformats-ubuntu-distrib
16 drwx-----@ 5 jacquesmenu staff 160 Jun 9 11:43 musicformats-windows-distrib

```

8. create the new release repository on the remote GitHub repository:
 - (a) click on the Releases link in the repository root window's right pane;
 - (b) click on the 'Draft a new release' button;
 - (c) click on the 'Choose a tag' button to create a new tag such as 'v0.9.67-alpha';
 - (d) describe this release as needed;
 - (e) drop the three Zip files in the **releases** folder to attach them to this release;
 - (f) click on the 'Publish release' button to copy the Zip file to GitHub.

9. create a new version number and date, for example:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > devtools/SetMusicFormatsVersionNumber.
    bash 0.9.64
2 -bash: devtools/SetMusicFormatsVersionNumber.: No such file or directory
3 jacquesmenu@macmini: ~/musicformats-git-dev > devtools/SetMusicFormatsVersionNumber.
    bash 0.9.64
4 ==> PWD is:
5 /Users/jacquesmenu/musicformats-git-dev
6
7 ==> Writing MusicFormats version number 0.9.64 to MusicFormatsVersionNumber.txt
8
9 8 -rw-r--r-- 1 jacquesmenu staff 6 Jun 9 12:14:57 2022 MusicFormatsVersionNumber.
    txt
10 0.9.64
11 ==> PWD is:
12 /Users/jacquesmenu/musicformats-git-dev/src
13
14 ==> Writing MusicFormats version number 0.9.64 to MusicFormatsVersionNumber.h
15

```

```
16 8 -rw-r--r--  1 jacquesmenu  staff   45 Jun  9 12:14:57 2022 MusicFormatsVersionNumber
    .h
17 #define MUSICFORMATS_VERSION_NUMBER "0.9.64"
```

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > devtools/SetMusicFormatsVersionDate.
  bash "June 9, 2022"
2 ==> PWD is:
3 /Users/jacquesmenu/musicformats-git-dev
4
5 ==> Writing MusicFormats version date June 9, 2022 to MusicFormatsVersionDate.txt
6
7 8 -rw-r--r--  1 jacquesmenu  staff   13 Jun  9 12:15:52 2022 MusicFormatsVersionDate.
    txt
8 June 9, 2022
9
10 ==> PWD is:
11 /Users/jacquesmenu/musicformats-git-dev/src
12
13 ==> Writing MusicFormats version date June 9, 2022 to MusicFormatsVersionDate.h
14
15 8 -rw-r--r--  1 jacquesmenu  staff   49 Jun  9 12:15:52 2022 MusicFormatsVersionDate.h
16 #define MUSICFORMATS_VERSION_DATE "June 9, 2022"
```

Check the result with:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > devtools/ShowMusicFormatsVersion.bash
2 Version number:
3 0.9.64Version date:
4 June 9, 2022
```

Part XIII

Selected topics

Chapter 38

Initializations

Some initialization activities in MusicFormats use the OAH facility. OAH should thus be initialized first.

38.1 Options and help initializations

There is no initialization of the OAH architecture as such, but there are `create*OahGroup ()` functions to create the various OAH groups.

For example, global variable `gServiceRunData` is supplied by `src/mflibrary/mfServiceRunData.h/.cpp`:

```
1 EXP extern S_generalOahGroup gServiceRunData;
2
3 // -----
4 EXP S_generalOahGroup createGlobalGeneralOahGroup ();
```

```
1 S_generalOahGroup createGlobalGeneralOahGroup ()
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     if (gEarlyOptions.getTraceEarlyOptions ()) {
5         std::stringstream ss;
6
7         ss <<
8             "Creating global general OAH group";
9
10        gWaeHandler->waeTraceWithoutInputLocation (
11            __FILE__, __LINE__,
12            ss.str ());
13    }
14 #endif // MF_TRACE_IS_ENABLED
15
16    // protect library against multiple initializations
17    if (! gServiceRunData) {
18        // create the global general options group
19        gServiceRunData =
20            generalOahGroup::create ();
21        assert (gServiceRunData != 0);
22    }
23
24    // return the global OAH group
25    return gServiceRunData;
26 }
```

38.2 Representations initializations

There are `initialize*` () functions such as `initializeLPSR` () and `initializeBSR` (). They essentially build global data structures, such as the tables of supported languages and their correspondance with an internal enumeration type both ways.

For example, `initializeMSR` () is defined in `src/formats/msr/msr.h/.cpp`:

```
1 void EXP initializeMSR ();

1 static S_mfcVersions pMsrRepresentationComponent;
2
3 static void initializeMsrRepresentationComponent ()
4 {
5     // create the component
6     pMsrRepresentationComponent =
7         mfcVersions::create ("MSR");
8
9     // populate it
10    pMsrRepresentationComponent->
11        appendVersionDescrToComponent (
12            mfcVersionDescr::create (
13                mfcVersionDescr::create (
14                    mfcVersionNumber::createFromString ("0.9.50"),
15                    "October 6, 2021",
16                    std::list<std::string> {
17                        "Start of sequential versions numbering"
18                    }
19                )))
20 }
21
22 void initializeMSR ()
23 {
24     // protect library against multiple initializations
25     static Bool pPrivateThisMethodHasBeenRun (false);
26
27     if (! pPrivateThisMethodHasBeenRun) {
28         // initialize the history
29         initializeMsrRepresentationComponent ();
30
31         // initialize
32         initializeMsrBasicTypes ();
33
34         pPrivateThisMethodHasBeenRun = true;
35     }
36 }
```

38.2.1 MSR initialization

`src/formats/msr/msrBasicTypes.h/.cpp` defines function `initializeMSRBasicTypes` () for this initialization:

```
1 void initializeMsrBasicTypes ()
2 {
3     // protect library against multiple initializations
4     static Bool pPrivateThisMethodHasBeenRun (false);
5
6     if (! pPrivateThisMethodHasBeenRun) {
7 #ifdef MF_TRACE_IS_ENABLED
8         if (gEarlyOptions.getTraceEarlyOptions () && ! gEarlyOptions.getEarlyQuietOption ()) {
9             gLog <<
10                "Initializing MSR basic types handling" <<
```



```

11         std::endl;
12
13     gWaeHandler->waeTraceWithoutInputLocation (
14         __FILE__, __LINE__,
15         ss.str ());
16 }
17 #endif // MF_TRACE_IS_ENABLED
18
19 // languages handling
20 // -----
21
22 initializeQuarterTonesPitchesLanguageKinds ();
23
24 // clefs handling
25 // -----
26
27 initializeClefKinds ();
28
29 // harmonies handling
30 // -----
31
32 initializeHarmonyKinds ();
33
34 // harmony structures handling
35 // -----
36
37 initializeHarmonyStructuresMap ();
38
39 // MSR lengths handling
40 // -----
41
42 initializeMsrLengthUnitKindsMap ();
43
44 // MSR margins types handling
45 // -----
46
47 initializeMsrMarginTypeKindsMap ();
48
49 pPrivateThisMethodHasBeenRun = true;
50 }
51 }

```

38.2.2 LPSR initialization

38.2.3 BSR initialization

38.3 Passes initializations

38.4 Converters initializations

The converters create only the global OAH groups they need. Since the order of initializations is critical, initialization of the formats is done when the latter's insider handler is created.

This is how class `xml2lyInsiderHandler` initializes the MSR and LSPR formats in method `xml2lyInsiderHandler::createTheXml2lyOptionGroups ()` in `src/converters/musicxml2lilypond/musicxml2lilypondInsiderHandler.cpp`:

```

1 void xml2lyInsiderHandler::createTheXml2lyOptionGroups (
2     std::string serviceName)
3 {
4     // ... ..
5
6     // initialize options handling, phase 1
7     // -----
8
9     // create the OAH OAH group first
10    appendGroupToHandler (
11        createGlobalOahOahGroup (
12            serviceName));
13
14    // create the WAE OAH group
15    appendGroupToHandler (
16        createGlobalWaeOahGroup ());
17
18    #ifdef MF_TRACE_IS_ENABLED
19        // create the trace OAH group
20        appendGroupToHandler (
21            createGlobalTraceOahGroup (
22                this));
23    #endif // MF_TRACE_IS_ENABLED
24
25    // create the output file OAH group
26    appendGroupToHandler (
27        createGlobalOutputFileOahGroup ());
28
29    // initialize the library
30    // -----
31
32    initializeWAE ();
33
34    initializeMSR ();
35    initializeLPSR ();
36
37    // initialize options handling, phase 2
38    // -----
39
40    // create the MXSR OAH group
41    appendGroupToHandler (
42        createGlobalMxsrOahGroup ());
43
44    // create the mxsr2msr OAH group
45    appendGroupToHandler (
46        createGlobalMxsr2msrOahGroup (
47            this));
48
49    // create the MSR OAH group
50    appendGroupToHandler (
51        createGlobalMsrOahGroup ());
52
53    // create the msr2msr OAH group
54    appendGroupToHandler (
55        createGlobalMsr2msrOahGroup ());
56
57    // create the msr2lpsr OAH group
58    appendGroupToHandler (
59        createGlobalMsr2lpsrOahGroup ());
60
61    // create the LPSR OAH group
62    appendGroupToHandler (
63        createGlobalLpsrOahGroup ());
64
65    // create the LilyPond generation OAH group
66    appendGroupToHandler (

```

```
67     createGlobalLpsr2lilypondOahGroup ());
68
69     // create the extra OAH group
70     appendGroupToHandler (
71         createGlobalHarmoniesExtraOahGroup ());
72
73     // 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 39

The OAH atoms collection

These handy general-purpose OAH atoms are used in MusicFormats itself. They are defined in [src/oah/oahAtomsCollection.h](#).

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

39.1 OAH macro atoms

A OAH macro atom is a combination, a list of several options under a single name. The `oahMacroAtom` class is defined in [src/oah/oahAtomsCollection.h/.cpp](#):

```

1 class EXP oahMacroAtom : public oahAtom
2 {
3     /*
4      a list of atoms
5     */
6
7     // ... ..
8
9     public:
10
11     // public services
12     // -----
13
14     void                oahValueLessAtomAlias (S_oahAtom atom);
15
16     void                applyValueLessAtom (std::ostream& os) override;
17
18     private:
19
20     // private fields
21     // -----
22
23     std::list<S_oahAtom> fMacroValueLessAtomsList;
24 };

```

Populating field `oahMacroAtom::fMacroValueLessAtomsList` is straightforward:

```

1 void oahMacroAtom::oahValueLessAtomAlias (S_oahAtom atom)
2 {
3     #ifdef MF_SANITY_CHECKS_ARE_ENABLED
4         // sanity check
5         mfAssert (
6             __FILE__, __LINE__,
7             atom != nullptr,
8             "atom is null");
9     #endif // MF_SANITY_CHECKS_ARE_ENABLED
10
11     fMacroValueLessAtomsList.push_back (atom);
12 }

```

Applying the macro atom is done in method `oahMacroAtom::applyElement ()`:

```

1 void oahMacroAtom::applyValueLessAtom (std::ostream& os)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gEarlyOptions.getTraceEarlyOptions ()) {
5             std::stringstream ss;
6
7             ss <<
8                 "=> option '" << fetchNames () << "' is a oahMacroAtom";
9
10            gWaeHandler->waeTraceWithoutInputLocation (
11                __FILE__, __LINE__,
12                ss.str ());
13        }
14    #endif // MF_TRACE_IS_ENABLED
15
16    for (
17        std::list<S_oahAtom>::const_iterator i =
18            fMacroValueLessAtomsList.begin ();
19        i != fMacroValueLessAtomsList.end ();
20        ++i
21    ) {
22        S_oahAtom atom = (*i);
23    }

```

```

24     if (
25         // oahAtomStoringAValue?
26         S_oahAtomStoringAValue
27         atomWithVariable =
28             dynamic_cast<oahAtomStoringAValue*>(&(*atom))
29     ) {
30         //     atomWithVariable-> JMI ???
31         //         applyAtomWithValue (theString, os);
32     }
33     else {
34         // value-less atom
35         atom->
36             applyValueLessAtom (os);
37     }
38 } // for
39 }
40
41 fSelected = true;

```

39.2 A OAH macro atom example

xml2brl has the `-auto-utf8`, `-au8d` option:

```

1 jacquesmenu@macmini > xml2brl -query auto-utf8d
2 --- Help for atom "auto-utf8d" in subgroup "Files"
3     -auto-utf8d, -au8d
4         Combines -auto-output-file-name, -utf8d and -use-encoding-in-file-name

```

This macro options is defined in `src/formatsgeneration/brailleGeneration/brailleGenerationOah.cpp` the following way:

```

1 void brailleGenerationOahGroup::initializeMacroOptions ()
2 {
3     S_oahSubGroup
4     subGroup =
5         oahSubGroup::create (
6             "Macros",
7             "help-braille-generation-macros", "hbgm",
8             R"()",
9             oahElementVisibilityKind::kElementVisibilityWhole,
10            this);
11
12     appendSubGroupToGroup (subGroup);
13
14     // create the auto utfd8 macro
15
16     S_oahMacroAtom
17     autoUTFd8MacroAtom =
18         oahMacroAtom::create (
19             "auto-utf8d", "au8d",
20             "Combines -auto-output-file-name, -utf8d and -use-encoding-in-file-name");
21
22     subGroup->
23         appendAtomToSubGroup (
24             autoUTFd8MacroAtom);
25
26     // populate it
27     autoUTFd8MacroAtom->
28         oahValueLessAtomAlias (
29             gGlobalOutputFileOahGroup->getAutoOutputFileNameAtom ());
30
31     fBrailleOutputKindAtom->
32         applyAtomWithValue (

```

```

33     "utf8d",
34     gLog);
35 autoUTFd8MacroAtom->
36     oahValueLessAtomAlias (
37         fBrailleOutputKindAtom);
38
39 autoUTFd8MacroAtom->
40     oahValueLessAtomAlias (
41         fUseEncodingInFileNameAtom);
42 }

```

39.3 LilyPond octave entry

Pass `lpsr2lilypond` has three options to choose this, all controlling one and the same variable:

```

1 jacquesmenu@macmini > xml2ly -query absolute
2 --- Help for atom "absolute" in subgroup "Notes"
3     -abs, -absolute
4         Use absolute octave entry in the generated LilyPond code.

```

```

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

```

```

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

```

This is done in `src/formatsgeneration/lilypondGeneration/lpsr2lilypondOah.h/.cpp` using a single instance of class `msrOctaveEntryVariable`:

```

1 class EXP msrOctaveEntryVariable : public smartable
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        std::string          fVariableName;
11        msrOctaveEntryKind    fOctaveEntryKind;
12 };

```

The three classes:

- `lilypondAbsoluteOctaveEntryAtom`

- `lilypondRelativeOctaveEntryAtom`
- `lilypondFixedOctaveEntryAtom`

all contain an alias for an class `msrOctaveEntryVariable` variable:

```
1 // private fields
2 // -----
3
4 msrOctaveEntryVariable&
5     fOctaveEntryKindVariable;
```

The `fOctaveEntryVariable` field of class `lpsr2lilypondOahGroup` shared by all three options atoms is:

```
1 // notes
2 // -----
3
4 msrOctaveEntryVariable
5     fOctaveEntryVariable;
```


Chapter 40

Measures handling

Measures are presented at section [20.29](#) [Measures], page [205](#).

40.1 Voices contents

Class `msrVoice` contains a list of the first elements and a last segment:

```
1  std::list<S_msrVoiceElement>
2      fVoiceInitialElementsList;
3
4  // fVoiceLastSegment contains the music
5  // not yet stored in fVoiceInitialElementsList,
6  // it is thus logically the end of the latter,
7  // and is created implicitly for every voice.
8  // It is needed 'outside' of the 'std::list<S_msrElement>'
9  // because it is not a mere S_msrElement, but a S_msrSegment
10 S_msrSegment      fVoiceLastSegment;
```

40.2 Voice elements

The class `msrVoiceElement` sub-classes instances in `fVoiceInitialElementsList` can be of types:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src > grep 'public msrVoiceElement' formats/
   msr/*.h
2 formats/msr/msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
3 formats/msr/msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
4 formats/msr/msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
5 formats/msr/msrMultiMeasureRests.h:class EXP msrMultiMeasureRest : public msrVoiceElement
6 formats/msr/msrSegments.h:class EXP msrSegment : public msrVoiceElement
```

Class `msrSegment` contains a list of measures:

```
1  // the measures in the segment contain the mmusic
2  std::list<S_msrMeasure>
3      fSegmentElementsList;
```

class contains a list of measure elements:

```
1  // elements
2
3  std::list<S_msrMeasureElement>
4      fMeasureElementsList;
```

40.3 Measure elements

The class `msrMeasureElements` sub-classes instances in can be of types:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'public
  msrMeasureElement' *.h
2 msrBarChecks.h:class EXP msrBarCheck : public msrMeasureElement
3 msrBarLines.h:class EXP msrBarLine : public msrMeasureElement
4 msrBarNumberChecks.h:class EXP msrBarNumberCheck : public msrMeasureElement
5 msrBreaks.h:class EXP msrLineBreak : public msrMeasureElement
6 msrBreaks.h:class EXP msrPageBreak : public msrMeasureElement
7 msrClefs.h:class EXP msrClef : public msrMeasureElement
8 msrCodas.h:class EXP msrCoda : public msrMeasureElement
9 msrDoubleTremolos.h:class EXP msrDoubleTremolo : public msrMeasureElement
10 msrEyeGlasses.h:class EXP msrEyeGlasses : public msrMeasureElement
11 msrFiguredBasses.h:class EXP msrFiguredBass : public msrMeasureElement
12 msrHarmonies.h:class EXP msrHarmony : public msrMeasureElement
13 msrHiddenMeasureAndBarLines.h:class EXP msrHiddenMeasureAndBarLine : public
  msrMeasureElement
14 msrInstruments.h:class EXP msrScordatura : public msrMeasureElement
15 msrInstruments.h:class EXP msrAccordionRegistration : public msrMeasureElement
16 msrInstruments.h:class EXP msrHarpPedalsTuning : public msrMeasureElement
17 msrInstruments.h:class EXP msrPedal : public msrMeasureElement
18 msrInstruments.h:class EXP msrDamp : public msrMeasureElement
19 msrInstruments.h:class EXP msrDampAll : public msrMeasureElement
20 msrKeys.h:class EXP msrKey : public msrMeasureElement
21 msrLyrics.h:class EXP msrSyllable : public msrMeasureElement
22 msrMusicXMLSpecifics.h:class EXP msrPrintLayout : public msrMeasureElement
23 msrRehearsalMarks.h:class EXP msrRehearsalMark : public msrMeasureElement
24 msrSegnos.h:class EXP msrSegno : public msrMeasureElement
25 msrDalSegnos.h:class EXP msrDalSegno : public msrMeasureElement
26 msrStavesDetails.h:class EXP msrStaffDetails : public msrMeasureElement
27 msrTempos.h:class EXP msrTempo : public msrMeasureElement
28 msrTimeSignatures.h:class EXP msrTimeSignature : public msrMeasureElement
29 msrTranspositions.h:class EXP msrOctaveShift : public msrMeasureElement
30 msrTranspositions.h:class EXP msrTransposition : public msrMeasureElement
31 msrVoiceStaffChanges.h:class EXP msrVoiceStaffChange : public msrMeasureElement

```

40.4 Appending measure elements to a measure

Appending music elements to a measure is done by method `msrMeasure::appendElementToMeasure ()`, defined in `src/formats/msr/msrMeasures.h/.cpp`:

```

1 void msrMeasure::appendElementToMeasure (const S_msrMeasureElement& elem)
2 {
3     int inputLineNumber =
4         elem->getInputStartLineNumber ();
5
6     #ifdef MF_TRACE_IS_ENABLED
7         if (gTraceOahGroup->getTraceMeasures ()) {
8             std::stringstream ss;
9
10            ss <<
11            "Appending element " <<
12            elem->asShortString () <<
13            " to measure " <<
14            asShortString () <<
15            " in voice \"" <<
16            fetchMeasureUpLinkToVoice ()->
17            getVoiceName () <<
18            "\", currentMeasureCurrentAccumulatedWholeNotesDuration: \"" <<
19            fCurrentMeasureCurrentAccumulatedWholeNotesDuration.asString () <<
20            ", line " << inputLineNumber;

```

```

21
22     gWaeHandler->waeTraceWithoutInputLocation (
23         __FILE__, __LINE__,
24         ss.str ());
25 }
26 #endif // MF_TRACE_IS_ENABLED
27
28 // set elem's measure number
29 elem->
30     setMeasureNumber (
31         fBarLineUpLinkToMeasure->getMeasureNumber ());
32
33 // set elem's measure position
34 elem->
35     setMeasurePosition (
36         this,
37         fCurrentMeasureCurrentAccumulatedWholeNotesDuration,
38         "appendElementToMeasure()");
39
40 fMeasureElementsList.push_back (elem);
41
42 // take elem's sounding whole notes into account JMI ???
43 if (false) // JMI v0.9.67
44     incrementCurrentMeasureCurrentAccumulatedWholeNotesDuration (
45         inputLineNumber,
46         elem->
47             getSoundingWholeNotes ());
48 }

```

Here is how a harmony instance is appended to a measure:

```

1 void msrMeasure::appendHarmonyToMeasure (const S_msrHarmony& harmony)
2 {
3     int inputLineNumber =
4         harmony->getInputStartLineNumber ();
5
6 #ifdef MF_TRACE_IS_ENABLED
7     if (gTraceOahGroup->getTraceHarmonies ()) {
8         gLog <<
9             "Appending harmony " << harmony->asString () <<
10             " to measure " <<
11             this->asShortString () <<
12             " in segment '" <<
13             fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
14             "' in voice \"" <<
15             fMeasureUpLinkToSegment->
16                 getSegmentUpLinkToVoice ()->
17                     getVoiceName () <<
18             "\", currentMeasureCurrentAccumulatedWholeNotesDuration: \"" <<
19             fCurrentMeasureCurrentAccumulatedWholeNotesDuration.asString () <<
20             ", line " << inputLineNumber;
21
22         gWaeHandler->waeTraceWithoutInputLocation (
23             __FILE__, __LINE__,
24             ss.str ());
25     }
26 #endif // MF_TRACE_IS_ENABLED
27
28 // set harmony's measure number
29 harmony->
30     setMeasureNumber (
31         fBarLineUpLinkToMeasure->getMeasureNumber ());
32
33 // append the harmony to the measure elements list
34 // DON'T call 'appendElementToMeasure (harmony)':
35 // that would override harmony's measure position,

```

```

36 // which already has the correct value, thus:
37 fMeasureElementsList.push_back (harmony);
38
39 // get harmony sounding whole notes
40 mfRational
41     harmonySoundingWholeNotes =
42     harmony->
43         getSoundingWholeNotes ();
44
45 // account for harmony duration in measure whole notes
46 incrementCurrentMeasureCurrentAccumulatedWholeNotesDuration (
47     inputLineNumber,
48     harmonySoundingWholeNotes);
49
50 // this measure contains music
51 fMeasureContainsMusic = true;
52 }

```

The task is simpler when appending a harmony to a measure clone, because the clone's harmony's measure number comes from the clone's original:

```

1 void msrMeasure::appendHarmonyToMeasureClone (const S_msrHarmony& harmony)
2 {
3     int inputLineNumber =
4         harmony->getInputStartLineNumber ();
5
6 #ifdef MF_TRACE_IS_ENABLED
7     if (gTraceOahGroup->getTraceHarmonies ()) {
8         gLog <<
9             "Appending harmony " << harmony->asString () <<
10             " to measure clone " <<
11             this->asShortString () <<
12             " in segment clone '" <<
13             fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
14             "' in voice clone \"" <<
15             fMeasureUpLinkToSegment->
16                 getSegmentUpLinkToVoice ()->
17                     getVoiceName () <<
18             "\", currentMeasureCurrentAccumulatedWholeNotesDuration: \"" <<
19             fCurrentMeasureCurrentAccumulatedWholeNotesDuration.asString () <<
20             ", line " << inputLineNumber;
21
22         gWaeHandler->waeTraceWithoutInputLocation (
23             __FILE__, __LINE__,
24             ss.str ());
25     }
26 #endif // MF_TRACE_IS_ENABLED
27
28 ++gIndenter;
29
30 // append the harmony to the measure elements list
31 appendElementToMeasure (harmony);
32
33 // fetch harmony sounding whole notes
34 mfRational
35     harmonySoundingWholeNotes =
36     harmony->
37         getSoundingWholeNotes ();
38
39 // account for harmony duration in measure whole notes
40 incrementCurrentMeasureCurrentAccumulatedWholeNotesDuration (
41     inputLineNumber,
42     harmonySoundingWholeNotes);
43
44 // this measure contains music
45 fMeasureContainsMusic = true;

```

```

46
47  --gIndenter;
48 }

```

40.5 Appending measures to a segment

Measures are appended to a segment by method `msrSegment::appendMeasureToSegment ()` in `src/formats/msr/msrSegments.h/.cpp`:

```

1 void msrSegment::appendMeasureToSegment (const S_msrMeasure& measure)
2 {
3     int inputLineNumber =
4         measure->getInputStartLineNumber ();
5
6     std::string measureNumber =
7         measure->getMeasureNumber ();
8
9     unsigned int segmentElementsListSize =
10        fSegmentElementsList.size ();
11
12     std::string currentMeasureNumber =
13        segmentElementsListSize == 0
14        ? ""
15        : fSegmentElementsList.back ()->getMeasureNumber ();
16
17 #ifdef MF_TRACE_IS_ENABLED
18     if (gTraceOahGroup->getTraceMeasures ()) {
19         std::stringstream ss;
20
21         ss <<
22             "Appending measure '" << measureNumber <<
23             "' to segment " << asString ();
24
25         if (fSegmentElementsList.size () == 0)
26             gLog <<
27                 ", as first measure";
28         else
29             gLog <<
30                 ", after measure number '" << currentMeasureNumber << "'";
31
32         gLog <<
33             "' in voice \"" <<
34             fSegmentUpLinkToVoice->getVoiceName () <<
35             "\"\" <<
36             ", line " << measure->getInputStartLineNumber ();
37
38         gWaeHandler->waeTraceWithoutInputLocation (
39             __FILE__, __LINE__,
40             ss.str ());
41     }
42 #endif // MF_TRACE_IS_ENABLED
43
44     if (measureNumber == currentMeasureNumber) {
45         std::stringstream ss;
46
47         ss <<
48             "appending measure number '" << measureNumber <<
49             "' occurs twice in a row in segment " <<
50             asString () <<
51             " in voice \"" <<
52             fSegmentUpLinkToVoice->getVoiceName () <<
53             "\"\";
54

```

```

55 //      msrInternalWarning ( // JMI
56     msrInternalError (
57         gServiceRunData->getInputSourceName (),
58         inputLineNumber,
59         __FILE__, __LINE__,
60         ss.str ());
61 }
62
63 // is measure the first one this segment?
64 if (segmentElementsListSize == 0) {
65     measure->
66         setMeasureFirstInSegmentKind (
67             msrMeasureFirstInSegmentKind::kMeasureFirstInSegmentKindYes);
68 }
69 else {
70     measure->
71         setMeasureFirstInSegmentKind (
72             msrMeasureFirstInSegmentKind::kMeasureFirstInSegmentKindNo);
73 }
74
75 // is measure the first one it the voice?
76 // this is necessary for voice clones,
77 // which don't go down the part-staff-voice-segment hierarchy
78 if (! fSegmentUpLinkToVoice->getVoiceFirstMeasure ()) {
79     // yes, register it as such
80     fSegmentUpLinkToVoice->
81         setVoiceFirstMeasure (measure);
82
83     measure->
84         setMeasureFirstInVoice ();
85 }
86
87 // append measure to the segment
88 fSegmentElementsList.push_back (measure);
89 }

```

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 `msrMultiMeasureRest::appendMeasureCloneToMultiMeasureRests ()` 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::appendMultiMeasureRestToVoice ()`

40.6 Appending measures to a voice

Method `msrVoice::appendMeasureCloneToVoiceClone ()` does the job in `src/formats/msr/msrVoices.h/.cpp`.

```

1 S_msrMeasure msrVoice::createAMeasureAndAppendItToVoice (
2     int      inputLineNumber,
3     std::string measureNumber,
4     msrMeasureImplicitKind
5         measureImplicitKind)
6 {
7     fVoiceCurrentMeasureNumber = measureNumber;
8
9     #ifdef MF_TRACE_IS_ENABLED
10    if (gTraceOahGroup->getTraceMeasures ()) {
11        std::stringstream ss;
12
13        ss <<
14            "Creating measure '" <<
15            measureNumber <<
16            "' and appending it to voice \"" << getVoiceName () << "\" " <<
17            "', line " << inputLineNumber;
18
19        gWaeHandler->waeTraceWithoutInputLocation (
20            __FILE__, __LINE__,
21            ss.str ());
22    }
23    #endif // MF_TRACE_IS_ENABLED
24
25    fCallsCounter++;
26
27    if (
28        //      true
29        //      ||
30        false
31        &&
32        (
33            fCallsCounter == 2 && getVoiceName ()
34            ==
35            "Part_POne_HARMONIES_Staff_Voice_Eleven_HARMONIES"
36        )
37    ) { // POUSSE JMI
38        gLog <<
39            std::endl <<
40            "++++ createAMeasureAndAppendItToVoice() POUSSE, fCallsCounter: " << fCallsCounter
41            << " +++++" <<
42            std::endl;
43        this->print (gLog);
44        gLog <<
45            std::endl;
46    }
47
48    #ifdef MF_TRACE_IS_ENABLED
49    if (gTraceOahGroup->getTraceMeasuresDetails ()) {
50        displayVoice (
51            inputLineNumber,
52            "createAMeasureAndAppendItToVoice() 1");
53    }
54    #endif // MF_TRACE_IS_ENABLED
55
56    ++gIndenter;
57
58    // create the voice last segment if needed
59    if (! fVoiceLastSegment) {
60        createNewLastSegmentForVoice (
61            inputLineNumber,
62            "createAMeasureAndAppendItToVoice() 2");

```

```

62 }
63
64 // append a new measure with given number to voice last segment
65 S_msrMeasure
66     result =
67         fVoiceLastSegment->
68             createAMeasureAndAppendItToSegment (
69                 inputLineNumber,
70                 measureNumber,
71                 measureImplicitKind);
72
73 // result is the new voice last appended measure
74 fVoiceLastAppendedMeasure = result;
75
76 #ifdef MF_TRACE_IS_ENABLED
77     if (gTrace0ahGroup->getTraceMeasuresDetails ()) {
78         displayVoice (
79             inputLineNumber,
80             "createAMeasureAndAppendItToVoice() 3");
81     }
82 #endif // MF_TRACE_IS_ENABLED
83
84 --gIndenter;
85
86 return result;
87 }

```

40.7 Translating from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

Upon the first visit of `S_measure`, as class `msrMeasure` instance is created and append to the current part:

```

1 void mxsr2msrTranslator::visitStart (S_measure& elt)
2 {
3     // ... ..
4
5     // take this measure into account
6     ++fPartMeasuresCounter;
7
8     // ... ..
9
10    // implicit
11
12    /*
13     Measures with an implicit attribute set to "yes"
14     never display a measure number,
15     regardless of the measure-numbering setting.
16     *
17     The implicit attribute is set to "yes" for measures where
18     the measure number should never appear, such as pickup
19     measures and the last half of mid-measure repeats. The
20     value is "no" if not specified.
21    */
22
23    std::string
24        implicit =
25        elt->getAttributeValue ("implicit");
26
27    msrMeasureImplicitKind
28        measureImplicitKind =
29        msrMeasureImplicitKind::kMeasureImplicitKindNo; // default value
30

```



```

31  if      (implicit == "yes") {
32      measureImplicitKind =
33          msrMeasureImplicitKind::kMeasureImplicitKindYes;
34  }
35  else if (implicit == "no") {
36      measureImplicitKind =
37          msrMeasureImplicitKind::kMeasureImplicitKindNo;
38  }
39  else {
40      if (implicit.size ()) {
41          std::stringstream ss;
42
43          ss <<
44              "implicit \"" << implicit <<
45              "\" is unknown";
46
47          musicxmlError (
48              gServiceRunData->getInputSourceName (),
49              inputLineNumber,
50              __FILE__, __LINE__,
51              ss.str ());
52      }
53  }
54
55  // append a new measure to the current part
56  fCurrentPart->
57      createAMeasureAndAppendItToPart (
58          inputLineNumber,
59          fCurrentMeasureNumber,
60          measureImplicitKind);
61
62  // ... ..
63 }

```

Upon the second visit of `S_measure`, the last appended measure appended to the current part is finalized:

```

1  void mxsr2msrTranslator::visitEnd (S_measure& elt)
2  {
3      // ... ..
4
5      // finalize current measure in the part,
6      // to add skips if necessary and set measure kind
7      fCurrentPart->
8          finalizeLastAppendedMeasureInPart (
9              inputLineNumber);
10
11     // ... ..
12 }

```

40.8 Translating from MXSR to MSR

A new class `msrMeasure` instance is created in `src/passes/mxsr2msr/mxsr2msrTranslator.cpp` upon the first visit of `S_measure`:

```

1  // -----
2  void mxsr2msrTranslator::visitStart (S_measure& elt)
3  {
4      // ... ..
5
6      // append a new measure to the current part
7      fCurrentPart->
8          createAMeasureAndAppendItToPart (
9              inputLineNumber,

```

```

10         fCurrentMeasureNumber,
11         measureImplicitKind);
12
13     // ... ..
14 }

```

This can lead to several class `msrMeasure` instances being created, depending on the MusicXML data. Hence there is no notion of a current measure in this translator.

Method `msrPart::createAMeasureAndAppendItToPart ()` creates and appends a measure to the part harmonies and figured bass staves if relevant, and then cascade s to the part staves:

```

1 void msrPart::createAMeasureAndAppendItToPart (
2     int         inputLineNumber,
3     std::string measureNumber,
4     msrMeasureImplicitKind
5     measureImplicitKind)
6 {
7     // ... ..
8
9     // set part current measure number
10    fPartCurrentMeasureNumber = measureNumber;
11
12    // create and append measure in all the staves
13    for (S_msrStaff staff : fPartAllStavesList) {
14        staff->
15            createAMeasureAndAppendItToStaff (
16                inputLineNumber,
17                measureNumber,
18                measureImplicitKind);
19    } // for
20
21    // ... ..

```

40.9 Translating from MSR to MSR

This is done in `src/passes/msr2msr/`.

40.10 Translating from MSR to LPSR

This is done in `src/passes/msr2lpsr/`.

40.11 Translating from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

Chapter 41

Finalizations

41.1 Clones vs non-clones finalization

Finalizing clones may be simpler than finalizing a just-created and populated non-clone, due to the information available in the clone's original.

For example, method `msrMeasure::finalizeMeasure ()` delegates part of the job to methods handling the three kinds of voices, respectively:

```
1 void msrMeasure::finalizeMeasure (
2     int                inputLineNumber,
3     msrMeasureRepeatContextKind measureRepeatContextKind,
4     std::string        context)
5 {
6     // ... ..
7
8     S_msrVoice
9     voice =
10         fMeasureUpLinkToSegment ->
11         getSegmentUpLinkToVoice ();
12
13     // ... ..
14
15     // delegate to voice kind specific methods
16     switch (voice->getVoiceKind ()) {
17     case msrVoiceKind::kVoiceKindRegular:
18         finalizeRegularMeasure (
19             inputLineNumber,
20             measureRepeatContextKind,
21             context);
22         break;
23
24     case msrVoiceKind::kVoiceKindDynamics:
25         break;
26
27     case msrVoiceKind::kVoiceKindHarmonies:
28         finalizeHarmonyMeasure (
29             inputLineNumber,
30             measureRepeatContextKind,
31             context);
32         break;
33
34     case msrVoiceKind::kVoiceKindFiguredBass:
35         finalizeFiguredBassMeasure (
36             inputLineNumber,
37             measureRepeatContextKind,
38             context);
39         break;
```

```

40     } // switch
41
42     // ... ..
43 }

```

In the case of harmony and figured bass voices, padding may have to be added to obtain a complete measure. This does not happen for clones of such voices: the padding skips are in the original voice and will be visited and handled without anything special to be done.

41.2 The finalization methods

There is a set of virtual method `finalize* ()` methods in `MusicFormats`. There basic ones are:

- method `msrPart::finalizePart ()` and method `msrPart::finalizePartClone ()`, defined in `src/formats/msr/msrParts.h/.cpp`
- method `msrStaff::finalizeStaff ()`, defined in `src/formats/msr/msrStaves.h/.cpp`
- method `msrVoice::finalizeVoice ()`, defined in `src/formats/msr/msrVoices.h/.cpp`
- method `msrSegment::finalizeAllTheMeasuresOfSegment ()`, defined in `src/formats/msr/msrSegments.h/.cpp`
- method `msrMeasure::finalizeMeasure ()`, method `msrMeasure::finalizeMeasureClone ()` and method `msrMeasure::finalizeRegularMeasure ()`, defined in `src/formats/msr/msrMeasures.h/.cpp`
- method `msrChord::finalizeChord ()`, defined in `src/formats/msr/msrChords.h/.cpp`
- method `msrTuplets::finalizeTuplet ()`, defined in `src/formats/msr/msrTuplets.h/.cpp`
- method `mxsr2msrTranslator::finalizeTupletAndPopItFromTupletsStack ()`, defined in `src/passes/mxsr2msr/mxsr2msrTranslator.h.h/.cpp`
- method `msrMeasure::finalizeFiguredBassMeasure ()`, defined in `src/formats/msr/msrMeasures.h/.cpp`
- method `msrMeasure::finalizeHarmonyMeasure ()`, defined in `src/formats/msr/msrMeasures.h/.cpp`
- method `msr2bsrTranslator::finalizeCurrentMeasureClone ()`, defined in `src/passes/msr2bsr/(.h/.cppmsr2bsrTranslator)`
- method `mxsr2msrTranslator::finalizeCurrentChord ()`, defined in `src/passes/mxsr2msr/mxsr2msrTranslator.h/.cpp`

Handling repeats is rather complex in `MusicFormats`. Repeat ends are finalized with these methods:

- method `msrPart::finalizeRepeatEndInPart ()`,
defined in `src/formats/msr/msrParts.h/.cpp`
- method `msrStaff::finalizeRepeatEndInStaff ()`,
defined in `src/formats/msr/msrStaves.h/.cpp`
- method `msrVoice::finalizeRepeatEndInVoice ()`,
defined in `src/formats/msr/msrVoices.h/.cpp`

There are also 'cascading' finalization methods: they propagate finalization going from class `msrPart` towards class `msrVoice`:

- method `msrPart::finalizeLastAppendedMeasureInPart ()`,
defined in `src/formats/msr/msrParts.h/.cpp`
- method `msrStaff::finalizeLastAppendedMeasureInStaff ()`,
defined in `src/formats/msr/msrStaves.h/.cpp`
- method `msrVoice::finalizeLastAppendedMeasureInVoice ()`,
defined in `src/formats/msr/msrVoices.h/.cpp`
- method `msrPart::finalizePartAndAllItsMeasures ()`,
defined in `src/formats/msr/msrParts.h/.cpp`
- method `msrVoice::finalizeVoiceAndAllItsMeasures ()`,
defined in `src/formats/msr/msrVoices.h/.cpp`

A typical example of cascading finalization is method `msrPart::finalizePartAndAllItsMeasures ()`:

```

1 void msrPart::finalizePartAndAllItsMeasures (
2     int inputLineNumber)
3 {
4     #ifdef MF_TRACE_IS_ENABLED
5         if (gTraceOahGroup->getTraceParts ()) {
6             std::stringstream ss;
7
8             ss <<
9                 "Finalizing part clone " <<
10                getPartCombinedName () <<
11                ", line " << inputLineNumber;
12
13            gWaeHandler->waeTraceWithoutInputLocation (
14                __FILE__, __LINE__,
15                ss.str ());
16        }
17    #endif // MF_TRACE_IS_ENABLED
18
19    #ifdef MF_TRACE_IS_ENABLED
20        if (gTraceOahGroup->getTraceVoices ()) {
21            std::stringstream ss;
22
23            ss <<
24                "Finalizing all the measures of part \"" <<
25                getPartCombinedName () <<
26                "\", line " << inputLineNumber;
27
28            gWaeHandler->waeTraceWithoutInputLocation (
29                __FILE__, __LINE__,
30                ss.str ());
31        }
32    #endif // MF_TRACE_IS_ENABLED
33
34    for (
35        std::list<S_msrVoice>::const_iterator i = fPartAllVoicesList.begin ();
36        i != fPartAllVoicesList.end ();
37        ++i
38    ) {
39        const S_msrVoice& voice = (*i);
40
41        voice->
42            finalizeVoiceAndAllItsMeasures (
43                inputLineNumber);

```

```

44 } // for
45
46 // collect the part measures slices from the staves
47 collectPartMeasuresSlices (
48     inputLineNumber);
49 }

```

41.3 Finalizing parts

Method `msrPart::finalizePart ()` warns if there are no staves in the part, and otherwise handles them, including cascading to method `msrStaff::finalizeStaff ()`:

```

1 void msrPart::finalizePart (
2     int inputLineNumber)
3 {
4     #ifdef MF_TRACE_IS_ENABLED
5         if (gTraceOahGroup->getTraceParts ()) {
6             std::stringstream ss;
7
8             ss <<
9                 "Finalizing part " <<
10                getPartCombinedName () <<
11                ", line " << inputLineNumber;
12
13            gWaeHandler->waeTraceWithoutInputLocation (
14                __FILE__, __LINE__,
15                ss.str ());
16        }
17    #endif // MF_TRACE_IS_ENABLED
18
19    ++gIndenter;
20
21    if (! getPartStaveNumbersToStavesMap.size ()) {
22        std::stringstream ss;
23
24        ss <<
25            "Part " <<
26            getPartCombinedName () <<
27            " appears in the part list, but doesn't contain any staff";
28
29        musicxmlWarning (
30            gServiceRunData->getInputSourceName (),
31            inputLineNumber,
32            ss.str ());
33    }
34
35    else {
36        // sort the staves to have harmonies above and figured bass below the part
37        fPartAllStavesList.sort (
38            compareStavesToHaveFiguredBassesBelowCorrespondingPart);
39
40        // finalize the staves
41        for (
42            std::map<int, S_msrStaff>::const_iterator i =
43            getPartStaveNumbersToStavesMap.begin ();
44            i != getPartStaveNumbersToStavesMap.end ();
45            ++i
46        ) {
47            S_msrStaff staff = (*i).second;
48
49            staff->
50                finalizeStaff (
51                    inputLineNumber);

```

```

52     } // for
53 }
54
55 // set score instrument names max lengths if relevant
56 setPartInstrumentNamesMaxLengthes ();
57
58 // collect the part measures slices from the staves
59 collectPartMeasuresSlices (
60     inputLineNumber);
61
62 --gIndenter;
63 }

```

41.4 Finalizing staves

Method `msrStaff::finalizeStaff ()` cascade s to method `msrVoice::finalizeVoice ()` and collects the staff measures slices:

```

1 void msrStaff::finalizeStaff (int inputLineNumber)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gTraceOahGroup->getTraceStaves ()) {
5             std::stringstream ss;
6
7             ss <<
8                 "Finalizing staff \"" <<
9                 getStaffName () << "\" " <<
10                ", line " << inputLineNumber;
11
12             gWaeHandler->waeTraceWithoutInputLocation (
13                 __FILE__, __LINE__,
14                 ss.str ());
15         }
16     #endif // MF_TRACE_IS_ENABLED
17
18     ++gIndenter;
19
20     // finalize the voices
21     #ifdef MF_TRACE_IS_ENABLED
22         if (gTraceOahGroup->getTraceVoices ()) {
23             std::stringstream ss;
24
25             ss <<
26                 "Finalizing the voices in staff \"" <<
27                 getStaffName () << "\" " <<
28                 ", line " << inputLineNumber;
29
30             gWaeHandler->waeTraceWithoutInputLocation (
31                 __FILE__, __LINE__,
32                 ss.str ());
33         }
34     #endif // MF_TRACE_IS_ENABLED
35
36     for (
37         std::map<int, S_msrVoice>::const_iterator i =
38             fStaffVoiceNumbersToAllVoicesMap.begin ();
39         i != fStaffVoiceNumbersToAllVoicesMap.end ();
40         ++i
41     ) {
42         S_msrVoice
43             voice = (*i).second;
44
45         voice->

```



```

46         finalizeVoice (
47             inputLineNumber);
48     } // for
49
50     // collect the staff measures slices from the voices
51     collectStaffMeasuresSlices (
52         inputLineNumber);
53
54     --gIndenter;
55 }

```

41.5 Finalizing voices

Method `msrVoice::finalizeVoice ()` handles pending repeats if any and collects the voice measures into a flat list. It does not, however, cascade to finalizing the voice repeats and measures.

```

1 void msrVoice::finalizeVoice (
2     int inputLineNumber)
3 {
4     // ... ..
5
6     if (fVoiceHasBeenFinalized) {
7         std::stringstream ss;
8
9         ss <<
10            "Attempting to finalize voice \"" <<
11            asShortString () <<
12            "\" more than once";
13
14         msrInternalError (
15             gServiceRunData->getInputSourceName (),
16             fInputStartLineNumber,
17             __FILE__, __LINE__,
18             ss.str ());
19     }
20
21     // set part shortest note duration if relevant
22     S_msrPart
23         voicePart =
24             fetchVoiceUpLinkToPart ();
25
26     mfRational
27         partShortestNoteWholeNotes =
28             voicePart->
29                 getPartShortestNoteWholeNotes ();
30
31     // ... ..
32
33     if (fVoiceShortestNoteWholeNotes < partShortestNoteWholeNotes) {
34         // set the voice part shortest note duration
35         voicePart->
36             setPartShortestNoteWholeNotes (
37                 fVoiceShortestNoteWholeNotes);
38
39         // set the voice part shortest note tuplet factor // JMI
40         voicePart->
41             setPartShortestNoteTupletFactor (
42                 fVoiceShortestNoteTupletFactor);
43     }
44
45     // is this voice totally empty? this should be rare...
46     if (
47         fVoiceInitialElementsList.size () == 0

```

```

48     &&
49     fVoiceLastSegment->getSegmentElementsList ().size () == 0
50 ) {
51     std::stringstream ss;
52
53     ss <<
54     "Voice \"" <<
55     getVoiceName () <<
56     "\" is totally empty, no contents ever specified for it" <<
57     std::endl;
58
59     musicxmlWarning (
60         gServiceRunData->getInputSourceName (),
61         inputLineNumber,
62         ss.str ());
63 }
64
65 // are there pending repeats in the voice repeats stack???
66 unsigned int voicePendingRepeatDescrsStackSize =
67     fVoicePendingRepeatDescrsStack.size ();
68
69 // ... ..
70
71 // collect the voice measures into the flat list
72 collectVoiceMeasuresIntoFlatList (
73     inputLineNumber);
74
75 fVoiceHasBeenFinalized = true;
76
77 // ... ..
78 }

```

41.6 Finalizing repeats

41.7 Finalizing measures

Method `msrMeasure::finalizeMeasure ()` is not cascaded. It delegates finalization to voice kind specific methods presented in the subsections below, handles pending repeats if any, and assigns positions in the measure to the measure's elements:

```

1 void msrMeasure::finalizeMeasure (
2     int inputLineNumber,
3     msrMeasureRepeatContextKind measureRepeatContextKind,
4     std::string context)
5 {
6     if (fMeasureHasBeenFinalized) {
7         std::stringstream ss;
8
9         ss <<
10         "Attempting to finalize measure " <<
11         this->asShortString () <<
12         " more than once in segment '" <<
13         fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
14         "', context: " << context <<
15         "', measureFinalizationContext: " << fMeasureFinalizationContext <<
16         " in voice \"" <<
17         fMeasureUpLinkToSegment->
18         getSegmentUpLinkToVoice ()->
19         getVoiceName () <<
20         "\" (" << context << ")" <<
21         ", line " << inputLineNumber;
22

```

```

23 // ... ..
24
25 msrInternalWarning (
26     gServiceRunData->getInputSourceName (),
27     fInputStartLineNumber,
28     ss.str ());
29 }
30
31 else {
32     S_msrVoice
33     voice =
34         fMeasureUpLinkToSegment->
35         getSegmentUpLinkToVoice ();
36
37 // ... ..
38
39 // delegate to voice kind specific methods
40 switch (voice->getVoiceKind ()) {
41     case msrVoiceKind::kVoiceKindRegular:
42         finalizeRegularMeasure (
43             inputLineNumber,
44             measureRepeatContextKind,
45             context);
46         break;
47
48     case msrVoiceKind::kVoiceKindDynamics:
49         break;
50
51     case msrVoiceKind::kVoiceKindHarmonies:
52         finalizeHarmonyMeasure (
53             inputLineNumber,
54             measureRepeatContextKind,
55             context);
56         break;
57     case msrVoiceKind::kVoiceKindFiguredBass:
58         finalizeFiguredBassMeasure (
59             inputLineNumber,
60             measureRepeatContextKind,
61             context);
62         break;
63 } // switch
64
65 // position in voice
66 mfRational
67 voicePosition =
68     fetchMeasureUpLinkToVoice ()->
69     getCurrentVoicePosition ();
70
71 // assign measure' elements measure position
72 for (
73     std::list<S_msrMeasureElement>::const_iterator i = fMeasureElementsList.begin ();
74     i != fMeasureElementsList.end ();
75     ++i
76 ) {
77     S_msrMeasureElement measureElement = (*i);
78
79     measureElement->
80         setVoicePosition (
81             voicePosition,
82             "finalizeMeasure()");
83 } // for
84
85 // register finalization
86 fMeasureHasBeenFinalized = true;
87 fMeasureFinalizationContext = context;
88 }
89 }

```

41.7.1 Finalizing regular measures

```

1 void msrMeasure::finalizeRegularMeasure (
2     int inputLineNumber,
3     msrMeasureRepeatContextKind measureRepeatContextKind,
4     std::string context)
5 {
6     // fetch the regular voice
7     S_msrVoice
8     voice =
9         fMeasureUpLinkToSegment->
10         getSegmentUpLinkToVoice ();
11
12     // fetch the regular voice's part
13     S_msrPart
14     regularPart =
15         voice->
16         fetchVoiceUpLinkToPart ();
17
18     mfAssert (
19         __FILE__, __LINE__,
20         regularPart != nullptr,
21         "regularPart is null");
22
23     if (false) { // JMI
24         gLog <<
25             "---> regularPart: " <<
26             std::endl;
27
28         ++gIndenter;
29         gLog <<
30             regularPart <<
31             std::endl;
32         --gIndenter;
33         gLog << std::endl;
34     }
35
36     mfRational
37     measureCurrentAccumulatedWholeNotesDurationFromPartMeasuresVector =
38         regularPart->
39         getPartMeasuresWholeNotessVector () [
40             fMeasureOrdinalNumberInVoice - 1 ];
41
42 #ifdef MF_TRACE_IS_ENABLED
43     if (gTraceOahGroup->getTraceMeasures ()) {
44         std::stringstream ss;
45
46         ss <<
47             "Finalizing regular measure " <<
48             this->asShortString () <<
49             " in segment '" <<
50             fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
51             "' in regular voice \"" <<
52             voice->getVoiceName () <<
53             "\" (" << context << ")" <<
54             ", measureCurrentAccumulatedWholeNotesDurationFromPartMeasuresVector: " <<
55             measureCurrentAccumulatedWholeNotesDurationFromPartMeasuresVector <<
56             ", line " << inputLineNumber;
57
58         gWaeHandler->waeTraceWithoutInputLocation (
59             __FILE__, __LINE__,
60             ss.str ());
61     }

```

```

62 #endif // MF_TRACE_IS_ENABLED
63
64 ++gIndenter;
65
66 #ifdef MF_TRACE_IS_ENABLED
67 if (gTraceOahGroup->getTraceMeasuresDetails ()) {
68     displayMeasure (
69         inputLineNumber,
70         "finalizeRegularMeasure() 1");
71 }
72 #endif // MF_TRACE_IS_ENABLED
73
74 padUpToPositionAtTheEndOfTheMeasure (
75     inputLineNumber,
76     measureCurrentAccumulatedWholeNotesDurationFromPartMeasuresVector);
77
78 // register this measures's length in the part
79 S_msrPart
80 part =
81     this->fetchMeasureUpLinkToPart ();
82
83 part->
84     registerOrdinalMeasureNumberWholeNotes (
85         inputLineNumber,
86         fMeasureOrdinalNumberInVoice,
87         fCurrentMeasureCurrentAccumulatedWholeNotesDuration);
88
89 // determine the measure kind and purist number
90 determineMeasureKindAndPuristNumber (
91     inputLineNumber,
92     measureRepeatContextKind);
93
94 // pad measure up to whole measure whole notes high tide JMI ???
95 switch (fMeasureKind) {
96     case msrMeasureKind::kMeasureKindCadenza:
97         break;
98
99     case msrMeasureKind::kMeasureKindOvercomplete:
100     case msrMeasureKind::kMeasureKindAnacrusis:
101     case msrMeasureKind::kMeasureKindRegular:
102     case msrMeasureKind::kMeasureKindIncompleteStandalone: // JMI
103     case msrMeasureKind::kMeasureKindIncompleteLastMeasure: // JMI
104     case msrMeasureKind::kMeasureKindIncompleteLastInRepeatCommonPart: // JMI
105     case msrMeasureKind::kMeasureKindIncompleteLastInRepeatHookedEnding: // JMI
106     case msrMeasureKind::kMeasureKindIncompleteLastInRepeatHooklessEnding: // JMI
107     case msrMeasureKind::kMeasureKindIncompleteNextMeasureAfterCommonPart: // JMI
108     case msrMeasureKind::kMeasureKindIncompleteNextMeasureAfterHookedEnding: // JMI
109     case msrMeasureKind::kMeasureKindIncompleteNextMeasureAfterHooklessEnding: // JMI
110         break;
111
112     case msrMeasureKind::kMeasureKindUnknown:
113         // JMI ???
114         break;
115
116     case msrMeasureKind::kMeasureKindMusicallyEmpty:
117     {
118         /* JMI
119          */
120     }
121     break;
122 } // switch
123
124 // is there a single note or rest occupying the full measure?
125 if (fMeasureLongestNote) {
126     if (
127         fMeasureLongestNote->getSoundingWholeNotes ()
128         ==

```

```

129     fFullMeasureWholeNotesDuration
130 ) {
131 #ifdef MF_TRACE_IS_ENABLED
132     if (gTraceOahGroup->getTraceMeasures ()) {
133         gLog <<
134             "Note '" <<
135             fMeasureLongestNote->asShortString () <<
136             "' occupies measure " <<
137             this->asShortString () <<
138             " fully in segment '" <<
139             fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
140             "' in voice \"" <<
141             voice->getVoiceName () <<
142             "\", line " << inputLineNumber <<
143             std::endl;
144
145         gWaeHandler->waeTraceWithoutInputLocation (
146             __FILE__, __LINE__,
147             ss.str ());
148     }
149 #endif // MF_TRACE_IS_ENABLED
150
151     fMeasureLongestNote->
152     setNoteOccupiesAFullMeasure ();
153 }
154 }
155
156 #ifdef MF_TRACE_IS_ENABLED
157     if (gTraceOahGroup->getTraceMeasuresDetails ()) {
158         displayMeasure (
159             inputLineNumber,
160             "finalizeRegularMeasure() 2");
161     }
162 #endif // MF_TRACE_IS_ENABLED
163
164 --gIndenter;
165 }

```

41.7.2 Finalizing harmonies measures

```

1 void msrMeasure::finalizeHarmonyMeasure (
2     int inputLineNumber,
3     msrMeasureRepeatContextKind measureRepeatContextKind,
4     std::string context)
5 {
6     // fetch the harmonies voice
7     S_msrVoice
8     harmoniesVoice =
9     fMeasureUpLinkToSegment->
10     getSegmentUpLinkToVoice ();
11
12     // fetch the harmonies part
13     S_msrPart
14     harmoniesPart =
15     harmoniesVoice->
16     fetchVoiceUpLinkToPart ();
17
18     mfAssert (
19         __FILE__, __LINE__,
20         harmoniesPart != nullptr,
21         "harmoniesPart is null");
22
23 #ifdef MF_TRACE_IS_ENABLED
24     if (gTraceOahGroup->getTraceHarmonies ()) {

```

```

25     std::stringstream ss;
26
27     ss <<
28         "Finalizing harmonies measure " <<
29         this->asShortString () <<
30         " in segment '" <<
31         fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
32         "' in harmonies voice \"" <<
33         harmoniesVoice->getVoiceName () <<
34         "\" (" << context << ")" <<
35         ", line " << inputLineNumber;
36
37     gWaeHandler->waeTraceWithoutInputLocation (
38         __FILE__, __LINE__,
39         ss.str ());
40 }
41 #endif // MF_TRACE_IS_ENABLED
42
43 ++gIndenter;
44
45 #ifdef MF_TRACE_IS_ENABLED
46     if (gTraceOahGroup->getTraceHarmoniesDetails ()) {
47         displayMeasure (
48             inputLineNumber,
49             "finalizeHarmonyMeasure() 1");
50     }
51 #endif // MF_TRACE_IS_ENABLED
52
53 #ifdef MF_TRACE_IS_ENABLED
54     // get the harmoniesPart number of measures
55     int
56     harmoniesPartNumberOfMeasures =
57     harmoniesPart->
58         getPartNumberOfMeasures ();
59
60     if (gTraceOahGroup->getTraceHarmonies ()) {
61         std::stringstream ss;
62
63         ss <<
64             "fMeasureOrdinalNumberInVoice: \"" <<
65             fMeasureOrdinalNumberInVoice <<
66             ", harmoniesPartNumberOfMeasures: \"" <<
67             harmoniesPartNumberOfMeasures;
68
69         gWaeHandler->waeTraceWithoutInputLocation (
70             __FILE__, __LINE__,
71             ss.str ());
72     }
73 #endif // MF_TRACE_IS_ENABLED
74
75     // the measureCurrentAccumulatedWholeNotesDuration has to be computed
76     mfRational
77     measureCurrentAccumulatedWholeNotesDurationFromPartMeasuresVector =
78     harmoniesPart->
79         getPartMeasuresWholeNotessVector () [
80             fMeasureOrdinalNumberInVoice - 1 ];
81
82     // handle the harmonies in this measure
83     finalizeTheHarmoniesInAHarmoniesMeasure (
84         inputLineNumber,
85         context);
86
87     // pad the measure up to
88     measureCurrentAccumulatedWholeNotesDurationFromPartMeasuresVector
89     padUpToPositionAtTheEndOfTheMeasure (
90         inputLineNumber,
91         measureCurrentAccumulatedWholeNotesDurationFromPartMeasuresVector);

```

```

91
92 // determine the measure kind and purist number
93 determineMeasureKindAndPuristNumber (
94     inputLineNumber,
95     measureRepeatContextKind);
96
97 #ifdef MF_TRACE_IS_ENABLED
98     if (gTraceOahGroup->getTraceHarmoniesDetails ()) {
99         displayMeasure (
100             inputLineNumber,
101             "finalizeHarmonyMeasure() 2");
102     }
103 #endif // MF_TRACE_IS_ENABLED
104
105 --gIndenter;
106 }

```

41.7.3 Finalizing figured bass measures

```

1 void msrMeasure::finalizeFiguredBassMeasure (
2     int inputLineNumber,
3     msrMeasureRepeatContextKind measureRepeatContextKind,
4     std::string context)
5 {
6     // fetch the figured bass voice
7     S_msrVoice
8         figuredBassVoice =
9         fMeasureUpLinkToSegment->
10         getSegmentUpLinkToVoice ();
11
12     // fetch the figured bass part
13     S_msrPart
14         figuredBassPart =
15         figuredBassVoice->
16         fetchVoiceUpLinkToPart ();
17
18     mfAssert (
19         __FILE__, __LINE__,
20         figuredBassPart != nullptr,
21         "figuredBassPart is null");
22
23 #ifdef MF_TRACE_IS_ENABLED
24     if (gTraceOahGroup->getTraceFiguredBasses ()) {
25         std::stringstream ss;
26
27         ss <<
28             "Finalizing figured bass measure " <<
29             this->asShortString () <<
30             " in segment '" <<
31             fMeasureUpLinkToSegment->getSegmentAbsoluteNumber () <<
32             "' in figured bass voice \"" <<
33             figuredBassVoice->getVoiceName () <<
34             "\" (" << context << ")" <<
35             ", line " << inputLineNumber;
36
37         gWaeHandler->waeTraceWithoutInputLocation (
38             __FILE__, __LINE__,
39             ss.str ());
40     }
41 #endif // MF_TRACE_IS_ENABLED
42
43 ++gIndenter;
44
45 #ifdef MF_TRACE_IS_ENABLED

```



```

46  if (gTraceOahGroup->getTraceFiguredBassesDetails ()) {
47      displayMeasure (
48          inputLineNumber,
49          "finalizeFiguredBassMeasure() 1");
50  }
51  #endif // MF_TRACE_IS_ENABLED
52
53  #ifdef MF_TRACE_IS_ENABLED
54      // get the figuredBassPart number of measures
55      int
56      figuredBassPartNumberOfMeasures =
57          figuredBassPart->
58              getPartNumberOfMeasures ();
59
60      if (gTraceOahGroup->getTraceHarmonies ()) {
61          std::stringstream ss;
62
63          ss <<
64              "fMeasureOrdinalNumberInVoice: \"" <<
65              fMeasureOrdinalNumberInVoice <<
66              ", figuredBassPartNumberOfMeasures: \"" <<
67              figuredBassPartNumberOfMeasures;
68
69          gWaeHandler->waeTraceWithoutInputLocation (
70              __FILE__, __LINE__,
71              ss.str ());
72      }
73  #endif // MF_TRACE_IS_ENABLED
74
75      // the measureCurrentAccumulatedWholeNotesDuration has to be computed
76      mfRational
77      measureCurrentAccumulatedWholeNotesDuration =
78          figuredBassPart->
79              getPartMeasuresWholeNotesVector () [
80                  fMeasureOrdinalNumberInVoice - 1 ];
81
82      // handle the figured bass elements in this measure
83      finalizeTheFiguredBassesInAFiguredBassMeasure (
84          inputLineNumber,
85          context);
86
87      // pad the measure up to fFullMeasureWholeNotesDuration
88      padUpToPositionAtTheEndOfTheMeasure (
89          inputLineNumber,
90          measureCurrentAccumulatedWholeNotesDuration);
91
92      // determine the measure kind and purist number
93      determineMeasureKindAndPuristNumber (
94          inputLineNumber,
95          measureRepeatContextKind);
96
97  #ifdef MF_TRACE_IS_ENABLED
98      if (gTraceOahGroup->getTraceFiguredBassesDetails ()) {
99          displayMeasure (
100              inputLineNumber,
101              "finalizeFiguredBassMeasure() 2");
102      }
103  #endif // MF_TRACE_IS_ENABLED
104
105      --gIndenter;
106  }

```

41.8 Determining measure positionss

Chapter 42

Tempos handling

Tempos are presented at section [20.17](#) [Tempos], page [193](#).

42.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

42.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

42.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

42.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

42.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 43

Notes handling

Notes are presented at section ?? [Notes], page ??.

43.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

43.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

43.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

43.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

43.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 44

Segments handling

Segments are presented at section [20.36](#) [Segments], page [211](#).

The segments concept used by MusicFormats to describe music scores is not apparent to the users of GUI applications, in which music elements are *drawn* on the page. Their need is inherent to the representation of repeats, which contain music elements sequences (the segments) and even other repeats.

ALL SEGMENTS HANDLING in MusicFormats IS DONE INTERNALLY: the class `msrSegment` instances are created in voices and repeats BEHIND THE CURTAINS.

44.1 Segments creation

Instances of class `msrSegment` are created at four places in `src/formats/msr/msrVoices.cpp`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'msrSegment::create ('  
   *.cpp  
2 msrSegments.cpp:S_msrSegment msrSegment::create (  
3 msrSegments.cpp:      msrSegment::create (  
4 msrSegments.cpp:      msrSegment::create (  
5 msrVoices.cpp:      msrSegment::create (  
6 msrVoices.cpp:      msrSegment::create (  
7 msrVoices.cpp:      msrSegment::create (  
8 msrVoices.cpp:      msrSegment::create (  
9
```

Calls to method `msrSegment::createSegmentNewbornClone ()` occurs only when visiting class `msrSegment` instances in passes:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/passes > grep createSegmentNewbornClone  
   */*  
2 msr2bsr/msr2bsrTranslator.cpp:      elt->createSegmentNewbornClone (  
3 msr2lpsr/msr2lpsrTranslator.cpp:      elt->createSegmentNewbornClone (  
4 msr2msr/msr2msrTranslator.cpp:      elt->createSegmentNewbornClone (  
5
```

Method `msrSegment::createSegmentDeepClone ()` is not used at the time of this writing.

Explicit segments creation is thus entirely done in methods inside `src/formats/msr/msrVoices.cpp`: the passes are not aware of this happening.

The first occurrence of method `msrSegment::create ()` is in method `msrVoice::initializeVoice ()`: when a voice is created, a segment is created and stored in its `fVoiceLastSegment` if requested :

```

1 void msrVoice::initializeVoice (
2     msrVoiceCreateInitialLastSegmentKind
3     voiceCreateInitialLastSegmentKind)
4 {
5     // ... ..
6
7     // create the initial last segment if requested
8     switch (voiceCreateInitialLastSegmentKind) {
9         case msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes:
10            // sanity check // JMI LAST
11            mfAssert (
12                __FILE__, __LINE__,
13                fVoiceLastSegment == nullptr,
14                "fVoiceLastSegment is null");
15
16            // create the last segment
17            fVoiceLastSegment =
18                msrSegment::create (
19                    fInputStartLineNumber,
20                    this);
21
22            if (! fVoiceFirstSegment) {
23                fVoiceFirstSegment = fVoiceLastSegment;
24            }
25            break;
26        case msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentNo:
27            break;
28    } // switch
29
30    // ... ..
31 };

```

Method `msrVoice::createMeasureRepeatFromItsFirstMeasures ()` is presented in section 46 [Measure repeats handling], page 322, and the remaining two are presented in the next sections.

44.1.1 Creating a new last segment for a voice

There is method `msrVoice::createNewLastSegmentForVoice ()`, called at many places in `src/formats/msr/msrVoices.cpp`:

```

1 void msrVoice::createNewLastSegmentForVoice (
2     int    inputLineNumber,
3     const std::string& context)
4 {
5     // ... ..
6
7     // create the last segment
8     fVoiceLastSegment =
9         msrSegment::create (
10             inputLineNumber,
11             this);
12
13     if (! fVoiceFirstSegment) {
14         fVoiceFirstSegment = fVoiceLastSegment;
15     }
16
17     // ... ..
18 }

```

The calls to method `msrVoice::createNewLastSegmentForVoice ()` are in:

- method `msrVoice::createAMeasureAndAppendItToVoice ()`
- method `msrVoice::appendStaffDetailsToVoice ()`
- method `msrVoice::addGraceNotesGroupBeforeAheadOfVoiceIfNeeded ()`
- method `msrVoice::handleVoiceLevelRepeatStart ()`
- method `msrVoice::handleVoiceLevelRepeatEndWithoutStart ()`
- method `msrVoice::handleVoiceLevelContainingRepeatEndWithoutStart ()`
- method `msrVoice::handleVoiceLevelRepeatEndWithStart ()`
- method `msrVoice::handleVoiceLevelRepeatEndingStartWithoutExplicitStart ()`
- method `msrVoice::handleVoiceLevelRepeatEndingStartWithExplicitStart ()`
- method `msrVoice::handleMultiMeasureRestsStartInVoiceClone ()`
- method `msrVoice::handleHooklessRepeatEndingEndInVoice ()`
- method `msrVoice::appendBarLineToVoice ()`
- method `msrVoice::appendSegnoToVoice ()`
- method `msrVoice::appendCodaToVoice ()`
- method `msrVoice::appendEyeGlassesToVoice ()`
- method `msrVoice::appendPedalToVoice ()`
- method `msrVoice::appendDampToVoice ()`
- method `msrVoice::appendDampAllToVoice ()`

44.1.2 Creating a new last segment for a voice from its first measure

Method `msrVoice::createNewLastSegmentFromItsFirstMeasureForVoice ()` is used at several places in [src/formats/msr/msrVoices.cpp](#):

```

1 void msrVoice::createNewLastSegmentFromItsFirstMeasureForVoice (
2     int          inputLineNumber,
3     S_msrMeasure firstMeasure,
4     std::string  context)
5 {
6     // create the last segment
7     fVoiceLastSegment =
8         msrSegment::create (
9             inputLineNumber,
10            this);
11
12     if (! fVoiceFirstSegment) {
13         fVoiceFirstSegment = fVoiceLastSegment;
14     }
15
16     // ... ..
17

```

```

18 // append firstMeasure to fVoiceLastSegment
19 fVoiceLastSegment->
20     appendMeasureToSegment (firstMeasure);
21
22 // firstMeasure is the new voice last appended measure
23 fVoiceLastAppendedMeasure = firstMeasure;
24
25 // is firstMeasure the first one 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 324), measure repeats (section 46 [Measure repeats handling], page 322) and multi-measure rests (section 47 [Multi-measure rests handling], page 323).

44.2 Appending measures to a segment

Method `msrSegment::assertSegmentElementsListIsNotEmpty ()` is called as a sanity check by many methods in `src/formats/msr/msrSegments.cpp`:

```

1 void msrSegment::assertSegmentElementsListIsNotEmpty (
2     int inputLineNumber) const
3 {
4     if (! fSegmentElementsList.size ()) {
5 #ifdef MF_TRACE_IS_ENABLED
6         if (
7             gTraceOahGroup->getTraceMeasuresDetails ()
8             ||
9             gTraceOahGroup->getTraceSegmentsDetails ()
10            ||
11            gTraceOahGroup->getTraceRepeatsDetails ()
12        ) {
13            fSegmentUpLinkToVoice->
14                displayVoiceRepeatsStackMultiMeasureRestsMeasureRepeatAndVoice (
15                inputLineNumber,
16                "assertSegmentElementsListIsNotEmpty()");
17        }
18 #endif // MF_TRACE_IS_ENABLED
19
20        gLog <<
21            "assertSegmentElementsListIsNotEmpty()" <<
22            ", fSegmentElementsList is empty" <<
23            ", segment: " <<
24            this->asString () <<
25            ", in voice \"" <<
26            fSegmentUpLinkToVoice->getVoiceName () <<
27            "\" " <<
28            ", line " << inputLineNumber <<
29            std::endl;
30
31        mfAssert (
32            __FILE__, __LINE__,
33            false,
34            ", fSegmentElementsList is empty");

```

```

35 }
36 }

```

One such call is:

```

1 void msrSegment::appendKeyToSegment (
2     const S_msrKey& key)
3 {
4     #ifdef MF_TRACE_IS_ENABLED
5         if (gTraceOahGroup->getTraceKeys ()) {
6             std::stringstream ss;
7
8             ss <<
9                 "Appending key " << key
10                " to segment " << asString () <<
11                ", in voice \"" <<
12                fSegmentUpLinkToVoice->getVoiceName () <<
13                "\"";
14
15            gWaeHandler->waeTraceWithoutInputLocation (
16                __FILE__, __LINE__,
17                ss.str ());
18        }
19    #endif // MF_TRACE_IS_ENABLED
20
21    #ifdef MF_SANITY_CHECKS_ARE_ENABLED
22        // sanity check
23        assertSegmentElementsListIsNotEmpty (
24            key->getInputStartLineNumber ());
25    #endif // MF_SANITY_CHECKS_ARE_ENABLED
26
27    ++gIndenter;
28
29    // register key in segments's current measure
30    fSegmentElementsList.back ()->
31        appendKeyToMeasure (key);
32
33    --gIndenter;
34 }

```

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 [20.31](#) [Beat repeats], page [207](#).

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 [20.32](#) [Measure repeats], page [208](#).

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

Multi-measure rests handling

Multi-measure rests are presented at section [20.33](#) [Multi-measure rests], page [209](#).

Chapter 48

Repeats handling

Repeats are presented at section [20.35](#) [Repeats], page [210](#).

48.1 Translating repeats from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

The tough part is to handle MusicXML `<barLine/>` markups, since they are meant for drawing, and do not structure repeats as such.

Recognizing the structure of repeat relies on the attributes of the barLines. The following enumeration types are defined in `src/formats/msr/msrBarLines.h` local to class `msrBarLine`:

```
1 // location
2 enum class msrBarLineLocationKind {
3     kBarLineLocationNone,
4
5     kBarLineLocationLeft,
6     kBarLineLocationMiddle,
7     kBarLineLocationRight // by default
8 };
```

```
1 // style
2 enum class msrBarLineStyleKind {
3     kBarLineStyleNone,
4
5     kBarLineStyleRegular, // by default
6
7     kBarLineStyleDotted, kBarLineStyleDashed, kBarLineStyleHeavy,
8     kBarLineStyleLightLight, kBarLineStyleLightHeavy,
9     kBarLineStyleHeavyLight, kBarLineStyleHeavyHeavy,
10    kBarLineStyleTick, kBarLineStyleShort
11 };
```

```
1 // repeat direction
2 enum class msrBarLineRepeatDirectionKind {
3     kBarLineRepeatDirectionNone,
4     kBarLineRepeatDirectionForward, kBarLineRepeatDirectionBackward
5 };
```

```

1 // ending type
2 enum class msrBarLineEndingTypeKind {
3     kBarLineEndingTypeNone,
4
5     msrBarLineEndingTypeKind::kBarLineEndingTypeStart,
6     msrBarLineEndingTypeKind::kBarLineEndingTypeStop,
7     msrBarLineEndingTypeKind::kBarLineEndingTypeDiscontinue
8 };

```

```

1 // category
2 enum class msrBarLineCategoryKind {
3     kBarLineCategory_UNKNOWN_,
4
5     kBarLineCategoryStandalone,
6
7     kBarLineCategoryRepeatStart, kBarLineCategoryRepeatEnd,
8
9     kBarLineCategoryHookedEndingStart, kBarLineCategoryHookedEndingEnd,
10    kBarLineCategoryHooklessEndingStart, kBarLineCategoryHooklessEndingEnd
11 };

```

```

1 // segno
2 enum class msrBarLineHasSegnoKind {
3     kBarLineHasSegnoYes, kBarLineHasSegnoNo
4 };

```

```

1 // coda
2 enum class msrBarLineHasCodaKind {
3     kBarLineHasCodaYes, kBarLineHasCodaNo
4 };

```

```

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

```

The attributes of <barline/> are deciphered upon the first visit of S_barline in [src/passes/mxsr2msr/mxsr2msrTranslator.cpp](#):

```

1 void mxsr2msrTranslator::visitStart ( S_barline& elt )
2 {
3     // ... ..
4
5     // location
6
7     {
8         std::string
9             location =
10             elt->getAttributeValue ( "location" );
11
12         fCurrentBarLineLocationKind =
13             msrBarLineLocationKind::kBarLineLocationRight; // by default
14
15         if (location == "left") {
16             fCurrentBarLineLocationKind = msrBarLineLocationKind::kBarLineLocationLeft;
17         }
18         else if (location == "middle") {
19             fCurrentBarLineLocationKind = msrBarLineLocationKind::kBarLineLocationMiddle;
20         }
21         else if (location == "right") {
22             fCurrentBarLineLocationKind = msrBarLineLocationKind::kBarLineLocationRight;

```

```

23     }
24     else {
25         std::stringstream ss;
26
27         ss <<
28             "barLine location \"" << location <<
29             "\" is unknown, using 'right' by default";
30
31         // JMI    musicxmlError (
32             musicxmlWarning (
33                 gServiceRunData->getInputSourceName (),
34                 inputLineNumber,
35                 //    __FILE__, __LINE__,
36                 ss.str ());
37     }
38 }
39
40 fOnGoingBarLine = true;
41 }

```

Then the class `msrBarLine` instance is created upon the second visit of `S_barline`:

```

1 void mxsr2msrTranslator::visitEnd ( S_barline& elt )
2 {
3     // ... ..
4
5     // create the barLine
6     S_msrBarLine
7     barLine =
8         msrBarLine::create (
9             inputLineNumber,
10            fCurrentBarLineLocationKind,
11            fCurrentBarLineStyleKind,
12            fCurrentBarLineRepeatDirectionKind,
13            fCurrentBarLineEndingTypeKind,
14            fCurrentBarLineEndingNumber,
15            fCurrentBarLineTimes,
16            msrBarLineCategoryKind::kBarLineCategory_UNKNOWN_, // will be set afterwards
17            fCurrentBarLineHasSegnoKind,
18            fCurrentBarLineHasCodaKind,
19            fCurrentBarLineRepeatWingedKind);
20
21     // ... ..
22
23     // wait until its category is defined
24     // to append the barLine to the current segment
25
26     // handle the barLine according to: JMI
27     // http://www.musicxml.com/tutorial/the-midi-compatible-part/repeats/
28
29     Bool barLineHasBeenHandled = false;
30
31     switch (fCurrentBarLineLocationKind) {
32     case msrBarLineLocationKind::kBarLineLocationNone:
33         // should not occur
34         break;
35
36     case msrBarLineLocationKind::kBarLineLocationLeft:
37         if (
38             fCurrentBarLineEndingTypeKind
39             ==
40             msrBarLineEndingTypeKind::kBarLineEndingTypeStart
41         ) {
42             // ending start, don't know yet whether it's hooked or hookless
43             // -----
44             if (! fCurrentBarLineEndingNumber.size ()) {

```

```

45     musicxmlWarning (
46         gServiceRunData->getInputSourceName (),
47         inputLineNumber,
48         "mandatory ending number is missing, assuming \"1\\\"");
49
50     fCurrentBarLineEndingNumber = "1";
51 }
52
53 // don't know yet whether repeat ending start barLine is hooked or hookless
54 // remember it in fCurrentRepeatEndingStartBarLine,
55 fCurrentRepeatEndingStartBarLine = barLine;
56
57 // handle the repeat ending start
58 handleRepeatEndingStart (barLine);
59
60 barLineHasBeenHandled = true;
61 }
62
63 else if (
64     fCurrentBarLineRepeatDirectionKind
65     ==
66     msrBarLineRepeatDirectionKind::kBarLineRepeatDirectionForward
67 ) {
68     // repeat start
69     // -----
70     // set the barLine category
71     barLine->
72         setBarLineCategory (
73             msrBarLineCategoryKind::kBarLineCategoryRepeatStart);
74
75     // handle the repeat start
76     handleRepeatStart (barLine);
77
78     barLineHasBeenHandled = true;
79 }
80 break;
81
82 case msrBarLineLocationKind::kBarLineLocationMiddle:
83     // JMI ???
84     break;
85
86 case msrBarLineLocationKind::kBarLineLocationRight:
87     {
88         if (
89             fCurrentBarLineEndingTypeKind == msrBarLineEndingTypeKind::
kBarLineEndingTypeStop
90             &&
91             fCurrentBarLineEndingNumber.size () != 0
92         ) {
93             // hooked ending end
94             // -----
95             // set current barLine ending start category
96             fCurrentRepeatEndingStartBarLine->
97                 setBarLineCategory (
98                     msrBarLineCategoryKind::kBarLineCategoryHookedEndingStart);
99
100             // set this barLine's category
101             barLine->
102                 setBarLineCategory (
103                     msrBarLineCategoryKind::kBarLineCategoryHookedEndingEnd);
104
105             // handle the repeat hooked ending end
106             handleRepeatHookedEndingEnd (barLine);
107
108             barLineHasBeenHandled = true;
109         }
110

```



```

111     else if (
112         fCurrentBarLineRepeatDirectionKind
113         ==
114         msrBarLineRepeatDirectionKind::kBarLineRepeatDirectionBackward
115     ) {
116         // repeat end
117         // -----
118
119         // set this barLine's category
120         barLine->
121             setBarLineCategory (
122                 msrBarLineCategoryKind::kBarLineCategoryRepeatEnd);
123
124         // handle the repeat end
125         handleRepeatEnd (barLine);
126
127         barLineHasBeenHandled = true;
128     }
129
130     else if (
131         fCurrentBarLineEndingTypeKind == msrBarLineEndingTypeKind::
132         kBarLineEndingTypeDiscontinue
133         &&
134         fCurrentBarLineEndingNumber.size () != 0
135     ) {
136         // hookless ending end
137         // -----
138         // set current barLine ending start category
139         fCurrentRepeatEndingStartBarLine->
140             setBarLineCategory (
141                 msrBarLineCategoryKind::kBarLineCategoryHooklessEndingStart);
142
143         // set this barLine's category
144         barLine->
145             setBarLineCategory (
146                 msrBarLineCategoryKind::kBarLineCategoryHooklessEndingEnd);
147
148         // handle the repeat hookless ending end
149         handleRepeatHooklessEndingEnd (barLine);
150
151         barLineHasBeenHandled = true;
152     }
153
154     // forget about current repeat ending start barLine
155     fCurrentRepeatEndingStartBarLine = nullptr;
156 }
157 // switch
158
159 // set the barLine category to stand alone if not yet handled
160 if (! barLineHasBeenHandled) {
161     switch (fCurrentBarLineStyleKind) {
162     case msrBarLineStyleKind::kBarLineStyleRegular:
163     case msrBarLineStyleKind::kBarLineStyleDotted:
164     case msrBarLineStyleKind::kBarLineStyleDashed:
165     case msrBarLineStyleKind::kBarLineStyleHeavy:
166     case msrBarLineStyleKind::kBarLineStyleLightLight:
167     case msrBarLineStyleKind::kBarLineStyleLightHeavy:
168     case msrBarLineStyleKind::kBarLineStyleHeavyLight:
169     case msrBarLineStyleKind::kBarLineStyleHeavyHeavy:
170     case msrBarLineStyleKind::kBarLineStyleTick:
171     case msrBarLineStyleKind::kBarLineStyleShort:
172         barLine->
173             setBarLineCategory (
174                 msrBarLineCategoryKind::kBarLineCategoryStandalone);
175
176         // append the bar line to the current part

```

```

177      // ... ..
178
179      fCurrentPart->
180          appendBarLineToPart (barLine);
181
182      barLineHasBeenHandled = true;
183      break;
184
185  case msrBarLineStyleKind::kBarLineStyleNone:
186      std::stringstream ss;
187
188      ss <<
189          "barLine " <<
190          barLine->asString () <<
191          " has no barLine style";
192
193      musicxmlWarning (
194          gServiceRunData->getInputSourceName (),
195          inputLineNumber,
196          // __FILE__, __LINE__,
197          ss.str ());
198      break;
199  } // switch
200 }
201
202 // has this barLine been handled?
203 if (! barLineHasBeenHandled) {
204     std::stringstream ss;
205
206     ss << std::left <<
207         "cannot handle a barLine containing: " <<
208         barLine->asString ();
209
210     msrInternalWarning (
211         gServiceRunData->getInputSourceName (),
212         inputLineNumber,
213         ss.str ());
214 }
215
216 fOnGoingBarLine = false;
217 }

```

48.2 Translating repeats from MXSR to MSR

This is done in [src/passes/mxsr2msr/](#).

48.3 Translating repeats from MSR to MSR

This is done in [src/passes/msr2msr/](#).

48.4 Translating repeats from MSR to LPSR

This is done in [src/passes/msr2lpsr/](#).

48.5 Translating repeats from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

Chapter 49

Voices handling

Voices are presented at section [20.28](#) [Voices], page [204](#).

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 [20.26](#) [Staves], page [203](#).

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 [20.25](#) [Parts], page [202](#).

51.1 Parts browsing

Method `msrPart::browseData ()` defined in `src/formats/msr/msrParts.h/.cpp` is peculiar in that it imposes a *partial order* on the part staves browsing:

```
1 void msrPart::browseData (basevisitor* v)
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     if (gMsrOahGroup->getTraceMsrvVisitors ()) {
5         std::stringstream ss;
6
7         ss <<
8             "% ==> msrPart::browseData ()";
9
10        gWaeHandler->waeTraceWithoutInputLocation (
11            __FILE__, __LINE__,
12            ss.str ());
13    }
14 #endif // MF_TRACE_IS_ENABLED
15
16 #ifdef MF_TRACE_IS_ENABLED // JMI
17     if (gMsrOahGroup->getTraceMsrvVisitors ()) { // JMI TEMP
18         gLog <<
19             "+++++++ fPartAllStavesList.size(): " <<
20             fPartAllStavesList.size () <<
21             std::endl;
22
23         if (fPartAllStavesList.size ()) {
24             for (S_msrStaff staff : fPartAllStavesList) {
25                 gLog <<
26                     std::endl <<
27                     "+++++++ staff: +++++++" <<
28                     " \"\" << staff->getStaffName () << \"\" <<
29                     std::endl;
30             } // for
31         }
32
33         gLog <<
34             "+++++++ fPartNonHarmoniesNorFiguredBassStavesList.size(): " <<
35             fPartNonHarmoniesNorFiguredBassStavesList.size () <<
36             std::endl;
37
38         if (fPartNonHarmoniesNorFiguredBassStavesList.size ()) {
39             for (S_msrStaff staff : fPartNonHarmoniesNorFiguredBassStavesList) {
```

```

40     gLog <<
41         std::endl <<
42         "++++++ staff: ++++++" <<
43         " \\" << staff->getStaffName () << "\\" <<
44         std::endl;
45     } // for
46 }
47 }
48 #endif // MF_TRACE_IS_ENABLED
49
50 /* don't enforce any order here, leave it to the client thru sorting JMI */
51
52 // browse the part harmonies staff if any right now, JMI
53 // to place it before the corresponding part
54 if (fPartHarmoniesStaff) {
55     msrBrowser<msrStaff> browser (v);
56     browser.browse (*fPartHarmoniesStaff);
57 }
58
59 // browse all non harmonies and non figured bass staves
60 for (S_msrStaff staff : fPartNonHarmoniesNorFiguredBassStavesList) {
61     // browse the staff
62     msrBrowser<msrStaff> browser (v);
63     browser.browse (*staff);
64 } // for
65
66 // browse the part figured bass staff if any only now, JMI
67 // to place it after the corresponding part
68 if (fPartFiguredBassStaff) {
69     msrBrowser<msrStaff> browser (v);
70     browser.browse (*fPartFiguredBassStaff);
71 }
72
73 // // browse all the part staves JMI
74 // for (S_msrStaff staff : fPartAllStavesList) {
75 //     if (staff != fPartHarmoniesStaff && staff != fPartFiguredBassStaff) {
76 //         // browse the staff
77 //         msrBrowser<msrStaff> browser (v);
78 //         browser.browse (*staff);
79 //     }
80 // } // for
81 }

```

51.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

51.3 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

51.4 Translating from MSR to MSR ([src/passes/msr2msr/](#))

51.5 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

51.6 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 52

Part groups handling

MusicFormats part groups are presented at chapter [20.24](#) [Part groups], page [202](#).

In MusicXML, part groups can overlap, even though no one seems ever to have needed that. That seems to be more a feature in the Finale handling of MusicXMLexport that a true musical need.

MSR does not support overlapping part group. Handling part groups is done in `src/passes/mxsr2msr/mxsr2msrTran` where overlapping groups are identified and rejected:

```
1 jacquesmenu@macmini > xml2ly partgroups/OverlappingPartGroups.xml
2 ### MusicXML ERROR ### partgroups/OverlappingPartGroups.xml:169:
3 There are overlapping part groups, namely:
4   '2' ==> PartGroup_6 ('2', partGroupName "1
5 2"), lines 164..169
6 and
7   '1' ==> PartGroup_2 ('1', partGroupName ""), lines 76..170
8
9 Please contact the maintainers of MusicFormats (see option '-c, -contact'):
10 either you found a bug in the xml2ly converter,
11 or this MusicXML data is the first-ever real-world case
12 of a score exhibiting overlapping part groups
13 std::exception caught: mfException:
14 There are overlapping part groups, namely:
15   '2' ==> PartGroup_6 ('2', partGroupName "1
16 2"), lines 164..169
17 and
18   '1' ==> PartGroup_2 ('1', partGroupName ""), lines 76..170
19
20 Please contact the maintainers of MusicFormats (see option '-c, -contact'):
21 either you found a bug in the xml2ly converter,
22 or this MusicXML data is the first-ever real-world case
23 of a score exhibiting overlapping part groups
24
25 Error message(s) were issued for input line 169
26 ### xml2ly gIndenter final value: 1 ###
27 ### Conversion from MusicXML to LilyPond failed ###
```

class mxsrPartGroup contains:

```
1 struct mxsrPartGroup : public smartable
2 {
3   /*
4    positions represent the order in which the parts appear in <part-list />
5   */
6
7   // ... ..
8
9   private:
10
```



```

11 // private fields
12 // -----
13
14 int                fStartInputLineNumber;
15 int                fStopInputLineNumber;
16
17 int                fPartGroupNumber; // may be reused later
18
19 S_msrPartGroup     fPartGroup;
20
21 int                fStartPosition;
22 int                fStopPosition;
23 };

```

Part groups numbers number re-used and they can be nested, so there is an implicit outer-most part group at the top of their hierarchy, attached to the class `msrScore`:

```

1 class EXP mxsr2msrSkeletonBuilder :
2 // ... ..
3
4 // an implicit outer-most part group has to be created to contain everything,
5 // since there can be parts out of any explicit part group in MusicXML
6 S_mxsrPartGroup    fImplicitOuterMxsrPartGroup;
7 S_msrPartGroup     fImplicitOuterPartGroup;
8
9 void                createImplicitOuterPartGroup ();
10
11 // part groups numbers can be re-used, they're no identifier
12 // we use a map to access them by part group number
13 int                fPartGroupsCounter;
14 std::vector<S_mxsrPartGroup>
15                    fPartGroupDescsVector;
16 std::map<int, S_mxsrPartGroup>
17                    fAllMxsrPartGroupsMap;
18 std::map<int, S_mxsrPartGroup>
19                    fStartedMxsrPartGroupsMap;
20
21 // ... ..

```

Chapter 53

Scores handling

Scores are presented at section [20.23](#) [Scores], page [202](#).

53.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

53.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

53.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

53.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

53.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 54

Books handling

Books are presented at section [20.22](#) [Books], page [201](#).

54.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

54.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

54.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

54.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

54.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 55

Ornaments handling

Ornaments are presented at section ?? [Ornaments], page ??.

55.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

55.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

55.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

55.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

55.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 56

Ties handling

Ties are presented at section ?? [Ties], page ??.

56.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

56.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

56.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

56.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

56.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 57

Dynamics handling

Dynamics are presented at section ?? [Dynamics], page ??.

57.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

57.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

57.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

57.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

57.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 58

Beams handling

Beams are presented at section ?? [Beams], page ??.

58.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

58.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

58.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

58.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

58.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 59

Slurs handling

Slurs are presented at section ?? [Slurs], page ??.

59.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

59.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

59.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

59.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

59.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 60

Grace notes groups handling

Grace notes groups are presented at section [20.38](#) [Grace notes groups], page [213](#).

60.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

60.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

60.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

60.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

60.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 61

Chords handling

Chords are presented at section [20.39](#) [Chords], page [213](#).

61.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

61.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

61.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

61.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

61.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 62

Tuples handling

Tuples are presented at section [20.40](#) [Tuples], page [213](#).

62.1 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

62.2 Translating from MXSR to MSR ([src/passes/mxsr2msr/](#))

62.3 Translating from MSR to MSR ([src/passes/msr2msr/](#))

62.4 Translating from MSR to LPSR ([src/passes/msr2lpsr/](#))

62.5 Translating from LPSR to LilyPond ([src/passes/lpsr2lilypond/](#))

Chapter 63

Harmonies handling

Harmonies are presented at section [20.42](#) [Harmonies], page [215](#).

The useful options here are:

- option `-trace-harmonies`, `-tharms`
- option `-display-msr1-skeleton`, `-dmsrskel`
- option `-display-msr1`, `-dmsr1`
- option `-display-msr1-full`, `-dmsr1full`
- option `-display-msr2`, `-dmsr2`
- option `-display-msr2-full`, `-dmsr2ull`
- option `-display-lpsr`, `-dlpsr`
- option `-display-lpsr-full`, `-dlpsrull`

63.1 Harmonies in MusicXML

In the MusicXML view of music scores, harmonies are simply *drawn* at the current music position, so to say.

The `<harmony/>` markup occurs at the measure level in MusicXML, as notes do, but they apply at the part level, even though this is not explicitly stated, since there is no staff nor voice information associated with them:

```
1 <!--
2 The harmony elements are based on Humdrum's **harm
3 encoding, extended to support chord symbols in popular
4 music as well as functional harmony analysis in classical
5 music.
6
7 If there are alternate harmonies possible, this can be
8 specified using multiple harmony elements differentiated
9 by type. Explicit harmonies have all note present in the
10 music; implied have some notes missing but implied;
11 alternate represents alternate analyses.
12
13 The harmony object may be used for analysis or for
14 chord symbols. The print-object attribute controls
15 whether or not anything is printed due to the harmony
16 element. The print-frame attribute controls printing
17 of a frame or fretboard diagram. The print-style entity
```

```

18 sets the default for the harmony, but individual elements
19 can override this with their own print-style values.
20
21 A harmony element can contain many stacked chords (e.g.
22 V of II). A sequence of harmony-chord entities is used
23 for this type of secondary function, where V of II would
24 be represented by a harmony-chord with a V function
25 followed by a harmony-chord with a II function.
26 -->
27 <!ENTITY % harmony-chord "((root | function), kind,
28 inversion?, bass?, degree*)">
29
30 <!ELEMENT harmony ((%harmony-chord;)+, frame?,
31 offset?, %editorial;, staff?)>
32 <!ATTLIST harmony
33 type (explicit | implied | alternate) #IMPLIED
34 %print-object;
35 print-frame %yes-no; #IMPLIED
36 %print-style;
37 %placement;
38 %optional-unique-id;
39 >

```

The harmony `<function/>`, defined in MusicXML, is a string. It is stored in the class `msrHarmony`, but nothing more. No example of that is present in the MusicXML sample files this author could find, actually.

1 The `<function>` element represents classical functional harmony with an indication like I, II, III rather than C, D, E. It represents the Roman numeral part of a functional harmony rather than the `complete function` itself. It has been deprecated as of MusicXML 4.0 in favor of the `<numeral>` element.

The MusicXML 4.0 `<numeral/>` markup is described at <https://www.w3.org/2021/06/musicxml40/musicxml-reference/elements/numeral/>. Examples are:

```

1 <harmony default-y="-80">
2   <numeral>
3     <numeral-root text="IV">4</numeral-root>
4   </numeral>
5   <inversion>1</inversion>
6 </harmony>
7
8 <harmony default-y="20">
9   <numeral>
10    <numeral-root text="III">3</numeral-root>
11    <numeral-alter location="left">-1</numeral-alter>
12  </numeral>
13  <kind align="center" text="">major</kind>
14 </harmony>

```

63.2 Harmonies in MSR

Harmonies need special treatment since we need to determine their position in a harmony `msrVoice`, taking offsets if any into account.

Harmonies are handled this way, using denormalization:

- harmonies are attached to `msrNote` instances,
- they are placed in specific `msrVoice` instances, whose field `fVoiceKind::` contains `msrVoiceKind::kVoiceKindH`
- there are also attached to `msrNote`, `msrChord` and `msrTuplet` (denormalization);

In class `msrNote`, there is:

```

1  // harmonies
2  void                appendHarmonyToNote (
3                      const S_msrHarmony& harmony);
4
5  const std::list<S_msrHarmony>&
6      getNoteHarmoniesList () const
7      { return fNoteHarmoniesList; }
8
9  // ... ..
10
11 // harmonies
12 // -----
13
14 std::list<S_msrHarmony>
15     fNoteHarmoniesList;
16
17 // ... ..

```

63.3 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`:

Since a voice belongs to a `msrStaff` in `MSR`, there are `msrStave` instance to contain them,

```

1  public:
2
3  // constants
4  // -----
5
6  static const int K_PART_HARMONIES_STAFF_NUMBER;
7  static const int K_PART_HARMONIES_VOICE_NUMBER;

```

In class `msrStaff`, there is:

```

1  void                registerHarmoniesVoiceByItsNumber (
2                      int                inputLineNumber,
3                      const S_msrVoice& voice);

```

Class `msrPart` also contains:

```

1  // harmonies
2
3  S_msrVoice          createPartHarmoniesVoice (
4                      int                inputLineNumber,
5                      const std::string& currentMeasureNumber);
6
7  void                appendHarmonyToPart (
8                      const S_msrVoice&  harmonySupplierVoice,
9                      const S_msrHarmony& harmony);
10
11 void                appendHarmonyToPartClone (
12                    const S_msrVoice&  harmonySupplierVoice,
13                    const S_msrHarmony& harmony);

```

```

1  // harmonies
2
3  S_msrStaff          fPartHarmoniesStaff;
4  S_msrVoice          fPartHarmoniesVoice;

```

63.4 Harmonies staves creation

This is done in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.cpp.h/.cpp`:

```

1 S_msrVoice mxsr2msrSkeletonBuilder::createPartHarmoniesVoiceIfNotYetDone (
2     int         inputLineNumber,
3     const S_msrPart& part)
4 {
5     // is the harmonies voice already present in part?
6     S_msrVoice
7     partHarmoniesVoice =
8     part->
9     getPartHarmoniesVoice ();
10
11     if (! partHarmoniesVoice) {
12         // create the harmonies voice and append it to the part
13         partHarmoniesVoice =
14         part->
15         createPartHarmoniesVoice (
16             inputLineNumber,
17             fCurrentMeasureNumber);
18     }
19
20     return partHarmoniesVoice;
21 }

```

Method `msrPartcreatePartHarmoniesVoice` creates the part harmonies staff and the part harmonies voice, and then registers the latter in the former:

```

1 S_msrVoice msrPart::createPartHarmoniesVoice (
2     int         inputLineNumber,
3     std::string currentMeasureNumber)
4 {
5     // ... ..
6
7     // create the part harmonies staff
8     int partHarmoniesStaffNumber =
9     msrPart::K_PART_HARMONIES_STAFF_NUMBER;
10
11     // ... ..
12
13     fPartHarmoniesStaff =
14     addHarmoniesStaffToPart (
15         inputLineNumber);
16
17     // ... ..
18
19     // create the part harmonies voice
20     int partHarmoniesVoiceNumber =
21     msrPart::K_PART_HARMONIES_VOICE_NUMBER;
22
23     // ... ..
24
25     fPartHarmoniesVoice =
26     msrVoice::create (
27         inputLineNumber,
28         msrVoiceKind::kVoiceKindHarmonies,
29         partHarmoniesVoiceNumber,
30         msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes,
31         fPartHarmoniesStaff);
32
33     // register the part harmonies voice in part harmonies staff
34     fPartHarmoniesStaff->
35     registerVoiceInStaff (
36         inputLineNumber,
37         fPartHarmoniesVoice);

```

```

38
39 // ... ..
40
41 return fPartHarmoniesVoice;
42 }

```

63.5 Translating harmonies from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

The MSR score skeleton created in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.h/.cpp` contains the part groups, parts, staves and voices, as well as the number of measures. The voices do not contain any music elements yet.

A harmony belongs to a `<part/>` in MusicXML, but we sometimes need to have it attached to a note. When visiting an `S_harmony` element, field `mxsr2msrSkeletonBuilder::fThereAreHarmoniesToBeAttachedToCurrentNote` is used to account for that:

```

1 void mxsr2msrSkeletonBuilder::visitStart ( S_harmony& elt )
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
5             std::stringstream ss;
6
7             ss <<
8                 "--> Start visiting S_harmony" <<
9                 ", harmoniesVoicesCounter = " << fHarmoniesVoicesCounter <<
10                ", line " << elt->getInputStartLineNumber ();
11
12             gWaeHandler->waeTraceWithoutInputLocation (
13                 __FILE__, __LINE__,
14                 ss.str ());
15         }
16     #endif // MF_TRACE_IS_ENABLED
17
18     /* JMI ???
19        several harmonies can be attached to a given note,
20        leading to as many harmonies voices in the current part
21    */
22
23     // take harmonies voice into account
24     ++fHarmoniesVoicesCounter; // UNUSED JMI
25
26     fThereAreHarmoniesToBeAttachedToCurrentNote = true;
27 }

```

Upon the second visit of class `msrNote`, the part harmonies voice is created if harmonies are not to be ignored due to option `-ignore-musicxml-harmonies`, `-oharms` and it has not been created yet:

```

1 void mxsr2msrSkeletonBuilder::visitEnd ( S_note& elt )
2 {
3     // ... ..
4
5     // are there harmonies attached to the current note?
6     if (fThereAreHarmoniesToBeAttachedToCurrentNote) {
7         if (gGlobalMxsr2msrOahGroup->getIgnoreHarmonies ()) {
8             #ifdef MF_TRACE_IS_ENABLED
9                 if (gTraceOahGroup->getTraceHarmonies ()) {
10                     gLog <<
11                         "Ignoring the harmonies" <<
12                         ", line " <<
13                         inputLineNumber <<

```



```

14         std::endl;
15
16         gWaeHandler->waeTraceWithoutInputLocation (
17             __FILE__, __LINE__,
18             ss.str ());
19     }
20 #endif // MF_TRACE_IS_ENABLED
21 }
22 else {
23     // create the part harmonies voice if not yet done
24     S_msrVoice
25     partHarmoniesVoice =
26         createPartHarmoniesVoiceIfNotYetDone (
27             inputLineNumber,
28             fCurrentPart);
29 }
30
31 fThereAreHarmoniesToBeAttachedToCurrentNote = false;
32 }
33
34 // ... ..
35 }

```

Creating the part harmonies voice is delegated to the part:

```

1 S_msrVoice mxsr2msrSkeletonBuilder::createPartHarmoniesVoiceIfNotYetDone (
2     int inputLineNumber,
3     const S_msrPart& part)
4 {
5     // is the harmonies voice already present in part?
6     S_msrVoice
7     partHarmoniesVoice =
8     part->
9     getPartHarmoniesVoice ();
10
11     if (! partHarmoniesVoice) {
12         // create the harmonies voice and append it to the part
13         partHarmoniesVoice =
14         part->
15         createPartHarmoniesVoice (
16             inputLineNumber,
17             fCurrentMeasureNumber);
18     }
19
20     return partHarmoniesVoice;
21 }

```

63.5.1 First S_harmony visit

The first visit of S_harmony initializes the fields storing values to be gathered visiting subelements:

```

1 void mxsr2msrTranslator::visitStart ( S_harmony& elt )
2 {
3     int inputLineNumber =
4     elt->getInputStartLineNumber ();
5
6     #ifdef MF_TRACE_IS_ENABLED
7     if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
8         std::stringstream ss;
9
10        ss <<
11        "--> Start visiting S_harmony" <<
12        ", line " << inputLineNumber;
13    }

```

```

14     gWaeHandler->waeTraceWithoutInputLocation (
15         __FILE__, __LINE__,
16         ss.str ());
17 }
18 #endif // MF_TRACE_IS_ENABLED
19
20 ++fHarmoniesVoicesCounter;
21
22 fCurrentHarmonyInputLineNumber      = inputLineNumber;
23
24 fCurrentHarmonyRootDiatonicPitchKind = msrDiatonicPitchKind::kDiatonicPitch_UNKNOWN_;
25 fCurrentHarmonyRootAlterationKind    = msrAlterationKind::kAlterationNatural;
26 fCurrentHarmonyKind                  = msrHarmonyKind::kHarmony_UNKNOWN_;
27 fCurrentHarmonyKindText               = "";
28 fCurrentHarmonyInversion              = K_HARMONY_INVERSION_NONE;
29 fCurrentHarmonyBassDiatonicPitchKind = msrDiatonicPitchKind::kDiatonicPitch_UNKNOWN_;
30 fCurrentHarmonyBassAlterationKind     = msrAlterationKind::kAlterationNatural;
31 fCurrentHarmonyDegreeValue            = -1;
32 fCurrentHarmonyDegreeAlterationKind  = msrAlterationKind::kAlterationNatural;
33
34 fCurrentHarmonyWholeNotesOffset = mfRational (0, 1);
35
36 fOnGoingHarmony = true;
37 }

```

63.5.2 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::create ()`:

```

1 void mxsr2msrTranslator::visitEnd ( S_harmony& elt )
2 {
3     // ... ..
4
5     if (gGlobalMxsr2msrOahGroup->getIgnoreHarmonies ()) {
6         #ifdef MF_TRACE_IS_ENABLED
7             if (gTraceOahGroup->getTraceHarmonies ()) {
8                 gLog <<
9                     "Ignoring harmony" <<
10                     ", line " <<
11                     inputLineNumber;
12
13                 gWaeHandler->waeTraceWithoutInputLocation (
14                     __FILE__, __LINE__,
15                     ss.str ());
16             }
17         #endif // MF_TRACE_IS_ENABLED
18     }
19     else {
20         // create the harmony
21         #ifdef MF_TRACE_IS_ENABLED
22             if (gTraceOahGroup->getTraceHarmoniesDetails ()) {
23                 gLog <<
24                     "Creating a harmony" <<
25                     ", line " << inputLineNumber << ":" <<
26                     std::endl;
27
28                 // ... ..
29             }
30         #endif // MF_TRACE_IS_ENABLED
31
32         S_msrHarmony
33         harmony =

```

```

34     msrHarmony::create (
35         fCurrentHarmonyInputLineNumber,
36         // no harmoniesUpLinkToVoice yet
37
38         fCurrentHarmonyRootQuarterTonesPitchKind,
39
40         fCurrentHarmonyKind,
41         fCurrentHarmonyKindText,
42
43         fCurrentHarmonyInversion,
44
45         fCurrentHarmonyBassQuarterTonesPitchKind,
46
47         mfRational (1, 1),           // harmonySoundingWholeNotes,
48                                     // will be set upon next note handling
49         mfRational (1, 1),           // harmonyDisplayWholeNotes,
50                                     // will be set upon next note handling
51         fCurrentHarmoniesStaffNumber,
52         msrTupletFactor (1, 1),      // will be set upon next note handling
53         fCurrentHarmonyWholeNotesOffset);
54
55     // append pending harmony degrees if any to the harmony
56     if (! fCurrentHarmonyDegreesList.size ()) {
57 #ifdef MF_TRACE_IS_ENABLED
58         if (gTrace0ahGroup->getTraceHarmoniesDetails ()) {
59             musicxmlWarning (
60                 gServiceRunData->getInputSourceName (),
61                 inputLineNumber,
62                 "harmony has no degrees contents");
63         }
64 #endif // MF_TRACE_IS_ENABLED
65     }
66
67     else {
68         // handle harmony degrees if any
69         while (fCurrentHarmonyDegreesList.size ()) {
70             S_msrHarmonyDegree
71             harmonyDegree =
72                 fCurrentHarmonyDegreesList.front ();
73
74             // ... ..
75
76             // append it to harmony's degrees list
77             harmony->
78                 appendHarmonyDegreeToHarmony (
79                     harmonyDegree);
80
81             // remove it from the list
82             fCurrentHarmonyDegreesList.pop_front ();
83         } // while
84     }
85
86     // attach the current frame if any to the harmony
87     if (fCurrentFrame) {
88         harmony->setHarmonyFrame (fCurrentFrame);
89     }
90
91     // append the harmony to the pending harmonies list
92     fPendingHarmoniesList.push_back (harmony);
93 }
94
95 fOnGoingHarmony = false;
96 }

```

63.5.3 Attaching msrHarmony instances to notes

msrHarmony

The contents of `mxsr2msrTranslator::fPendingHarmoniesList` is attached to the class `msrNote` instance in method `mxsr2msrTranslator::populateNote ()`:

```

1 void mxsr2msrTranslator::populateNote (
2     int         inputLineNumber,
3     const S_msrNote& newNote)
4 {
5     // ... ..
6
7     // handle the pending harmonies if any
8     if (fPendingHarmoniesList.size ()) {
9         // get voice to insert harmonies into
10        S_msrVoice
11        voiceToInsertHarmoniesInto =
12            fCurrentPart->
13            getPartHarmoniesVoice ();
14
15        // ... ..
16
17        handlePendingHarmonies (
18            newNote,
19            voiceToInsertHarmoniesInto);
20
21        // reset harmony counter
22        fHarmoniesVoicesCounter = 0;
23    }
24 }
```

63.5.4 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::appendHarmonyToNote ()`:

```

1 void mxsr2msrTranslator::handlePendingHarmonies (
2     const S_msrNote& newNote,
3     const S_msrVoice& voiceToInsertInto)
4 {
5     // ... ..
6
7     mfRational
8     newNoteSoundingWholeNotes =
9         fCurrentNote->
10        getSoundingWholeNotes (),
11     newNoteDisplayWholeNotes =
12         fCurrentNote->
13        getNoteDisplayWholeNotes ();
14
15     while (fPendingHarmoniesList.size ()) { // recompute at each iteration
16         S_msrHarmony
17         harmony =
18             fPendingHarmoniesList.front ();
19
20         /*
21         MusicXML harmonies don't have a duration,
22         and MSR could follow this line, but LilyPond needs one...
23         So:
24         - we register all harmonies with the duration of the next note
25         - they will be sorted by position in the measure in finalizeMeasure(),

```

```

26         at which time their duration may be shortened
27         so that the offsets values are enforced
28         and they don't overflow the measure
29     It is VITAL that harmonies measures be finalized
30     AFTER the corresponding measure in the regular voice,
31     since the current sounding whole notes of the latter is needed for that
32 */
33
34 // set the harmony's sounding whole notes
35 harmony->
36     setSoundingWholeNotes (
37         newNoteSoundingWholeNotes,
38         "mxsr2msrTranslator::handlePendingHarmonies()");
39
40 // set the harmony's display whole notes JMI useless???
41 harmony->
42     setHarmonyDisplayWholeNotes (
43         newNoteDisplayWholeNotes);
44
45 // set the harmony's tuplet factor
46 harmony->
47     setHarmonyTupletFactor (
48         msrTupletFactor (
49             fCurrentNoteActualNotes,
50             fCurrentNoteNormalNotes));
51
52 // attach the harmony to newNote
53 fCurrentNote->
54     appendHarmonyToNote (
55         harmony);
56
57 // get the part harmonies voice
58 S_msrVoice
59     partHarmoniesVoice =
60         fCurrentPart->
61             getPartHarmoniesVoice ();
62
63 #ifdef MF_SANITY_CHECKS_ARE_ENABLED
64     // sanity check
65     mfAssert (
66         __FILE__, __LINE__,
67         partHarmoniesVoice != nullptr,
68         "partHarmoniesVoice is null");
69 #endif // MF_SANITY_CHECKS_ARE_ENABLED
70
71 // set the harmony's voice upLink
72 // only now that we know which harmonies voice will contain it
73 harmony->
74     setHarmoniesUpLinkToVoice (
75         partHarmoniesVoice);
76
77 /* JMI v0.9.67
78 // append the harmony to the part harmonies voice
79 partHarmoniesVoice->
80     appendHarmonyToVoice (
81         harmony);
82 */
83 // don't append the harmony to the part harmonies voice // BLARK
84 // before the note itself has been appended to the voice
85
86 // remove the harmony from the list
87 fPendingHarmoniesList.pop_front ();
88 } // while
89 }

```

63.5.5 First S_harmony visit

msrHarmony

Method `msrNote::appendHarmonyToNote ()` is where the harmony's note uplink is set:

```

1 void msrNote::appendHarmonyToNote (const S_msrHarmony& harmony)
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     if (gTrace0ahGroup->getTraceHarmonies ()) {
5         gLog <<
6             "Appending harmony " <<
7             harmony->asString () <<
8             " to the harmonies list of " <<
9             asString () <<
10            ", line " << fInputStartLineNumber;
11
12        gWaeHandler->waeTraceWithoutInputLocation (
13            __FILE__, __LINE__,
14            ss.str ());
15    }
16 #endif // MF_TRACE_IS_ENABLED
17
18    // update the harmony whole notes if it belongs to a tuplet ??? utf8.xml JMI
19
20    fNoteHarmoniesList.push_back (harmony);
21
22    // register this note as the harmony note upLink
23    harmony->
24        setHarmonyUpLinkToNote (this);
25 }

```

When a harmony is attached to a note that is a chord member, we have to attach it to the chord too, to facilitate setting its measure position when setting the chord's one.

```

1 void mxsr2msrTranslator::copyNoteHarmoniesToChord (
2     const S_msrNote& note,
3     const S_msrChord& chord)
4 {
5     // copy note's harmony if any from the first note to chord
6
7     const std::list<S_msrHarmony>&
8         noteHarmoniesList =
9         note->getNoteHarmoniesList ();
10
11    if (noteHarmoniesList.size ()) {
12        std::list<S_msrHarmony>::const_iterator i;
13        for (i = noteHarmoniesList.begin (); i != noteHarmoniesList.end (); ++i) {
14            S_msrHarmony harmony = (*i);
15
16 #ifdef MF_TRACE_IS_ENABLED
17             if (gTrace0ahGroup->getTraceHarmonies ()) {
18                 gLog <<
19                     "Copying harmony '" <<
20                     harmony->asString () <<
21                     "' from note " << note->asString () <<
22                     " to chord '" << chord <<
23                     '\'', <<
24                     std::endl;
25
26                 gWaeHandler->waeTraceWithoutInputLocation (
27                     __FILE__, __LINE__,
28                     ss.str ());
29             }
30 #endif // MF_TRACE_IS_ENABLED
31

```

```

32     chord->
33         appendHarmonyToChord (harmony);
34
35     } // for
36 }
37 }

```

63.5.6 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 (const S_msrNote& note)
2 {
3     // ... ..
4
5     // are there harmonies attached to this note? // BLARK
6     const std::list<S_msrHarmony>&
7         noteHarmoniesList =
8         note->
9         getNoteHarmoniesList ();
10
11     if (noteHarmoniesList.size ()) {
12         // get the current part's harmonies voice
13         S_msrVoice
14         partHarmoniesVoice =
15         part->
16         getPartHarmoniesVoice ();
17
18         for (S_msrHarmony harmony : noteHarmoniesList) {
19             // append the harmony to the part harmonies voice
20             partHarmoniesVoice->
21             appendHarmonyToVoice (
22                 harmony);
23         } // for
24     }
25
26     // ... ..
27 };

```

63.6 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:

```

1 void msr2msrTranslator::visitStart (S_msrHarmony& elt)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gMsrOahGroup->getTraceMsrVisitors ()) {
5             std::stringstream ss;
6
7             ss <<
8             "--> Start visiting msrHarmony '" <<
9             elt->asString () <<
10            ", fOnGoingNonGraceNote: " << fOnGoingNonGraceNote <<

```

```

11     ", fOnGoingChord: " << fOnGoingChord <<
12     ", fOnGoingHarmoniesVoice: " << fOnGoingHarmoniesVoice <<
13     ", fOnGoingHarmony: " << fOnGoingHarmony <<
14     "', line " << elt->getInputStartLineNumber ();
15
16     gWaeHandler->waeTraceWithoutInputLocation (
17         __FILE__, __LINE__,
18         ss.str ());
19 }
20 #endif // MF_TRACE_IS_ENABLED
21
22 // create a harmony newborn clone
23 fCurrentHarmonyClone =
24     elt->
25         createHarmonyNewbornClone (
26             fCurrentVoiceClone);
27
28 if (fOnGoingNonGraceNote) {
29     // register the harmony in the current non-grace note clone
30     fCurrentNonGraceNoteClone->
31         appendHarmonyToNote (
32             fCurrentHarmonyClone);
33
34     // don't append the harmony to the part harmony,
35     // this has been done in pass2b // JMI ???
36 }
37
38 else if (fOnGoingChord) {
39     // register the harmony in the current chord clone
40     fCurrentChordClone->
41         appendHarmonyToChord (fCurrentHarmonyClone); // JMI
42 }
43
44 else if (fOnGoingHarmoniesVoice) {
45     /* JMI
46     // get the harmony whole notes offset
47     mfRational
48     harmonyWholeNotesOffset =
49         elt->getHarmonyWholeNotesOffset ();
50
51     // is harmonyWholeNotesOffset not equal to 0?
52     if (harmonyWholeNotesOffset.getNumerator () != 0) {
53         // create skip with duration harmonyWholeNotesOffset
54         S_msrNote
55         skip =
56             msrNote::createSkipNote (
57                 elt->getInputStartLineNumber (),
58                 "666", // JMI elt->getHarmoniesMeasureNumber (),
59                 elt->getHarmonyDisplayWholeNotes (), // would be 0/1 otherwise
60                 JMI
61                 elt->getHarmonyDisplayWholeNotes (),
62                 0, // JMI elt->getHarmonyDotsNumber (),
63                 fCurrentVoiceClone->getRegularVoiceStaffSequentialNumber (), // JMI
64                 fCurrentVoiceClone->getVoiceNumber ());
65
66         // append it to the current voice clone
67         // to 'push' the harmony aside
68         fCurrentVoiceClone->
69             appendNoteToVoice (skip);
70     }
71     */
72
73     // append the harmony to the current voice clone
74     fCurrentVoiceClone->
75         appendHarmonyToVoiceClone (
76             fCurrentHarmonyClone);
77 }

```



```

77
78 else {
79     std::stringstream ss;
80
81     ss <<
82     "harmony is out of context, cannot be handled: " <<
83     elt->asShortString ();
84
85     msrInternalError (
86         gServiceRunData->getInputSourceName (),
87         elt->getInputStartLineNumber (),
88         __FILE__, __LINE__,
89         ss.str ());
90 }
91
92 fOnGoingHarmony = true;
93 }

```

There are only fields updates upon the second visit:

```

1 void msr2msrTranslator::visitEnd (S_msrHarmony& elt)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gMsrOahGroup->getTraceMsrvVisitors ()) {
5             std::stringstream ss;
6
7             ss <<
8             "--> End visiting msrHarmony '" <<
9             elt->asString () <<
10            '\n' <<
11            ", line " << elt->getInputStartLineNumber ();
12
13            gWaeHandler->waeTraceWithoutInputLocation (
14                __FILE__, __LINE__,
15                ss.str ());
16        }
17    #endif // MF_TRACE_IS_ENABLED
18
19    fCurrentHarmonyClone = nullptr;
20    fOnGoingHarmony = false;
21 }

```

63.7 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: :

```

1 void msr2lpsrTranslator::visitStart (S_msrHarmony& elt)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gMsrOahGroup->getTraceMsrvVisitors ()) {
5             std::stringstream ss;
6
7             ss <<
8             "--> Start visiting msrHarmony '" <<
9             elt->asString () <<
10            ", onGoingNonGraceNote: " << fOnGoingNonGraceNote <<
11            ", onGoingChord: " << fOnGoingChord <<
12            ", onGoingHarmoniesVoice: " << fOnGoingHarmoniesVoice <<
13            ", onGoingHarmony: " << fOnGoingHarmony <<

```

```

14     "', line " << elt->getInputStartLineNumber ();
15
16     gWaeHandler->waeTraceWithoutInputLocation (
17         __FILE__, __LINE__,
18         ss.str ());
19 }
20 #endif // MF_TRACE_IS_ENABLED
21
22 // create a harmony newborn clone
23 fCurrentHarmonyClone =
24     elt->
25         createHarmonyNewbornClone (
26             fCurrentVoiceClone);
27
28 if (fOnGoingNonGraceNote) {
29     // register the harmony in the current non-grace note clone
30     fCurrentNonGraceNoteClone->
31         appendHarmonyToNote (
32             fCurrentHarmonyClone);
33
34     // don't append the harmony to the part harmony,
35     // this has been done in pass2b // JMI ???
36 }
37
38 else if (fOnGoingChord) {
39     // register the harmony in the current chord clone
40     fCurrentChordClone->
41         appendHarmonyToChord (fCurrentHarmonyClone); // JMI
42 }
43
44 else if (fOnGoingHarmoniesVoice) {
45     /* JMI
46     // get the harmony whole notes offset
47     mfRational
48         harmonyWholeNotesOffset =
49             elt->getHarmonyWholeNotesOffset ();
50
51     // is harmonyWholeNotesOffset not equal to 0?
52     if (harmonyWholeNotesOffset.getNumerator () != 0) {
53         // create skip with duration harmonyWholeNotesOffset
54         S_msrNote
55             skip =
56             msrNote::createSkipNote (
57                 elt->
58                     getInputStartLineNumber (),
59                     "666", // JMI elt->
60                         getHarmoniesMeasureNumber (),
61                         getHarmonyDisplayWholeNotes (), // would be 0/1 otherwise
62
63                 JMI
64                     elt->
65                         getHarmonyDisplayWholeNotes (),
66                         0, // JMI elt->
67                             getHarmonyDotsNumber (),
68                             fCurrentVoiceClone-> getRegularVoiceStaffSequentialNumber (), // JMI
69                             fCurrentVoiceClone-> getVoiceNumber ());
70
71         // append it to the current voice clone
72         // to 'push' the harmony aside
73         fCurrentVoiceClone->
74             appendNoteToVoice (skip);
75     }
76 */
77
78 // append the harmony to the current voice clone
79 fCurrentVoiceClone->
80     appendHarmonyToVoiceClone (
81         fCurrentHarmonyClone);
82 }
83
84 else {
85     std::stringstream ss;

```

```
80
81     ss <<
82     "harmony is out of context, cannot be handled: " <<
83     elt->asShortString ();
84
85     msrInternalError (
86         gServiceRunData->getInputSourceName (),
87         elt->getInputStartLineNumber (),
88         __FILE__, __LINE__,
89         ss.str ());
90 }
91
92 fOnGoingHarmony = true;
93 }
```

Here too, there are only fields updates upon the second visit of `S_msrHarmony` instances:

```

1 void msr2lpsrTranslator::visitEnd (S_msrHarmony& elt)
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     if (gMsrOahGroup->getTraceMsrVisitors ()) {
5         std::stringstream ss;
6
7         ss <<
8         "--> End visiting msrHarmony '" <<
9         elt->asString () <<
10        '\n' <<
11        ", line " << elt->getInputStartLineNumber ();
12
13        gWaeHandler->waeTraceWithoutInputLocation (
14            __FILE__, __LINE__,
15            ss.str ());
16    }
17 #endif // MF_TRACE_IS_ENABLED
18
19     fCurrentHarmonyClone = nullptr;
20     fOnGoingHarmony = false;
21 }

```

63.8 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:

```

1 void lpsr2lilypondTranslator::visitStart (S_msrHarmony& elt)
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     {
5         Bool
6         traceMsrVisitors =
7             gMsrOahGroup->
8             getTraceMsrVisitors (),
9         generateMsrVisitingInformation =
10            gGlobalLpsr2lilypondOahGroup->
11            getGenerateMsrVisitingInformation ();
12
13        if (traceMsrVisitors || generateMsrVisitingInformation) {
14            std::stringstream ss;
15
16            ss <<
17            "% --> Start visiting msrHarmony '" <<
18            elt->asString () <<
19            '\n' <<
20            ", fOnGoingNotesStack.size (): \"" <<
21            fOnGoingNotesStack.size () <<
22            ", fOnGoingChord: \"" <<
23            fOnGoingChord <<
24            ", fOnGoingHarmoniesVoice: \"" <<
25            fOnGoingHarmoniesVoice <<
26            ", line " << elt->getInputStartLineNumber () <<
27            std::endl;
28
29            if (traceMsrVisitors) {
30                gLog << ss.str ();
31            }
32        }
33    }
34 #endif
35 }

```

```

32
33     if (generateMsrVisitingInformation) {
34         fLilypondCodeStream << ss.str ();
35     }
36 }
37 }
38 #endif // MF_TRACE_IS_ENABLED
39
40 if (fOnGoingNotesStack.size () > 0) {
41     /* JMI
42 #ifdef MF_TRACE_IS_ENABLED
43     if (gTraceOahGroup->getTraceHarmonies ()) {
44         fLilypondCodeStream <<
45             "%{ fOnGoingNotesStack.size () S_msrHarmony JMI " <<
46             elt->asString () <<
47             " %}";
48
49         gWaeHandler->waeTraceWithoutInputLocation (
50             __FILE__, __LINE__,
51             ss.str ());
52     }
53 #endif // MF_TRACE_IS_ENABLED
54 */
55 }
56
57 else if (fOnGoingChord) { // JMI
58 }
59
60 else if (fOnGoingHarmoniesVoice) {
61     // actual LilyPond code generation
62     fLilypondCodeStream <<
63         harmonyAsLilypondString (elt) <<
64         ' ';
65
66     // generate the input line number as comment if relevant
67     if (
68         gGlobalLpsr2lilypondOahGroup->getInputStartLineNumbers ()
69         ||
70         gGlobalLpsr2lilypondOahGroup->getGenerateMeasurePositions ()
71     ) {
72         generateInputLineNumberAndOrMeasurePositionAsAComment (
73             elt);
74     }
75 }
76 }

```

Chapter 64

Figured bass handling

Figured bass elements are presented at section ?? [Figured bass elements], page ??.

The useful options here are:

- option `-trace-figured-bass`, `-tfigbass`
- option `-display-msr1-skeleton`, `-dmsrskel`
- option `-display-msr1`, `-dmsr1`
- option `-display-msr1-full`, `-dmsr1full`
- option `-display-msr2`, `-dmsr2`
- option `-display-msr2-full`, `-dmsr2full`
- option `-display-lpsr`, `-dlpsr`
- option `-display-lpsr-full`, `-dlpsrfull`

64.1 Figured bass in MusicXML

In the MusicXML view of figured bass, figured bass elements are simply *drawn* at the current music position, so to say.

```
1 <!--
2   Figured bass elements take their position from the first
3   regular note (not a grace note or chord note) that follows
4   in score order. The optional duration element is used to
5   indicate changes of figures under a note.
6
7   Figures are ordered from top to bottom. A figure-number is
8   a number. Values for prefix and suffix include plus and
9   the accidental values sharp, flat, natural, double-sharp,
10  flat-flat, and sharp-sharp. Suffixes include both symbols
11  that come after the figure number and those that overstrike
12  the figure number. The suffix values slash, back-slash, and
13  vertical are used for slashed numbers indicating chromatic
14  alteration. The orientation and display of the slash usually
15  depends on the figure number. The prefix and suffix elements
16  may contain additional values for symbols specific to
17  particular figured bass styles. The value of parentheses
18  is "no" if not present.
19  -->
20 <!ELEMENT figured-bass (figure+, duration?, %editorial;)>
```

```

21 <!ATTLIST figured-bass
22     %print-style;
23     %printout;
24     parentheses %yes-no; #IMPLIED
25     %optional-unique-id;
26 >
27 <!ELEMENT figure
28     (prefix?, figure-number?, suffix?, extend?, %editorial;)>
29 <!ELEMENT prefix (#PCDATA)>
30 <!ATTLIST prefix
31     %print-style;
32 >
33 <!ELEMENT figure-number (#PCDATA)>
34 <!ATTLIST figure-number
35     %print-style;
36 >
37 <!ELEMENT suffix (#PCDATA)>
38 <!ATTLIST suffix
39     %print-style;
40 >

```

64.2 Figured bass description

Figured bass is represented in MSR by classes defined in `src/formats/msr/msrFiguredBasses.h/.cpp`. There is class `msrFiguredBass`:

```

1 class EXP msrFiguredBass : public msrMeasureElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrNote          fFiguredBassUpLinkToNote;
12        S_msrVoice         fFiguredBassUpLinkToVoice; // for use in figured bass voices JMI
13
14        mfRational          fFiguredBassDisplayWholeNotes;
15
16        msrFiguredBassParenthesesKind
17                           fFiguredBassParenthesesKind;
18
19        std::list<S_msrBassFigure> fFiguredBassFiguresList;
20
21        msrTupletFactor      fFiguredBassTupletFactor;
22 };

```

The figured bass figures are defined in:

```

1 class EXP msrBassFigure : public msrElement
2 {
3     // ... ..
4
5     private:
6
7         // private fields
8         // -----
9
10        // upLinks
11        S_msrPart          fFigureUpLinkToPart;
12

```

```

13     msrBassFigurePrefixKind
14                               fFigurePrefixKind;
15     int                        fFigureNumber;
16     msrBassFigureSuffixKind
17                               fFigureSuffixKind;
18 };

```

Figured bass elements need special treatment since we need to determine their position in a figured bass voice. This is different than MusicXML, where they are simply *drawn* at the current music position, so to say.

They are handled this way:

- figured bass elements are stored in class `msrNote`:
- they are also stored in class `msrPart` and class `msrChord` and class `msrTuplet` (denormalization);

In class `msrNote`, there is:

```

1  // figured bass
2  void                        appendFiguredBassToNote (
3                               const S_msrFiguredBass& figuredBass);
4
5  const std::list<S_msrFiguredBass>&
6      getNoteFiguredBassesList () const
7      { return fNoteFiguredBassesList; }
8
9  // ... ..
10
11 // figured bass
12 // -----
13
14 std::list<S_msrFiguredBass>
15     fNoteFiguredBassesList;

```

64.3 Figured bass staves and voices

Every class `msrVoice` instance in `MusicFormats` belongs to an class `msrStaff` instance. Staves are created specifically to hold figured bass voices, using specific numbers defined in `src/formats/msr/msrParts.h`:

```

1  public:
2
3  // constants
4  // -----
5
6  // ... ..
7
8  #define msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER  20
9  #define msrPart::K_PART_FIGURED_BASS_VOICE_NUMBER  21

```

In class `msrStaff`, there is:

```

1  void                        registerFiguredBassVoiceByItsNumber (
2                               int            inputLineNumber ,
3                               const S_msrVoice& voice);

```

Class `msrPart` also contains:


```

1  // figured bass
2
3  S_msrVoice      createPartFiguredBassVoice (
4                  int      inputLineNumber,
5                  std::string currentMeasureNumber);
6
7  void            appendFiguredBassToPart (
8                  const S_msrVoice&          figuredBassSupplierVoice,
9                  S_msrFiguredBass figuredBass);
10
11 void            appendFiguredBassToPartClone (
12                const S_msrVoice&          figuredBassSupplierVoice,
13                const S_msrFiguredBass& figuredBass);

```

```

1  // figured bass
2
3  S_msrStaff      fPartFiguredBassStaff;
4  S_msrVoice      fPartFiguredBassVoice;

```

64.4 Figured bass staves creation

This is done in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.cpp.h/.cpp`:

```

1 S_msrVoice mxsr2msrSkeletonBuilder::createPartFiguredBassVoiceIfNotYetDone (
2     int      inputLineNumber,
3     const S_msrPart& part)
4 {
5     // is the figured bass voice already present in part?
6     S_msrVoice
7     partFiguredBassVoice =
8     part->
9     getPartFiguredBassVoice ();
10
11     if (! partFiguredBassVoice) {
12         // create the figured bass voice and append it to the part
13         partFiguredBassVoice =
14         part->
15         createPartFiguredBassVoice (
16             inputLineNumber,
17             fCurrentMeasureNumber);
18     }
19
20     return partFiguredBassVoice;
21 }

```

Method `msrPart::createPartFiguredBassVoice ()` creates the part figured bass staff and the part figured bass voice, and then registers the latter in the former:

```

1 S_msrVoice msrPart::createPartFiguredBassVoice (
2     int      inputLineNumber,
3     std::string currentMeasureNumber)
4 {
5     // ... ..
6
7     // create the part figured bass staff
8     int partFiguredBassStaffNumber =
9     msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER;
10
11     // ... ..
12
13     fPartFiguredBassStaff =
14     addHFiguredBassStaffToPart (

```

```

15     inputLineNumber);
16
17     // ... ..
18
19     // create the figured bass voice
20     int partFiguredBassVoiceNumber =
21         msrPart::K_PART_FIGURED_BASS_VOICE_NUMBER;
22
23     // ... ..
24
25     fPartFiguredBassVoice =
26         msrVoice::create (
27             inputLineNumber,
28             msrVoiceKind::kVoiceKindFiguredBass,
29             partFiguredBassVoiceNumber,
30             msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes,
31             fPartFiguredBassStaff);
32
33     // register the figured bass voice in the part figured bass staff
34     fPartFiguredBassStaff->
35         registerVoiceInStaff (
36             inputLineNumber,
37             fPartFiguredBassVoice);
38
39     // ... ..
40
41     return fPartFiguredBassVoice;
42 }

```

64.5 Translating figured bass from MXSR to MSR

This is done in `src/passes/mxsr2msr/`, and this is where the class `msrFiguredBass` instances are created. There several methods for Figured bass elements creation:

```

1 jacquesmenu@macmini:~/musicformats-git-dev/src/representations/msr > grep create
   msrFiguredBasses.h
2     static SMARTP<msrBassFigure> create (
3     SMARTP<msrBassFigure> createFigureNewbornClone (
4     SMARTP<msrBassFigure> createFigureDeepClone ( // JMI ???
5     static SMARTP<msrFiguredBass> create (
6     static SMARTP<msrFiguredBass> create (
7     SMARTP<msrFiguredBass> createFiguredBassNewbornClone (
8     SMARTP<msrFiguredBass> createFiguredBassDeepClone ();

```

The MSR score skeleton created in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.h/.cpp` contains the part groups, parts, staves and voices, as well as the number of measures. The voices do not contain any music elements yet.

A figured bass element belongs to `<part/>` in MusicXML, but we sometimes need to have it attached to a note.

`Fieldmxsr2msrSkeletonBuilder::ThereAreFiguredBassToBeAttachedToCurrentNote` it used when visiting an `S_FiguredBass` element to account for that:

```

1 void mxsr2msrSkeletonBuilder::visitStart ( S_figured_bass& elt )
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
5         std::stringstream ss;
6
7         ss <<
8         "--> Start visiting S_figured_bass" <<
9         ", figuredBassVoicesCounter = " << fFiguredBassVoicesCounter <<

```

```

10     ", line " << elt->getInputStartLineNumber ();
11
12     gWaeHandler->waeTraceWithoutInputLocation (
13         __FILE__, __LINE__,
14         ss.str ());
15 }
16 #endif // MF_TRACE_IS_ENABLED
17
18 /* JMI
19     several figured bass elements can be attached to a given note,
20     leading to as many figured bass voices in the current part JMI TRUE???
21 */
22
23 // take figured bass voice into account
24 ++fFiguredBassVoicesCounter;
25
26 fThereAreFiguredBassToBeAttachedToCurrentNote = true;
27 }

```

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-musicxml-figured-bass`, `-ofigbass` and it has not been created yet:

```

1 void mxsr2msrSkeletonBuilder::visitEnd ( S_note& elt )
2 {
3     // ... ..
4
5     // are there figured bass attached to the current note?
6     if (fThereAreFiguredBassToBeAttachedToCurrentNote) {
7         if (gGlobalMxsr2msrOahGroup->getIgnoreFiguredBasses ()) {
8 #ifdef MF_TRACE_IS_ENABLED
9             if (gTraceOahGroup->getTraceFiguredBasses ()) {
10                 gLog <<
11                     "Ignoring the figured bass elements" <<
12                     ", line " <<
13                     inputLineNumber <<
14                     std::endl;
15
16                 gWaeHandler->waeTraceWithoutInputLocation (
17                     __FILE__, __LINE__,
18                     ss.str ());
19             }
20 #endif // MF_TRACE_IS_ENABLED
21         }
22         else {
23             // create the part figured bass voice if not yet done
24             S_msrVoice
25                 partFiguredBassVoice =
26                 createPartFiguredBassVoiceIfNotYetDone (
27                     inputLineNumber,
28                     fCurrentPart);
29         }
30
31         fThereAreFiguredBassToBeAttachedToCurrentNote = false;
32
33         // ... ..
34     }

```

Creating the part figured bass voice is delegated to the part:

```

1 S_msrVoice mxsr2msrSkeletonBuilder::createPartFiguredBassVoiceIfNotYetDone (
2     int         inputLineNumber,
3     const S_msrPart& part)
4 {
5     // is the figured bass voice already present in part?

```

```

6   S_msrVoice
7       partFiguredBassVoice =
8           part->
9               getPartFiguredBassVoice ();
10
11  if (! partFiguredBassVoice) {
12      // create the figured bass voice and append it to the part
13      partFiguredBassVoice =
14          part->
15              createPartFiguredBassVoice (
16                  inputLineNumber,
17                  fCurrentMeasureNumber);
18  }
19
20  return partFiguredBassVoice;
21 }

```

64.5.1 First S_figured_bass visit

The first visit of S_figured_bass initializes the fields storing values to be gathered visiting subelements:

```

1  void mxsr2msrTranslator::visitStart ( S_figured_bass& elt )
2  {
3      int inputLineNumber =
4          elt->getInputStartLineNumber ();
5
6      #ifdef MF_TRACE_IS_ENABLED
7          if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
8              std::stringstream ss;
9
10             ss <<
11                 "--> Start visiting S_figured_bass" <<
12                 ", line " << inputLineNumber;
13
14             gWaeHandler->waeTraceWithoutInputLocation (
15                 __FILE__, __LINE__,
16                 ss.str ());
17         }
18     #endif // MF_TRACE_IS_ENABLED
19
20     ++fFiguredBassVoicesCounter;
21
22     std::string parentheses = elt->getAttributeValue ("parentheses");
23
24     fCurrentFiguredBassParenthesesKind =
25         msrFiguredBassParenthesesKind::kFiguredBassParenthesesNo; // default value
26
27     if (parentheses.size ()) {
28         if (parentheses == "yes")
29             fCurrentFiguredBassParenthesesKind =
30                 msrFiguredBassParenthesesKind::kFiguredBassParenthesesYes;
31
32         else if (parentheses == "no")
33             fCurrentFiguredBassParenthesesKind =
34                 msrFiguredBassParenthesesKind::kFiguredBassParenthesesNo;
35
36         else {
37             std::stringstream ss;
38
39             ss <<
40                 "parentheses value " << parentheses <<
41                 " should be 'yes' or 'no'";
42
43             musicxmlError (

```

```

44     gServiceRunData->getInputSourceName (),
45     inputLineNumber ,
46     __FILE__, __LINE__,
47     ss.str ());
48 }
49 }
50
51 fCurrentFiguredBassInputLineNumber    = -1;
52
53 fCurrentFigureNumber = -1;
54
55 fCurrentFigurePrefixKind = msrBassFigurePrefixKind::kBassFigurePrefix_UNKNOWN_;
56 fCurrentFigureSuffixKind = msrBassFigureSuffixKind::kBassFigureSuffix_UNKNOWN_;
57
58 fCurrentFiguredBassSoundingWholeNotes = mfrational (0, 1);
59 fCurrentFiguredBassDisplayWholeNotes  = mfrational (0, 1);
60
61 fOnGoingFiguredBass = true;
62 }

```

64.5.2 Second S_figured_bass visit

Upon the second visit of `S_figured_bass`, the class `msrFiguredBass` instance is created, populated and appended to `mxsr2msrTranslatorfPendingFiguredBassesList`:

```

1 void mxsr2msrTranslator::visitEnd ( S_figured_bass& elt )
2 {
3     int inputLineNumber =
4         elt->getInputStartLineNumber ();
5
6     #ifdef MF_TRACE_IS_ENABLED
7         if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
8             std::stringstream ss;
9
10            ss <<
11                "--> End visiting S_figured_bass" <<
12                ", line " << inputLineNumber;
13
14            gWaeHandler->waeTraceWithoutInputLocation (
15                __FILE__, __LINE__,
16                ss.str ());
17        }
18    #endif // MF_TRACE_IS_ENABLED
19
20    // create the figured bass element
21    #ifdef MF_TRACE_IS_ENABLED
22        if (gTraceOahGroup->getTraceFiguredBasses ()) {
23            std::stringstream ss;
24
25            ss <<
26                "Creating a figured bass" <<
27                ", line " << inputLineNumber << ":";
28
29            gWaeHandler->waeTraceWithoutInputLocation (
30                __FILE__, __LINE__,
31                ss.str ());
32        }
33    #endif // MF_TRACE_IS_ENABLED
34
35    // create the figured bass element
36    // if the sounding whole notes is 0/1 (no <duration /> was found), JMI ???
37    // it will be set to the next note's sounding whole notes later
38    S_msrFiguredBass
39        figuredBass =

```

```

40     msrFiguredBass::create (
41         inputLineNumber,
42         // JMI         fCurrentPart,
43         fCurrentFiguredBassSoundingWholeNotes,
44         fCurrentFiguredBassDisplayWholeNotes,
45         fCurrentFiguredBassParenthesesKind,
46         msrTupletFactor (1, 1));    // will be set upon next note handling
47
48 // attach pending figures to the figured bass element
49 if (! fPendingFiguredBassFiguresList.size ()) {
50     musicxmlWarning (
51         gServiceRunData->getInputSourceName (),
52         inputLineNumber,
53         "figured-bass has no figures contents, ignoring it");
54 }
55 else {
56     // append the pending figures to the figured bass element
57     for (S_msrBassFigure bassFigure : fPendingFiguredBassFiguresList) {
58         figuredBass->
59             appendFigureToFiguredBass (bassFigure);
60     } // for
61
62     // forget about those pending figures
63     fPendingFiguredBassFiguresList.clear ();
64
65     // append the figured bass element to the pending figured bass elements list
66     fPendingFiguredBassesList.push_back (figuredBass);
67 }
68
69 fOnGoingFiguredBass = false;
70 }

```

64.5.3 Attaching msrFiguredBass instances to notes

msrFiguredBass

The contents of `mxsr2msrTranslator.fPendingFiguredBassesList` is attached to the class `msrNote` instance in method

method `mxsr2msrTranslator::populateNote ()`:

```

1 void mxsr2msrTranslator::populateNote (
2     int         inputLineNumber,
3     const S_msrNote& newNote)
4 {
5     // ... ..
6
7     // handle the pending figured bass elements if any
8     if (fPendingFiguredBassesList.size ()) {
9         // get voice to insert figured bass elements into
10        S_msrVoice
11        voiceToInsertFiguredBassesInto =
12            fCurrentPart->
13                getPartFiguredBassVoice ();
14
15        // ... ..
16
17        handlePendingFiguredBasses (
18            newNote,
19            voiceToInsertFiguredBassesInto);
20
21        // reset figured bass counter
22        fFiguredBassVoicesCounter = 0;
23    }
24 }

```

64.5.4 Populating msrFiguredBass instances

In `src/formats/msr/mxsr2msrTranslator.cpp`, the class `msrFiguredBass` instances are populated further and attached to the note by method `mxsr2msrTranslator::handlePendingFiguredBasses ()`:

```

1 void mxsr2msrTranslator::handlePendingFiguredBasses (
2     const S_msrNote& newNote,
3     const S_msrVoice& voiceToInsertInto)
4 {
5     // ... ..
6
7     mfRational
8         newNoteSoundingWholeNotes =
9         fCurrentNote->
10             getSoundingWholeNotes (),
11         newNoteDisplayWholeNotes =
12         fCurrentNote->
13             getNoteDisplayWholeNotes ();
14
15     while (fPendingFiguredBassesList.size ()) { // recompute at each iteration
16         S_msrFiguredBass
17             figuredBass =
18             fPendingFiguredBassesList.front ();
19
20         /*
21          Figured bass elements take their position from the first
22          regular note (not a grace note or chord note) that follows
23          in score order. The optional duration element is used to
24          indicate changes of figures under a note.
25          */
26
27         // set the figured bass element's sounding whole notes
28         figuredBass->
29             setSoundingWholeNotes (
30                 newNoteSoundingWholeNotes,
31                 "handlePendingFiguredBasses()");
32
33         // set the figured bass element's display whole notes JMI useless???
34         figuredBass->
35             setFiguredBassDisplayWholeNotes (
36                 newNoteDisplayWholeNotes);
37
38         // set the figured bass element's tuplet factor
39         figuredBass->
40             setFiguredBassTupletFactor (
41                 msrTupletFactor (
42                     fCurrentNoteActualNotes,
43                     fCurrentNoteNormalNotes));
44
45         // append the figured bass to newNote
46         fCurrentNote->
47             appendFiguredBassToNote (
48                 figuredBass);
49
50         /* JMI
51          // get the figured bass voice for the current voice
52          S_msrVoice
53              voiceFiguredBassVoice =
54              voiceToInsertInto->
55                  getRegularVoiceForwardLinkToFiguredBassVoice ();
56
57         #ifdef MF_SANITY_CHECKS_ARE_ENABLED
58          // sanity check
59          mfAssert (
60              __FILE__, __LINE__,
61              voiceFiguredBassVoice != nullptr,

```

```

62     "voiceFiguredBassVoice is null");
63 #endif // MF_SANITY_CHECKS_ARE_ENABLED
64
65     // set the figuredBass's voice upLink
66     // only now that we know which figured bass voice will contain it
67     figuredBass->
68         setFiguredBassUpLinkToVoice (
69             voiceFiguredBassVoice);
70
71     // append the figured bass to the figured bass voice for the current voice
72     voiceFiguredBassVoice->
73         appendFiguredBassToVoice (
74             figuredBass);
75 */
76
77     // don't append the figured bass to the part figured bass voice
78     // before the note itself has been appended to the voice
79
80     // remove the figured bass from the list
81     fPendingFiguredBassesList.pop_front ();
82 } // while
83 }

```

```

1 %void mxsr2msrTranslator::copyNoteHarmoniesToChord (
2 % S_msrNote note, S_msrChord chord)
3 %{
4 % // copy note's harmony if any from the first note to chord
5 %
6 % const std::list<S_msrHarmony>&
7 %     noteHarmoniesList =
8 %     note->getNoteHarmoniesList ();
9 %
10 % if (noteHarmoniesList.size ()) {
11 %     std::list<S_msrHarmony>::const_iterator i;
12 %     for (i = noteHarmoniesList.begin (); i != noteHarmoniesList.end (); ++i) {
13 %         S_msrHarmony harmony = (*i);
14 %
15 % #ifdef MF_TRACE_IS_ENABLED
16 %         if (gTraceOahGroup->getTraceHarmonies ()) {
17 %             std::stringstream ss;
18 %
19 %             ss <<
20 %                 "Copying harmony '" <<
21 %                 harmony->asString () <<
22 %                 "' from note " << note->asString () <<
23 %                 " to chord '" << chord <<
24 %                 '\n' <<
25 %                 std::endl;
26 %
27 %             gWaeHandler->waeTraceWithoutInputLocation (
28 %                 __FILE__, __LINE__,
29 %                 ss.str ());
30 %         }
31 % #endif // MF_TRACE_IS_ENABLED
32 %
33 %         chord->
34 %             appendHarmonyToChord (harmony);
35 %
36 %     } // for
37 % }
38 %}
39 %

```


64.5.5 Inserting S_msrFiguredBass instances in the part figured bass voice

Method `msrVoice::appendNoteToVoice ()` in `src/formats/msr/msrNotes.cpp` inserts the figured bass elements in the part figured bass voice:

```

1 void msrVoice::appendNoteToVoice (const S_msrNote& note)
2 {
3     // ... ..
4
5     // are there figured bass elements attached to this note?
6     const std::list<S_msrFiguredBass>&
7         noteFiguredBassesList =
8         note->
9             getNoteFiguredBassesList ();
10
11     if (noteFiguredBassesList.size ()) {
12         // get the current part's figured bass voice
13         S_msrVoice
14             partFiguredBassVoice =
15             part->
16                 getPartFiguredBassVoice ();
17
18         for (S_msrFiguredBass figuredBass : noteFiguredBassesList) {
19             // append the figured bass element to the part figured bass voice
20             partFiguredBassVoice->
21                 appendFiguredBassToVoice (
22                     figuredBass);
23         } // for
24     }
25 };

```

64.6 Translating figured bass from MSR to MSR

In `src/passes/msr2msr/msr2msrTranslator.cpp`, a newborn clone of the figured bass element is created upon the first visit, stored in `msr2msrTranslatorfCurrentFiguredBassClone`, and appended to the current non grace note clone, the current chord clone or to the current voice clone, if the latter is a figured bass voice:

```

1 void msr2msrTranslator::visitStart (S_msrFiguredBass& elt)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gMsrOahGroup->getTraceMsrvVisitors ()) {
5             std::stringstream ss;
6
7             ss <<
8                 "--> Start visiting msrFiguredBass '" <<
9                 elt->asString () <<
10                 '\n' <<
11                 ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
12                 ", line " << elt->getInputStartLineNumber ();
13
14             gWaeHandler->waeTraceWithoutInputLocation (
15                 __FILE__, __LINE__,
16                 ss.str ());
17         }
18     #endif // MF_TRACE_IS_ENABLED
19
20     // create a figured bass element newborn clone
21     fCurrentFiguredBassClone =
22         elt->
23             createFiguredBassNewbornClone (
24                 fCurrentVoiceClone);

```

```

25
26 if (fOnGoingNonGraceNote) {
27     // append the figured bass to the current non-grace note clone
28     fCurrentNonGraceNoteClone->
29         appendFiguredBassToNote (
30             fCurrentFiguredBassClone);
31
32     // don't append the figured bass to the part figured bass, JMI ???
33     // this will be done below
34 }
35
36 /* JMI
37 else if (fOnGoingChord) {
38     // register the figured bass in the current chord clone
39     fCurrentChordClone->
40         setChordFiguredBass (fCurrentFiguredBassClone); // JMI
41 }
42 */
43
44 else if (fOnGoingFiguredBassVoice) { // JMI
45     /*
46     // register the figured bass in the part clone figured bass
47     fCurrentPartClone->
48         appendFiguredBassToPartClone (
49             fCurrentVoiceClone,
50             fCurrentFiguredBassClone);
51     */
52     // append the figured bass to the current voice clone
53     fCurrentVoiceClone->
54         appendFiguredBassToVoiceClone (
55             fCurrentFiguredBassClone);
56 }
57
58 else {
59     std::stringstream ss;
60
61     ss <<
62         "figured bass is out of context, cannot be handled: " <<
63         elt->asShortString ();
64
65     msrInternalError (
66         gServiceRunData->getInputSourceName (),
67         elt->getInputStartLineNumber (),
68         __FILE__, __LINE__,
69         ss.str ());
70 }
71 }

```

There are only fields updates upon the second visit:

```

1 void msr2msrTranslator::visitEnd (S_msrFiguredBass& elt)
2 {
3 #ifdef MF_TRACE_IS_ENABLED
4     if (gMsrOahGroup->getTraceMsrVisitors ()) {
5         std::stringstream ss;
6
7         ss <<
8             "--> End visiting msrFiguredBass '" <<
9             elt->asString () <<
10             '\n' <<
11             ", line " << elt->getInputStartLineNumber ();
12
13         gWaeHandler->waeTraceWithoutInputLocation (
14             __FILE__, __LINE__,
15             ss.str ());
16     }

```

```

17 #endif // MF_TRACE_IS_ENABLED
18
19     fCurrentFiguredBassClone = nullptr;
20 }

```

64.7 Translating figured bass from MSR to LPSR

The same occurs in `src/passes/msr2lpsr/msr2lpsrTranslator.cpp`: a newborn clone of the figured bass element is created and appended to the current non grace note clone, the current chord clone or to the current voice clone, if the latter is a figured bass voice:

```

1 void msr2lpsrTranslator::visitStart (S_msrFiguredBass& elt)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gMsrOahGroup->getTraceMsrVisitors ()) {
5             std::stringstream ss;
6
7             ss <<
8             "--> Start visiting msrFiguredBass '" <<
9             elt->asString () <<
10             '\n' <<
11             ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
12             ", line " << elt->getInputStartLineNumber ();
13
14             gWaeHandler->waeTraceWithoutInputLocation (
15                 __FILE__, __LINE__,
16                 ss.str ());
17         }
18     #endif // MF_TRACE_IS_ENABLED
19
20     // create a figured bass newborn clone
21     fCurrentFiguredBassClone =
22         elt->
23             createFiguredBassNewbornClone (
24                 fCurrentVoiceClone);
25
26     if (fOnGoingNonGraceNote) {
27         // append the figured bass to the current non-grace note clone
28         fCurrentNonGraceNoteClone->
29             appendFiguredBassToNote (
30                 fCurrentFiguredBassClone);
31
32         // don't append the figured bass to the part figured bass, JMI ???
33         // this will be done below
34     }
35
36     /* JMI
37     else if (fOnGoingChord) {
38         // register the figured bass in the current chord clone
39         fCurrentChordClone->
40             setChordFiguredBass (fCurrentFiguredBassClone); // JMI
41     }
42     */
43
44     else if (fOnGoingFiguredBassVoice) { // JMI
45         /*
46         // register the figured bass in the part clone figured bass
47         fCurrentPartClone->
48             appendFiguredBassToPartClone (
49                 fCurrentVoiceClone,
50                 fCurrentFiguredBassClone);
51         */
52         // append the figured bass to the current voice clone

```

```

53     fCurrentVoiceClone->
54         appendFiguredBassToVoiceClone (
55             fCurrentFiguredBassClone);
56 }
57
58 else {
59     std::stringstream ss;
60
61     ss <<
62         "figured bass is out of context, cannot be handled: " <<
63         elt->asShortString ();
64
65     msrInternalError (
66         gServiceRunData->getInputSourceName (),
67         elt->getInputStartLineNumber (),
68         __FILE__, __LINE__,
69         ss.str ());
70 }
71 }

```

Here too, there are only fields updates upon the second visit of `S_msrFiguredBass` instances:

```

1 void msr2lpsrTranslator::visitEnd (S_msrFiguredBass& elt)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gMsrOahGroup->getTraceMsrVisitors ()) {
5             std::stringstream ss;
6
7             ss <<
8                 "--> End visiting msrFiguredBass '" <<
9                 elt->asString () <<
10                 '\n' <<
11                 ", line " << elt->getInputStartLineNumber ();
12
13             gWaeHandler->waeTraceWithoutInputLocation (
14                 __FILE__, __LINE__,
15                 ss.str ());
16         }
17     #endif // MF_TRACE_IS_ENABLED
18
19     fCurrentFiguredBassClone = nullptr;
20 }

```

64.8 Translating figured bass from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

There is only one visit of class `msrFiguredBass` instances in `src/passes/lpsr2lilypond/lpsr2lilypondTranslator.cpp`.

The LilyPond code is generated only if the figured bass element belongs to a figured bass voice: this is where denormalization ends in the workflow:

```

1 void msr2lpsrTranslator::visitStart (S_msrFiguredBass& elt)
2 {
3     #ifdef MF_TRACE_IS_ENABLED
4         if (gMsrOahGroup->getTraceMsrVisitors ()) {
5             std::stringstream ss;
6
7             ss <<
8                 "--> Start visiting msrFiguredBass '" <<
9                 elt->asString () <<
10                 '\n' <<

```

```

11     ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
12     ", line " << elt->getInputStartLineNumber ();
13
14     gWaeHandler->waeTraceWithoutInputLocation (
15         __FILE__, __LINE__,
16         ss.str ());
17 }
18 #endif // MF_TRACE_IS_ENABLED
19
20 // create a figured bass newborn clone
21 fCurrentFiguredBassClone =
22     elt->
23         createFiguredBassNewbornClone (
24             fCurrentVoiceClone);
25
26 if (fOnGoingNonGraceNote) {
27     // append the figured bass to the current non-grace note clone
28     fCurrentNonGraceNoteClone->
29         appendFiguredBassToNote (
30             fCurrentFiguredBassClone);
31
32     // don't append the figured bass to the part figured bass, JMI ???
33     // this will be done below
34 }
35
36 /* JMI
37 else if (fOnGoingChord) {
38     // register the figured bass in the current chord clone
39     fCurrentChordClone->
40         setChordFiguredBass (fCurrentFiguredBassClone); // JMI
41 }
42 */
43
44 else if (fOnGoingFiguredBassVoice) { // JMI
45     /*
46     // register the figured bass in the part clone figured bass
47     fCurrentPartClone->
48         appendFiguredBassToPartClone (
49             fCurrentVoiceClone,
50             fCurrentFiguredBassClone);
51     */
52     // append the figured bass to the current voice clone
53     fCurrentVoiceClone->
54         appendFiguredBassToVoiceClone (
55             fCurrentFiguredBassClone);
56 }
57
58 else {
59     std::stringstream ss;
60
61     ss <<
62         "figured bass is out of context, cannot be handled: " <<
63         elt->asShortString ();
64
65     msrInternalError (
66         gServiceRunData->getInputSourceName (),
67         elt->getInputStartLineNumber (),
68         __FILE__, __LINE__,
69         ss.str ());
70 }
71 }

```

Chapter 65

Lyrics handling

Lyrics are presented at section [20.44](#) [Lyrics], page [216](#).

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 [20.45](#) [MIDI], page [217](#).

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 XIV

MusicFormats Scripting Language (MFSL)

Chapter 67

MFSL (MusicFormats Scripting Language)

67.1 A script example

This script illustrates the basic features of MFSL:

```
1 #!//Users/jacquesmenu/musicformats-git-dev/build/bin/mfsl
2
3 # the MusicFormats service to be used
4 service : xml2ly
5
6 # the input file
7 input :test.mfsl
8
9 # parts
10 -keep-musicxml-part-id P1
11
12 # the voices choice
13 choice VOICES_CHOICE : voice1Only | voice2Only ;
14 # could be : choice VOICES_CHOICE : ... .. ;
15
16 set VOICES_CHOICE = voice1Only ;
17 # change this to voice2Only to switch to another subset of options
18 # could even be parameter to the script such a $1
19
20 # choose which options to use according to VOICES_CHOICE
21 case VOICES_CHOICE :
22   voice1Only:
23     -title "Joli morceau - voix 1"
24     -ignore-msr-voice Part_POne_Staff_One_Voice_Two
25   ;
26
27   voice2Only:
28     -title "Joli morceau - voix 2"
29     --ignore-msr-voice Part_POne_Staff_One_Voice_One
30
31     -display-options-values
32
33     -global-staff-size 25.5
34   ;
35 ;
```

This first line of an MFSL script is the so-called *shebang* containing the path to the interpreter, allow for running such scripts by their name provided they are made executable.

67.2 Implementation principles

MFSL is implemented with the `flex` and `bison` C++ code generators:

- `src/interpreters/mfsl/mfslScanner.ll` contains the `flex` lexical description of MFSL.
It is used to create `src/interpreters/mfsl/mfslScanner.cpp`;
- `src/interpreters/mfsl/mfslParser.yy` is the syntax and semantics description of MFSL.
From it, `bison` creates `src/interpreters/mfsl/mfslParser.h`, `src/interpreters/mfsl/mfslParser.cpp` and `src/interpreters/mfsl/mfslParser.output`.
The latter file can be used to check the grammar, in particular if LR conflicts are detected;
- communication between the code generated this way is done by a so-called *driver*, along the lines of the C++-calc example provided by `bison` v3.8.1;
- the way the tokens description is shared by the scanner and parser is described at section 67.7 [Tokens description], page 387;
- the whole power of OAH is used to handle the contents of MFSL scripts as well as the options to the MFSL interpreter itself.

Only the predefined `bool` type is used, since the generated C++ code relies on this. This is why `getValue ()` is used in `src/clisamples/mfsl.cpp`:

```

1  std::string      theMfService;
2  std::string      theInputFile;
3  oahOptionsAndArguments optionsAndArguments;
4
5  err =
6      launchMfslInterpreter (
7          inputSourceName,
8          traceScanning.getValue (),
9          traceParsing.getValue (),
10         displayTokens.getValue (),
11         displayNonTerminals.getValue (),
12         theMfService,
13         theInputFile,
14         optionsAndArguments);

```

67.3 The contents of the MFSL folder

`src/interpreters/mfsl/location.hh` defines the `yy::location` class, that contains the script file name and input line number:

```

1  jacquesmenu@macmini: ~/musicformats-git-dev/src/interpreters/mfsl > ls -sal
2  total 776
3   0 drwxr-xr-x  28 jacquesmenu  staff      896 Mar 15 05:16 .
4   0 drwxr-xr-x@  4 jacquesmenu  staff      128 Mar 13 00:47 ..
5  16 -rw-r--r--@  1 jacquesmenu  staff    6148 Mar 14 10:18 .DS_Store
6   8 -rw-r--r--@  1 jacquesmenu  staff    1266 Mar 15 05:15 Makefile
7  16 -rw-r--r--@  1 jacquesmenu  staff    7864 Mar 15 05:16 location.hh
8  24 -rw-r--r--@  1 jacquesmenu  staff   10106 Mar 14 18:26 mfslBasicTypes.cpp
9  16 -rw-r--r--@  1 jacquesmenu  staff    4568 Mar 14 18:16 mfslBasicTypes.h
10   8 -rw-r--r--@  1 jacquesmenu  staff    1585 Mar 15 05:12 mfslDriver.cpp
11   8 -rw-r--r--@  1 jacquesmenu  staff    3413 Mar 15 05:12 mfslDriver.h
12   8 -rw-r--r--@  1 jacquesmenu  staff    3041 Mar  9 07:35 mfslInterpreterComponent.cpp
13   8 -rw-r--r--@  1 jacquesmenu  staff     661 Mar  9 07:02 mfslInterpreterInterface.h

```

```

14 24 -rw-r--r--@ 1 jacquesmenu staff 11981 Mar 10 11:38 mfs1InterpreterInsiderHandler.
    cpp
15 16 -rw-r--r--@ 1 jacquesmenu staff 5270 Mar 10 07:11 mfs1InterpreterInsiderHandler.h
16 8 -rw-r--r--@ 1 jacquesmenu staff 1161 Mar 15 05:13 mfs1InterpreterInterface.h
17 16 -rw-r--r--@ 1 jacquesmenu staff 7116 Mar 14 15:53 mfs1Interpreter0ah.cpp
18 16 -rw-r--r--@ 1 jacquesmenu staff 4692 Mar 14 15:51 mfs1Interpreter0ah.h
19 24 -rw-r--r--@ 1 jacquesmenu staff 10070 Mar 14 15:53 mfs1InterpreterRegularHandler.
    cpp
20 8 -rw-r--r--@ 1 jacquesmenu staff 3533 Mar 9 08:22 mfs1InterpreterRegularHandler.h
21 88 -rw-r--r-- 1 jacquesmenu staff 43880 Mar 15 05:16 mfs1Parser.cpp
22 96 -rw-r--r-- 1 jacquesmenu staff 45868 Mar 15 05:16 mfs1Parser.h
23 24 -rw-r--r--@ 1 jacquesmenu staff 10722 Mar 13 16:57 mfs1Parser.output
24 16 -rw-r--r--@ 1 jacquesmenu staff 5930 Mar 14 18:19 mfs1Parser.yy
25 136 -rw-r--r-- 1 jacquesmenu staff 68514 Mar 15 05:16 mfs1Scanner.cpp
26 24 -rw-r--r--@ 1 jacquesmenu staff 11251 Mar 15 05:12 mfs1Scanner.ll
27 144 -rw-r--r--@ 1 jacquesmenu staff 71091 Mar 15 05:14 mfs1Scanner.log
28 8 -rw-r--r--@ 1 jacquesmenu staff 2047 Mar 9 11:45 mfs1Wae.cpp
29 8 -rw-r--r--@ 1 jacquesmenu staff 3681 Mar 9 11:44 mfs1Wae.h
30 8 -rwxr-xr-x@ 1 jacquesmenu staff 817 Mar 14 18:20 test.mfs1

```

67.4 The MFSL basic types

67.5 The MFSL Makefile

This Makefile is quite simple: the options to flex and bison are placed in `src/interpreters/mfs1/mfs1Scanner.ll` and `src/interpreters/mfs1/mfs1Parser.yy`, respectively:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/interpreters/mfs1 > cat Makefile
2 # ... ..
3
4 # variables
5 # -----
6
7 MAKEFILE = Makefile
8
9 GENERATED_FILES = mfs1Parser.h mfs1Scanner.cpp mfs1Parser.cpp
10
11 BISON = bison
12 FLEX = flex
13
14 CXXFLAGS = -I.. -DMAIN
15
16
17 # implicit target
18 # -----
19
20 all : $(GENERATED_FILES)
21
22
23 # generation rules
24 # -----
25
26 mfs1Scanner.cpp : $(MAKEFILE) mfs1Scanner.ll
27     $(FLEX) -omfs1Scanner.cpp mfs1Scanner.ll
28
29
30 mfs1Parser.h mfs1Parser.cpp : $(MAKEFILE) mfs1Parser.yy
31     $(BISON) --defines=mfs1Parser.h -obj mfs1Parser.cpp mfs1Parser.yy
32
33
34 # clean
35 # -----

```

```

36
37 clean:
38   rm -f $(GENERATED_FILES)

```

67.6 Locations handling

67.7 Tokens description

The tokens are described in `src/interpreters/mfsl/mfslParser.yy`, such as:

```

1 %token <std::string> OPTION "option"

```

Both `OPTION` and `"option"` can be used in the productions, but the grammar is more readable if the capitalized name is used:

```

1 Option
2   : OPTION
3     {
4       if (drv.getDisplayNonTerminals ()) {
5         gLog <<
6           " ==> option " << $1 <<
7           std::endl << std::endl;
8       }
9
10      $$ = oahOptionNameAndValue::create ($1, "");
11    }
12
13  | OPTION OptionValue
14    {
15      if (drv.getDisplayNonTerminals ()) {
16        gLog <<
17          " ==> option " << $1 << ' ' << $2 <<
18          std::endl << std::endl;
19      }
20
21      $$ = oahOptionNameAndValue::create ($1, $2);
22    }
23 ;

```

In case of error, `"option"` is used to display a message to the user.

The name `OPTION` is used in `src/interpreters/mfsl/mfslScanner.ll` prefixed by `yy::parser::make_`:

```

1 "--{name} |
2 "--{name} {
3   if (drv.getTraceTokens ()) {
4     gLog << "--> " << drv.getScannerLocation () <<
5     ": option [" << yytext << ']' <<
6     std::endl;
7   }
8   return yy::parser::make_OPTION (yytext, loc);
9 }

```

The suffix after `make_` has to be defined in the `src/interpreters/mfsl/mfslParser.yy` for this to do the link between the Flex-generated and Bison-generated code:

```

1 %token <std::string> OPTION "option"

```

In `src/interpreters/mfsl/mfslParser.cpp`, this becomes:

```
1 case symbol_kind::S_OPTION: // "option"
```

We don't have to create method `yy::parser::make_OPTION ()` ourselves, though: it is taken care of by Bison itself, since it returns a type `char*`.

The `calc++` example in the `bison` documentation contains the case of numbers:

```
1 %{
2 // A number symbol corresponding to the value in S.
3 yy::parser::symbol_type
4 make_NUMBER (const std::string &s, const yy::parser::location_type& loc);
5 %}
6
7 // ... ..
8
9 yy::parser::symbol_type
10 make_NUMBER (const std::string &s, const yy::parser::location_type& loc)
11 {
12     errno = 0;
13     long n = strtol (s.c_str(), NULL, 10);
14     if (! (INT_MIN <= n && n <= INT_MAX && errno != ERANGE))
15         throw yy::parser::syntax_error (loc, "integer is out of range: " + s);
16     return yy::parser::make_NUMBER ((int) n, loc);
17 }
```

67.8 The driver

Class `mfslDriver` contains everything needed to let the code generated by `flex` and `bison` communicate with each other, as well as any work variables needed during the analysis of MFSL input. This latter point allows for multiple analyzers to coexist.

`src/interpreters/mfsl/mfslDriver.h` contains a prototype of function `yylex ()`:

```
1 // -----
2 // Give Flex the prototype of yylex we want ...
3 # define YY_DECL \
4     yy::parser::symbol_type yylex (mfslDriver& drv)
5 // ... and declare it for the parser's sake.
6 YY_DECL;
```

Then it contains the declaration of class `mfslDriver`:

```
1 // Conducting the whole scanning and parsing of MFSL
2 class mfslDriver
3 {
4     public:
5
6         // constants
7         // -----
8
9         static const std::string K_ALL_PSEUDO_LABEL_NAME;
10
11         // // constructor/destructor
12         // -----
13
14         mfslDriver ();
15
16         virtual ~mfslDriver ();
17
18         // ... ..
19 }
```

```

20 public:
21
22     // public services
23     // -----
24
25     // run the parser, return 0 on success
26     int                parseInput_Pass1 ();
27
28     // handling the scanner
29     void                scanBegin ();
30     void                scanEnd ();
31
32     // ... ..
33
34 private:
35
36     // private fields
37     // -----
38
39     // the name of the MusicFormats service
40     std::string          fService;
41
42     // the name of the MusicFormats script
43     std::string          fScriptName;
44
45     // the names of the input sources
46     std::list<std::string> fInputSoucesList;
47
48
49
50     // scanning
51     bool                fTraceScanning;
52     mfs::location        fScannerLocation;
53
54     // ... ..
55 };

```

The definitions are placed in two files due to the specificity of the sharing of variables and function in the **flex** and **bison**-generated code:

- `src/interpreters/mfs/mfsDriver.cpp` contains method `mfsDriver::parseInput_Pass1 ()`, that runs the parser:

```

1 int mfsDriver::parseInput_Pass1 ()
2 {
3     // initialize scanner location
4     fScannerLocation.initialize (
5         &fScriptName);
6
7     // begin scan
8     scanBegin ();
9
10    if (fScriptName.empty () || fScriptName == "-") {
11        fScriptName = "stdin"; // nicer for warning and error messages
12    }
13
14    // do the parsing
15    mfs::parser theParser (*this);
16
17    theParser.set_debug_level (
18        fTraceParsing);
19
20    int parseResult = theParser ();
21
22    // end scan

```

```

23 |     scanEnd ();
24 |
25 |     // ... ..
26 |
27 |     // do the final semantics check
28 |     finalSemanticsCheck ();
29 |
30 |     return parseResult;
31 | }

```

- the remaining code is placed in the third part (service code) of `src/interpreters/mfsl/mfslScanner.ll`, since it needs to access variables in the code generated by `flex`:

```

1 | void mfslDriver::scanBegin ()
2 | {
3 |     yy_flex_debug = fTraceScanning;
4 |
5 |     if (fScriptName.empty () || fScriptName == "-") {
6 |         yyin = stdin;
7 |     }
8 |
9 |     else if (!(yyin = fopen (fScriptName.c_str (), "r")))
10 |     {
11 |         std::stringstream ss;
12 |
13 |         char*
14 |             errorCString =
15 |             mfStrErrorCString ();
16 |
17 |         if (errorString != nullptr) {
18 |             ss <<
19 |             gLanguage->cannotOpenScriptForWriting (fScriptName) <<
20 |             ": " <<
21 |             errorString <<
22 |             std::endl;
23 |
24 |             mfslFileError (
25 |                 fScriptName,
26 |                 ss.str ());
27 |         }
28 |     }
29 | }
30 |
31 | void mfslDriver::scanEnd ()
32 | {
33 |     fclose (yyin);
34 | }

```

67.9 Lexical analysis

The lexical definition of MFSL in `src/interpreters/mfsl/mfslScanner.ll` is described below.

67.9.1 Flex options

The `prefix` is used to allow for multiple `flex`-generated analyzers to coexist:

```

1 | %option prefix="mfsl"
2 |
3 | %option yylineno
4 |
5 | %option noyywrap

```

```

6
7 %option nounput noinput debug interactive

```

67.9.2 Flex regular expressions

The basic ones are:

```

1 blank          [ \t\r]
2 endOfLine      [\n]
3 character      .
4
5 letter         [A-Za-zéèëääöþýüí]
6 digit          [0-9]
7
8 name           {letter}(_|-|\.|{letter}|{digit})*
9 integer        {digit}+
10 exponent      [eE][+-]?{integer}
11
12 singleQuote    [']
13 doubleQuote    ["]
14 tabulator      [\t]
15 backSlash      [\\]
16
17 ... ..
18
19 %{
20     // Code run each time a pattern is matched.
21     # define YY_USER_ACTION  loc.columns (yyleng);
22 %}

```

Some exclusive modes are used for strings and comments:

```

1 %x          SINGLE_QUOTED_STRING_MODE
2 %x          DOUBLE_QUOTED_STRING_MODE
3
4 %x          COMMENT_TO_END_OF_LINE_MODE
5 %x          PARENTHESESIZED_COMMENT_MODE

```

Strings must be stored in a private buffer:

```

1 /* strings */
2
3 #define          STRING_BUFFER_SIZE 1024
4 char            pStringBuffer [STRING_BUFFER_SIZE];
5
6 // A handy shortcut to the location held by the mfsldriver
7 mfsll::location& loc = drv.getScannerLocationNonConst ();

```

Locating the tokens in the the MFSL input text is done with:

```

1 // Code run each time yylex() is called
2 loc.step ();

```

This lead for example to:

```

1 {blank} {
2     loc.step ();
3 }
4
5 {endOfLine} {
6     loc.lines (yyleng); loc.step ();
7 }

```


The numbers are handled by:

```

1 {integer} "." {integer} ({exponent})? |
2 {integer}{exponent} {
3   if (drv.getTraceTokens ()) {
4     gLog <<
5       "--> " << drv.getScannerLocation () <<
6       " double: " << yytext <<
7       std::endl;
8   }
9   return yy::parser::make_DOUBLE (yytext, loc);
10 }
11
12 {integer} {
13   if (drv.getTraceTokens ()) {
14     gLog <<
15       "--> " << drv.getScannerLocation () <<
16       " integer: " << yytext <<
17       std::endl;
18   }
19   return yy::parser::make_INTEGER (yytext, loc);
20 }

```

The MFSL keywords are handled with the make_... facility:

```

1 "service" {
2   if (drv.getTraceTokens ()) {
3     gLog <<
4       "--> " << drv.getScannerLocation () << ": " << yytext <<
5       std::endl;
6   }
7   return yy::parser::make_SERVICE (loc);
8 }

```

The names and the options are handled by:

```

1 {name} {
2   if (drv.getDisplayTokens ()) {
3     gLog << "--> " << drv.getScannerLocation () <<
4       ": name [" << yytext << ']' <<
5       std::endl;
6   }
7
8   loc.begin.column += yyleng;
9   loc.step ();
10
11   return
12     mfs1::parser::make_NAME (yytext, loc);
13 }
14
15
16 "--{name} |
17 "--{name} {
18   if (drv.getTraceTokens ()) {
19     gLog << "--> " << drv.getScannerLocation () <<
20       ": option [" << yytext << ']' <<
21       std::endl;
22   }
23   return yy::parser::make_OPTION (yytext, loc);
24 }
25
26
27
28 "(" {
29   if (drv.getTraceTokens ()) {

```

```

31     gLog <<
32         "--> " << drv.getScannerLocation () << ": " << yytext <<
33         std::endl;
34     }
35     return yy::parser::make_LEFT_PARENTHESIS (loc);
36 }

```

The catchall rule issues an error message:

```

1  . {
2      throw mfs1::parser::syntax_error (
3          loc,
4          "### invalid character: " + std::string (yytext));
5  }

```

And the end of the MFSL input is handled this way:

```

1  <<EOF>> {
2      return
3      mfs1::parser::make_YEEOF (loc);
4  }

```

67.10 Syntax and semantic analysis

67.10.1 Bison options for MFSL

Setting `api.prefix` allows for multiple analyzers to coexist:

```

1  %skeleton "lalr1.cc" // -*- C++ -*-
2  %require "3.8.1"
3  %defines
4
5  %define api.prefix {mfs1}
6
7  %define api.token.raw
8
9  %define api.token.constructor
10 %define api.value.type variant
11 %define parse.assert
12
13 %code requires {
14     #include <string>
15
16     class    mfs1Driver;
17 }
18
19 // the parsing context
20 %param { mfs1Driver& drv } // declaration, any parameter name is fine
21
22 %verbose // to produce mfs1Parser.output
23
24 %locations
25
26 // other Bison options
27 %define parse.trace
28 %define parse.error detailed
29 %define parse.lac full
30 // %define api.pure full
31
32 %printer { yyo << $$; } <*>;
33
34

```

```

35 %code {
36     #include "mfsBasicTypes.h"
37 }

```

67.10.2 The MFSL tokens

The MFSL tokens are:

```

1 %define api.token.prefix {MFSL_TOK_}
2
3 %token
4     BAR           "|"
5     AMPERSAND     "&"
6     EQUAL         "="
7     SEMICOLON     ";"
8     COLON         ":"
9     COMMA         ","
10
11     SERVICE       "service"
12     INPUT         "input"
13
14     CHOICE         "choice"
15     DEFAULT       "default"
16
17     CASE          "case"
18
19     SELECT        "select"
20     ALL           "all"
21 ;
22
23 %code {
24     #include "mfsDriver.h"
25 }
26
27 %token <std::string> INTEGER "integer number"
28 %token <std::string> DOUBLE  "double number"
29
30 %token <std::string> SINGLE_QUOTED_STRING "single quoted_string"
31 %token <std::string> DOUBLE_QUOTED_STRING "double quoted_string"
32
33 %token <std::string> NAME "name"
34
35 %token <std::string> OPTION "option"

```

67.10.3 The MFSL non-terminals and axiom

They are:

```

1 // the MFSL non-terminals
2 // -----
3
4 %nterm <std::string> Number
5
6 %nterm <std::string> SingleString
7 %nterm <std::string> std::string
8
9 %nterm <std::string> OptionValue
10
11 %nterm <std::string> LabelName
12
13
14 // the MFSL axiom

```

```

15 //-----
16
17 %start Script

```

67.11 Interface to the MFSL parser

This is provided by `src/interpreters/mfsl/mfslInterpreterInterface.h`:

```

1 EXP extern mfMusicformatsErrorKind launchMfslInterpreter (
2     const std::string&      inputSourceName,
3     bool                    traceScanning,
4     bool                    traceParsing,
5     bool                    displayTokens,
6     bool                    displayNonTerminals,
7     std::string&            theMfService,
8     std::string&            theInputFile,
9     oahOptionsAndArguments& optionsAndArguments);

```

The definition of this function is placed in `src/interpreters/mfsl/mfslScanner.ll`:

```

1 EXP mfMusicformatsErrorKind launchMfslInterpreter (
2     const std::string&      inputSourceName,
3     bool                    traceScanning,
4     bool                    traceParsing,
5     bool                    displayTokens,
6     bool                    displayNonTerminals,
7     std::string&            theMfService,
8     std::string&            theInputFile,
9     oahOptionsAndArguments& optionsAndArguments)
10 {
11     mfMusicformatsErrorKind
12     result =
13         mfMusicformatsErrorKind::kMusicformatsError_NONE;
14
15     mfslDriver
16     theDriver (
17         traceScanning,
18         traceParsing,
19         displayTokens,
20         displayNonTerminals);
21
22     int parseResult =
23         theDriver.parseFile (inputSourceName);
24
25     gLog <<
26         "--> parseResult: " << parseResult <<
27         std::endl;
28
29     if (! parseResult) {
30         result =
31             mfMusicformatsErrorKind::kMusicformatsErrorInvalidFile;
32     }
33
34     gLog <<
35         "inputFileName: " << theDriver.getInputFileName () <<
36         std::endl <<
37         "serviceName: " << theDriver.getServiceName () <<
38         std::endl;
39
40     theMfService      = theDriver.getServiceName ();
41     theInputFile      = theDriver.getInputFileName ();
42
43     return result;
44 }

```

67.12 Running the example MFSL script

Let's show show the MFSL interpreter uses the options above:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/src/interpreters/mfsl > ./test.mfsl -display-
  tokens -display-non-terminals -display-options-values
2 The options values for //Users/jacquesmenu/musicformats-git-dev/build/bin/mfsl are:
3 MFSL group (-help-mfsl-group, -hmfsl-group), 2 atoms selected:
4 -----
5 MFSL (-help-mfsl, -hmfsl), 2 atoms selected:
6 fTraceTokens : true, selected option
7 fDisplayNonTerminals : true, selected option
8
9 Options and help group (-help-oah-group, -hoah-group), 1 atom selected:
10 -----
11 Options and help (-help-oah, -hoah), 1 atom selected:
12 fDisplayOptionsValues : true, selected option
13
14
15 --> ./test.mfsl:2.1-39: service
16 --> ./test.mfsl:2.41: :
17 --> ./test.mfsl:2.43-48: name [xml2ly]
18 ==> service: xml2ly
19
20 --> ./test.mfsl:4.1-22: input
21 --> ./test.mfsl:4.24: :
22 --> ./test.mfsl:4.25-33: name [test.mfsl]
23 ==> input: test.mfsl
24
25 --> test.mfsl:6.11-32: option [-keep-musicxml-part-id]
26 --> test.mfsl:6.34-35: name [P1]
27 ==> option -keep-musicxml-part-id P1
28
29 --> test.mfsl:8.1-26: choice
30 --> test.mfsl:8.28-40: name [VOICES_CHOICE]
31 --> test.mfsl:8.42: :
32 --> test.mfsl:8.44-53: name [voice10only]
33 --> test.mfsl:8.55: |
34 --> test.mfsl:8.57-66: name [voice20only]
35 --> test.mfsl:8.68: ;
36 ==> ChoiceDeclaration VOICES_CHOICE : ...
37
38 --> test.mfsl:10.1-3: set
39 --> test.mfsl:10.5-17: name [VOICES_CHOICE]
40 --> test.mfsl:10.19: =
41 --> test.mfsl:10.21-30: name [voice10only]
42 --> test.mfsl:10.32: ;
43 ==> ChoiceSetting, set VOICES_CHOICE = voice10only
44
45 --> test.mfsl:12.1-61: case
46 --> test.mfsl:12.63-75: name [VOICES_CHOICE]
47 --> test.mfsl:12.77: :
48 --> test.mfsl:13.2-11: name [voice10only]
49 --> test.mfsl:13.12: :
50 --> test.mfsl:14.5-10: option [-title]
51 --> test.mfsl:14.12-34: double quoted std::string [""]
52 ==> option -title ""
53
54 --> test.mfsl:15.5-21: option [-ignore-msr-voice]
55 --> test.mfsl:15.23-51: name [Part_P0ne_Staff_One_Voice_Two]
56 ==> option -ignore-msr-voice Part_P0ne_Staff_One_Voice_Two
57
58 --> test.mfsl:16.3: ;
59 ==> Case voice10only : ...
60
61 --> test.mfsl:18.2-11: name [voice20only]

```

```

62 --> test.mfsl:18.12: :
63 --> test.mfsl:19.5-10: option [-title]
64 --> test.mfsl:19.12-34: double quoted std::string ["]
65 ==> option -title "
66
67 --> test.mfsl:20.5-22: option [--ignore-msr-voice]
68 --> test.mfsl:20.24-52: name [Part_P0ne_Staff_One_Voice_One]
69 ==> option --ignore-msr-voice Part_P0ne_Staff_One_Voice_One
70
71 --> test.mfsl:22.5-27: option [-display-options-values]
72 --> test.mfsl:24.5-22: option [-global-staff-size]
73 ==> option -display-options-values
74
75 --> test.mfsl:24.24-27 double: 25.5
76 ==> option -global-staff-size 25.5
77
78 --> test.mfsl:25.3: ;
79 ==> Case voice20nly : ...
80
81 --> test.mfsl:26.1: ;
82 ==> CaseStatement, VOICES_CHOICE : ...
83
84 --> parseResult: 0
85 ==> inputFileName: test.mfsl
86 ==> serviceName:      xml2ly
87 jacquesmenu@macmini: ~/musicformats-git-dev/src/interpreters/mfsl >

```

67.12.1 Error recovery

The MFSL interpreter uses a variant of the *stopper sets* method that was present in the early Pascal and Pascal-S converters. The latter passed a set of tokens not to be overtaken to the procedures in charge of accepting the various statements in the language. Strangely enough, this was not done for declarations.

We use a stack of tokens sets that grows and shrinks in parallel with the accepting functions, to know more contextual informations when deciding wether to consume a token or not. The corresponding term is it shift when building the analysis tables in LR technology.

Part XV

Music Scores Description Language (MSDL)

Chapter 68

MSDL (Music Scores Description Language)

MSDL is an attempt at a description of music score in a non-linear way, much like a painter puts touches of paint on his work. This is also what users do with GUI music scoring applications, but scores textual descriptions such as LilyPond and Guido impose a linear, left to right, writing of the scores contents.

Contrary to LilyPond, the `|` token in MSDL is not the end of a measure. Writing `|2` means that the music that follows will be placed in a new layer in measure 2.

68.1 Main features of MSDL

They are:

- note are written much like in LilyPond such as `b2...`;
- the keywords such as `pitches` and `music`, are reserved;
- they are available in a number of languages such as english, french, german and italian. It is easy to add other languages;

A first, limited converter is provided by MusicFormats with service `msdl`. It also performs reserved keywords translation from one language to another:

68.2 MSDL basic types

Some types used throughout MSR are defined in `src/formats/msdl/msdlEnumTypes.h/.cpp`:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msdl > egrep -rIn  '^// '
    msdlEnumTypes.h
2 msdlEnumTypes.h:28:// user languages
3 msdlEnumTypes.h:52:// comments types
4 msdlEnumTypes.h:74:// initialization
```


68.3 What the MSDL converter does

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/msdl > msdl -about
2 What msdlConverter does:
3
4     This multi-pass converter performs various passes depending on the output generated,
5     which should be specified a '-lilypond', '-braille', '-musicxml' or '-guido' option.
6
7     Other passes are performed according to the options, such as
8     displaying views of the internal data or printing a summary of the score.
9
10    The activity log and warning/error messages go to standard error.
11
12    The output format is selected via options.

```

68.3.1 LilyPond generation

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/msdl > msdl -lilypond -about
2 What msdlConverter does:
3
4     This multi-pass converter basically performs 3 passes when generating LilyPond output
5     output:
6
7     Pass 1:  converts the MSDL input into a first MSR
8     Pass 2:  converts the first MSR into a second MSR;
9     Pass 3:  converts the second MSR into a
10             LilyPond Score Representation (LPSR);
11     Pass 4:  converts the LPSR to LilyPond code
12             and writes it to standard output.
13
14     Other passes are performed according to the options, such as
15     displaying views of the internal data or printing a summary of the score.
16
17     The activity log and warning/error messages go to standard error.

```

68.3.2 Braille generation

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/msdl > msdl -braille -about
2 What msdlConverter does:
3
4     This multi-pass converter basically performs 4 passes when generating braille output
5     output:
6
7     Pass 1:  converts the MSDL input into a first MSR
8     Pass 2:  converts the first MSR into a second MSR;
9     Pass 3a: converts the second MSR into a
10             Braille Score Representation (BSR)
11             containing one Braille page per MusicXML page;
12     Pass 3b: converts the BSR into another BSR
13             with as many Braille pages as needed
14             to fit the line and page lengths;
15     Pass 4:  converts the BSR to Braille text
16             and writes it to standard output.)
17
18     In this preliminary version, pass 2b merely clones the BSR it receives.
19
20     Other passes are performed according to the options, such as
21     displaying views of the internal data or printing a summary of the score.
22
23     The activity log and warning/error messages go to standard error.

```

68.3.3 MusicXML generation

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/msdl > msdl -musicxml -about
2 What msdlConverter does:
3
4     This multi-pass converter basically performs 4 passes when generating MusicXML output
5     output:
6
7         Pass 1:  converts the MSDL input into a first MSR
8         Pass 2:  converts the first MSR into a second MSR;
9         Pass 3:  converts the second MSR into an MusicXML tree;
10        Pass 4:  converts the MusicXML tree to MusicXML code
11                and writes it to standard output.
12
13    Other passes are performed according to the options, such as
14    displaying views of the internal data or printing a summary of the score.
15
16    The activity log and warning/error messages go to standard error.

```

68.3.4 Guido generation

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/msdl > msdl -guido -about
2 What msdlConverter does:
3
4     This multi-pass converter basically performs 4 passes when generating Guido output
5     output:
6
7         Pass 1:  converts the MSDL input into a first MSR
8         Pass 2:  converts the first MSR into a second MSR;
9         Pass 3:  converts the second MSR into an MusicXML tree;
10        Pass 4:  converts the MusicXML tree to Guido code
11                and writes it to standard output.
12
13    Other passes are performed according to the options, such as
14    displaying views of the internal data or printing a summary of the score.
15
16    The activity log and warning/error messages go to standard error.
17 jacquesmenu@macmini: ~/musicformats-git-dev/msdl >

```

68.4 A first example

HelloWorld.msd is a minimal example:

```

1 %{
2     The unavoidable HelloWorld score
3     %}
4
5
6 % the language used for the pitches
7 % -----
8
9 pitches english           % default is english
10
11
12 % is there an anacrusis?
13 % -----
14
15 anacrusis                % measure numbers start at 0
16
17
18 % the structure

```

```

19 % -----
20
21 % score helloWorld = "Hello World in MSPL" {
22
23   music {
24     |1 c2.. d''8
25     |||                               % final bar
26   }
27
28 % } % helloWorld

```

68.5 First example output from the MSDL converter

Compiling HelloWorld.msd1 to LilyPond, we get the output below.

68.5.1 LilyPond output

```

1 \version "2.24.0"
2
3 % Comment or adapt next line as needed (default is 20)
4 #(set-global-staff-size 20 )
5
6 % Pick your choice from the next two lines as needed
7 %myBreak = { \break }
8 myBreak = {}
9
10 % Pick your choice from the next two lines as needed
11 %myPageBreak = { \pageBreak }
12 myPageBreak = {}
13
14 \header {
15   title               = ""
16   encodingDate        = "Sunday 2021-05-30 @ 12:11:50 CEST"
17   software             = "MSDL converter 1.0"
18 }
19
20 \paper {
21 }
22
23 \layout {
24   \context {
25     \Score
26     autoBeaming = ##f % to display tuplets brackets
27   }
28   \context {
29     \Voice
30   }
31 }
32
33 Part_Part_One_Staff_One_Voice_One = \absolute {
34   \language "nederlands"
35   c2.. d''8 }
36
37 \book {
38   \score {
39     <<
40
41     \new Staff = "Part_Part_One_Staff_One"
42     \with {
43       }
44     <<

```

```

45     \context Voice = "Part_Part_One_Staff_One_Voice_One" <<
46     \Part_Part_One_Staff_One_Voice_One
47     >>
48 >>
49
50 >>
51
52 \layout {
53     \context {
54         \Score
55         autoBeaming = ##f % to display tuplets brackets
56     }
57     \context {
58         \Voice
59     }
60 }
61
62 \midi {
63     \tempo 4 = 90
64 }
65 }
66
67 }

```

68.5.2 Braille output

With:

```
1 msdl -braille HelloWorld.msd1 -use-encoding-in-file-name -braille-output-kind utf8d
```

we get in file HelloWorld.msd1_UTF8Debug.brf Braille 6-dots cells, which can be displayed in a suitable editor as:

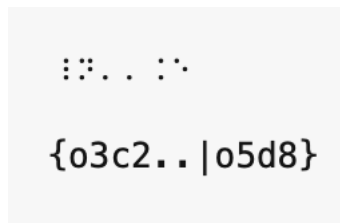


Figure 68.1: Braille for HelloWorld.xml with interpretation

The interpretation shows a textual view of the contents of the previous line. `obj*` indicates the octave number.

68.5.3 MusicXML output

Compiling HelloWorld.msd1 to MusicXML, we get:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
3     "http://www.musicxml.org/dtds/partwise.dtd">
4 <score-partwise version="3.1">
5     <!-- ===== Created by msdl 0.02 on Sunday 2021-05-30 @ 12:15:44 CEST from HelloWorld.
6     msdl ===== -->
7     <work>
8         <work-number/>

```

```

8      <work-title/>
9  </work>
10 <movement-number/>
11 <movement-title/>
12 <identification>
13   <encoding>
14     <software>msdl 0.02, https://github.com/jacques-menu/musicformats</software>
15     <encoding-date>2021-05-30</encoding-date>
16   </encoding>
17   <miscellaneous>
18     <miscellaneous-field name="description"/>
19   </miscellaneous>
20 </identification>
21 <part-std::list>
22   <score-part id="Part_One">
23     <part-name/>
24     <score-instrument id="Part_OneI1">
25       <instrument-name/>
26     </score-instrument>
27   </score-part>
28 </part-std::list>
29 <part id="Part_One">
30   <measure number="1">
31     <attributes>
32       <divisions>2</divisions>
33     </attributes>
34     <note>
35       <pitch>
36         <step>C</step>
37         <octave>3</octave>
38       </pitch>
39       <duration>7</duration>
40       <voice>1</voice>
41       <type>half</type>
42       <dot/>
43       <dot/>
44       <staff>1</staff>
45     </note>
46     <note>
47       <pitch>
48         <step>D</step>
49         <octave>5</octave>
50       </pitch>
51       <duration>1</duration>
52       <voice>1</voice>
53       <type>eighth</type>
54       <staff>1</staff>
55     </note>
56   </measure>
57 </part>
58 </score-partwise>

```

68.5.4 Guido output

Compiling HelloWorld.msd1 to Guido, we get:

```

1  {[ \staff<1> \set<autoHideTiedAccidentals="on"> \title<"> \barFormat<style= "system",
   range="1"> \bar<hidden="true"> \beamsOff c0/2.. \beamsOff d2/8 ]
2  }

```

68.6 A more realistic example

Thanks to Jean Abou-Samra for providing UnPetitAir.msd1:

```

1  %{
2    An explicit and implicit voices piano score
3  %}
4
5
6  % l'identification
7  % -----
8
9  titre      "Un petit air"
10 compositeur "Jean Abou Samra"
11
12
13 % la langue pour les hauteurs de notes
14 % -----
15
16 hauteurs francais          % par défaut: english
17
18
19 % la partition
20 % -----
21
22 musique unPetitAir =
23 {
24   |1  clef treble
25       key c
26       time 9/8
27       r4. a,4-> <e g bf>8~ <e g bf>4.~
28
29   |2  <e g bf>4. r2.
30
31   % Maintenant, je reviens en arrière pour la voix supérieure.
32   |2  fs''16 gs'' fs''8 cs'' ds'' e'' b' d'' a' e'
33
34   % La voix inférieure s'éteint.
35   |3  c''8 gs' d' c' fs' a' b' gs' b
36   |4  a'8 e' a g as gs' d'( a ds)
37   |5  e8( b g d' a' e'' b'' c'' b''
38   |6  e''4.) % Rien à la fin.
39
40   % Je décide d'ajouter une tenue de la basse.
41   |5  e2.~ e4.
42
43   % J'ajoute encore une voix. Au passage, je change la métrique.
44   |6  time 6/8
45   |6  r8 e'( f') e' c'' d''
46
47   % Et encore un changement de métrique.
48   |7  time 4/4
49   |7  e''1~
50
51   % Je finis la phrase.
52   |7  e''4 e' d''8 c'' b' a'
53   |8  b'1
54
55   % Je retourne sur mes pas pour introduire l'ostinato.
56   |7  r8 e8 f e f e c' a
57   |8  r8 e8 f e c' a e f
58   |9  r8 ds e ds e ds b fs
59
60   % etc.
61 }
```

Jean also provided the output created by hand with LilyPond, see figure 68.2 [Un Petit Air, par Jean Abou-Samra], page 406:

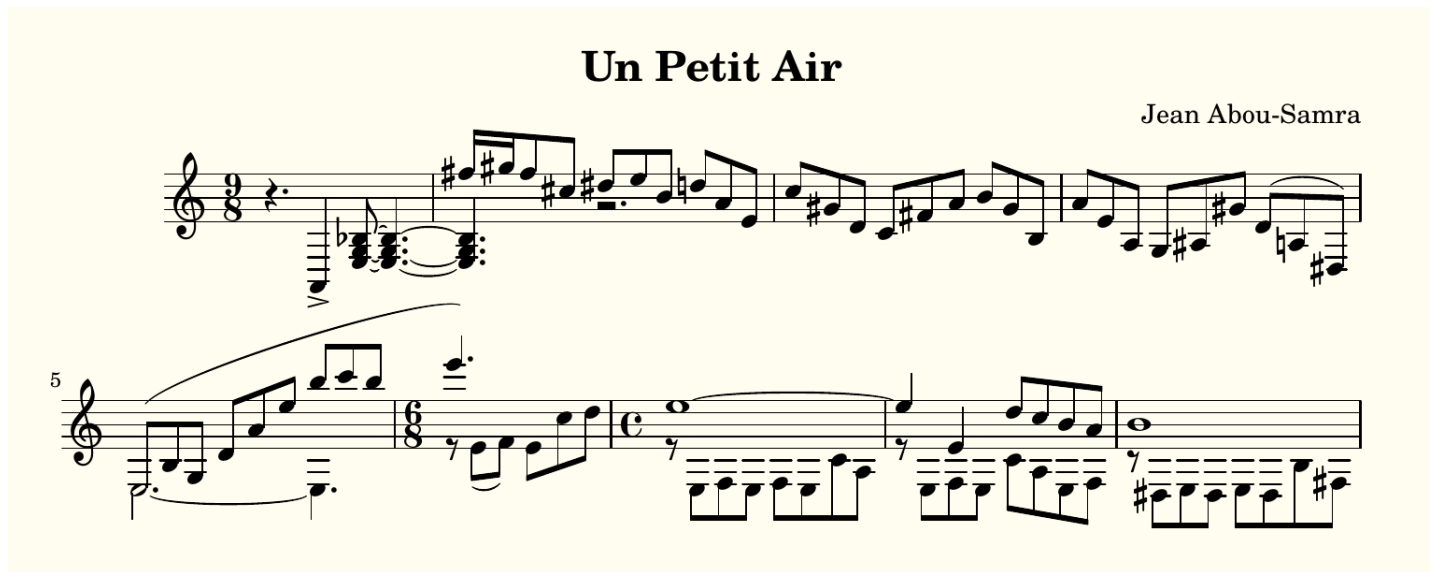


Figure 68.2: Un Petit Air, par Jean Abou-Samra

68.7 Multi-language support

68.7.1 Multi-language messages handling

68.7.2 Multi-language keywords handling

68.8 Lexical analysis

68.9 Music Scores Descriptions Representation (MSDR)

68.10 Syntax and semantic analysis

The language-dependent keywords leads to a recursive descent parser, since `flex`-generated scanners need 'fixed' keyword in the language description.

68.10.1 Error recovery

The MSDDL converter uses a variant of the *stopper sets* method that was present in the early Pascal and Pascal-S converters. The latter passed a set of tokens not to be overtaken to the procedures in charge of accepting the various statements in the language. Strangely enough, this was not done for declarations.

We use a stack of tokens sets that grows and shrinks in parallel with the accepting functions, to know more contextual informations when deciding wether to consume a token or not. The corresponding term is it shift when building the analysis tables in LR technology.

Part XVI

Debugging

Chapter 69

Debugging

Debugging MusicFormats can be quite time-consuming. The trace options available have been designed to provide fine-grained trace information to help locate issues.

Functions `catchSignals ()` at the beginning of the `main ()` function can be commented out in order to run a service under a debugger.

File `src/mflibrarymfPreprocessorSettings.h` contains:

```
1  /*
2   MusicFormats Library
3   Copyright (C) Jacques Menu 2016-2023
4
5   This Source Code Form is subject to the terms of the Mozilla Public
6   License, v. 2.0. If a copy of the MPL was not distributed with this
7   file, you can obtain one at http://mozilla.org/MPL/2.0/.
8
9   https://github.com/jacques-menu/musicformats
10  */
11
12  #ifndef __mfPreprocessorSettings__
13  #define __mfPreprocessorSettings__
14
15  /*
16   This file groups several build-time setting that influence
17   code security and speed, as well as a couple of facilities
18  */
19
20  // -----
21  // uncomment the following definition if no exceptions display is desired
22  #define MF_CAUGHT_EXCEPTIONS_DISPLAY_IS_ENABLED
23
24  // -----
25  // comment the following definition if abort on internal errors is desired
26  // CAUTION: DON'T USE THIS IN PRODUCTION CODE,
27  // since that could kill a session on a web server, for example
28  #define MF_ABORT_TO_DEBUG_ERRORS_IS_ENABLED
29
30  // -----
31  // uncomment the following definition if abort on internal errors is desired
32  #define MF_SANITY_CHECKS_ARE_ENABLED
33
34  // -----
35  // uncomment the following definition if harmonies extra options are desired
36  #define MF_HARMONIES_EXTRA_IS_ENABLED
37
38  #include "oahHarmoniesExtraOah.h"
39
40  // -----
```

```

41 // comment the following definition if no trace is desired
42 #define MF_TRACE_IS_ENABLED
43
44 #ifdef MF_TRACE_IS_ENABLED
45     #include "mfTrace0ah.h"
46 #endif // MF_TRACE_IS_ENABLED
47
48
49 #endif // __mfPreprocessorSettings__

```

69.1 Useful options

Here are the most basing options used when debugging:

- option `-trace-passes`, `-tpasses` this is the first option to use, to locate in which pass the problem arises
- option `-input-line-numbers`, `-iln` this option produces the music elements input-line numbers in the output files
- the `-display*` options

69.2 Removing the results of a build

The contents of `distrib/` after a build is:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > ll build/
2 total 80
3  0 drwxr-xr-x  10 jacquesmenu  staff      320 May  2 18:18:50 2022 ./
4  0 drwxr-xr-x  39 jacquesmenu  staff     1248 May  2 18:18:51 2022 ../
5 16 -rw-r--r--@  1 jacquesmenu  staff     6148 May  2 15:45:10 2022 .DS_Store
6  8 -rw-r--r--@  1 jacquesmenu  staff     1815 Jun 29 09:10:50 2021 Building.md
7 32 -rw-r--r--  1 jacquesmenu  staff    14849 May  2 18:18:50 2022 CMakeLists.txt
8  8 -rw-r--r--  1 jacquesmenu  staff       291 Jun 29 09:10:50 2021 MakePkg.bat
9 16 -rw-r--r--  1 jacquesmenu  staff     7463 May  2 18:18:50 2022 Makefile
10 0 drwxr-xr-x@ 27 jacquesmenu  staff       864 May  2 18:21:17 2022 bin/
11 0 drwxr-xr-x@  7 jacquesmenu  staff       224 May  2 18:21:11 2022 lib/
12 0 drwxr-xr-x 12 jacquesmenu  staff       384 May  2 18:19:25 2022 libdir/

```

The built files are in `distrib/bin`, `distrib/lib` and `distrib/libdir`. There is no `clean` target in Makefile. They can be removed in a single step with this alias:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > type rmbuild
2 rmbuild is aliased to 'cd ${MUSIC_FORMATS_DEV}/build ; rm -r bin lib libdir; ls -sal'

```

69.3 Reverting to a previous MusicFormats version

The GitHub MusicFormats repository keeps a number of recent releases, such as:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > git branch
2 * master
3   v0.9.60
4   v0.9.61
5   v0.9.62
```

Then `master` branch contains the development version. To switch back to another version, one should check it out:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > git checkout v0.9.61
2 Switched to branch 'v0.9.61'
3 Your branch is up to date with 'master/v0.9.61'.
4 jacquesmenu@macmini: ~/musicformats-git-dev > git branch
5   master
6   v0.9.60
7 * v0.9.61
8   v0.9.62
```

Now building this version with `make`, we get:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/build > xml2ly -version
2 Command line version of musicxml2lilypond converter v0.9.61 (March 3, 2022)
```

Chapter 70

Locating a bug with Git's bisection

A bug appeared in v0.9.63, in which the `-display-lpsr-short`, `-dlpsrshort` option causes `xml2ly` to crash. The symptom varies with the operating system, pointing to a probable memory corruption.

Reverting to MusicFormats previous versions show that:

- v0.9.60 behaves alright;
- v0.9.61 exhibits the bug:

```
1 jacquesmenu@macmini: ~/Desktop > lldb -- xml2ly fullbarrests/  
  MeasureRestWithoutBarLine.xml -display-lpsr-short -aofn  
2 (lldb) target create "xml2ly"  
3 Current executable set to 'xml2ly' (x86_64).  
4 (lldb) settings set -- target.run-args "fullbarrests/MeasureRestWithoutBarLine.xml"  
  "-display-lpsr" "-aofn"  
5 (lldb) r  
6 Process 28676 launched: '/Users/jacquesmenu/musicformats-git-dev/build/bin/xml2ly' (  
  x86_64)  
7      The measure with ordinal number 3 is now registered with a duration of 1/1 in  
  part Part_P0ne (partID "P1", partName "Soprano"), fPartMeasuresWholeNotessVector  
  .size () = 2  
8      The measure with ordinal number 4 is now registered with a duration of 1/1 in  
  part Part_P0ne (partID "P1", partName "Soprano"), fPartMeasuresWholeNotessVector  
  .size () = 2  
9 %-----  
10 Pass (ptional): displaying the LPSR as text  
11 %-----  
12  
13 ... ..  
14  
15 [PartGroup "PartGroup_1 ('0', fPartGroupName "Implicit")" (1 part), line 0  
16   fPartGroupName      : "Implicit"  
17  
18 Process 28676 stopped  
19 * thread #1, queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS (code=1,  
  address=0x1)  
20   frame #0: 0x00000001007b879a xml2ly`MusicFormats::msrPartGroup::print (this=0  
  x0000600003e08100, os=0x000000010ca04650) const at msrPartGroups.cpp:1185:13  
21   1182         i         = iBegin;  
22   1183  
23   1184     for ( ; ; ) {  
24 -> 1185         os << (*i);  
25   1186         if (++i == iEnd) break;  
26   1187         os << std::endl;  
27   1188     } // for  
28 Target 0: (xml2ly) stopped.  
29 (lldb)
```

This bug has thus been introduced between v0.9.60 and v0.9.61. There have been several `git push` occurrences leading from v0.9.60 to v0.9.61.

70.1 Locating a bug at random in the Git log

git provides various ways to display the commits history of the repository through `git log` options, for example:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > git log --pretty=format:"%h - %ad : %s"
2 ea338fd - Tue May 3 10:09:04 2022 +0200 : Complement to the Makefile
3 083db8a - Tue May 3 07:43:09 2022 +0200 : Before reverting to v0.9.60
4 12b6d93 - Mon May 2 09:41:37 2022 +0200 : Prior to bisecting
5 03d98be - Tue Apr 26 11:15:23 2022 +0200 : Finalized sone trace options
6 3dd7b72 - Tue Apr 26 10:10:38 2022 +0200 : Finalized sone trace options
7 7f7507c - Thu Apr 14 17:01:14 2022 +0200 : Finalized MFSL syntax and semantics, fixed a
   couple of issues
8 06109d3 - Thu Apr 14 17:00:50 2022 +0200 : Finalized MFSL syntax and semantics, fixed a
   couple of issues
9 62aa64c - Thu Apr 7 07:18:44 2022 +0200 : v0.9.62
10 671ffa4 - Thu Apr 7 06:26:34 2022 +0200 : Pre v0.9.62
11 bf9eb63 - Wed Apr 6 23:42:44 2022 +0200 : Pre v0.9.62
12 db4397c - Wed Apr 6 22:14:43 2022 +0200 : Pre v0.9.62
13 9a80b24 - Mon Apr 4 13:06:12 2022 +0200 : Added MFSL (MusicFormats Script Language)
14 2ef1150 - Mon Apr 4 12:07:07 2022 +0200 : Added MFSL (MusicFormats Script Language)
15 3f56d52 - Tue Mar 29 16:34:23 2022 +0200 : Added MFSL (MusicFormats Script Language)
16 fc1ea21 - Tue Mar 29 08:58:34 2022 +0200 : Added MFSL (MusicFormats Script Language)
17 737b996 - Mon Mar 28 23:42:03 2022 +0200 : Added MFSL (MusicFormats Script Language)
18 8c91155 - Sat Mar 26 08:35:55 2022 +0100 : Added MFSL (MusicFormats Script Language)
19 fc68a93 - Fri Mar 18 15:11:19 2022 +0100 : Added MFSL (MusicFormats Script Language)
20 01430a9 - Fri Mar 18 15:11:12 2022 +0100 : Added MFSL (MusicFormats Script Language)
21 4082813 - Thu Mar 17 18:50:02 2022 +0100 : Added MFSL (MusicFormats Script Language)
22 2696628 - Sun Mar 13 00:48:05 2022 +0100 : Added MFSL (MusicFormats Script Language)
23 a828231 - Thu Mar 10 14:28:11 2022 +0100 : Added MFSL (MusicFormats Script Language)
24 bf04937 - Wed Mar 9 12:53:17 2022 +0100 : Added MFSL (MusicFormats Script Language)
25 a855ee4 - Tue Mar 8 16:39:28 2022 +0100 : Added MFSL (MusicFormats Script Language)
26 b636816 - Mon Mar 7 14:49:54 2022 +0100 : Added MFSL (MusicFormats Script Language)
27 ecd5eaa - Sun Mar 6 00:11:05 2022 +0100 : Added 'keep-msr-voice, kmv' option
28 ec1c8ef - Sat Mar 5 08:48:39 2022 +0100 : Added 'ignore-msr-voice, imv' option
29 8246467 - Thu Mar 3 16:11:37 2022 +0100 : v0.9.61
30 603e19c - Thu Mar 3 13:43:48 2022 +0100 : Switched from C++11 to C++17 for <filesystem>
31 77d3d29 - Thu Mar 3 07:56:00 2022 +0100 : Switched from C++11 to C++17 for <filesystem>
32 a880063 - Thu Mar 3 07:44:08 2022 +0100 : Switched from C++11 to C++17 for <filesystem>
33 38b584f - Wed Mar 2 12:44:22 2022 +0100 : Switched from C++11 to C++17 for <filesystem>
34 662454a - Tue Mar 1 17:14:47 2022 +0100 : Pre-v0.9.61
35 2cb4d5f - Mon Feb 28 11:53:04 2022 +0100 : Renamed some documentation folders and files
36 c7839a8 - Mon Feb 28 09:56:46 2022 +0100 : Renamed some documentation folders and files
37 0e85f99 - Mon Feb 28 09:06:16 2022 +0100 : Renames some documentation folders and files
38 c5a43d9 - Fri Feb 25 17:48:29 2022 +0100 : Finalized files/musicxmlfiles/Makefile
39 21e3898 - Thu Feb 24 21:54:28 2022 +0100 : Added '-replicate-msr-measure' option
40 9738598 - Thu Feb 24 21:53:00 2022 +0100 : Added '-replicate-msr-measure' option
41 ae751c3 - Mon Feb 21 09:58:35 2022 +0100 : Added various options
42 f2d2f57 - Sat Feb 19 08:09:02 2022 +0100 : Workflow to publish Mac OS release
43 ac5ad6b - Sat Feb 19 08:00:57 2022 +0100 : Initializa npm package
44 29de34d - Fri Feb 18 11:00:42 2022 +0100 : v0.9.60
45 7c067d6 - Fri Feb 18 10:56:17 2022 +0100 : v0.9.60
46 5e3ba90 - Fri Feb 18 09:57:50 2022 +0100 : Pre v0.9.60
47 dfef7be - Fri Feb 18 09:56:07 2022 +0100 : Pre v0.9.60
48 c31dde3 - Wed Feb 16 11:50:42 2022 +0100 : Updates to the make and cmake configuration
49 fd6fef0 - Wed Feb 16 09:45:44 2022 +0100 : Complements to the installation doc
50 b7ad2af - Tue Feb 15 17:40:53 2022 +0100 : Distrib test 17
51 50a904c - Tue Feb 15 17:37:53 2022 +0100 : Distrib test 16
52 cf65bd3 - Tue Feb 15 08:41:14 2022 +0100 : Distrib test 15
53 9cda15e - Tue Feb 15 08:38:57 2022 +0100 : Distrib test 14
54 74a2b7f - Tue Feb 15 08:30:21 2022 +0100 : Distrib test 13
```

```

55 ee011e9 - Mon Feb 14 15:06:59 2022 +0100 : Distrib test 13
56 4d6f9cb - Mon Feb 14 08:54:23 2022 +0100 : Distrib test 12
57 7692b7b - Mon Feb 14 08:52:50 2022 +0100 : Distrib test 12
58 d9e943d - Sat Feb 12 09:30:45 2022 +0100 : Distrib test 11
59 3dde810 - Sat Feb 12 09:29:51 2022 +0100 : Distrib test 11
60 f07b02a - Fri Feb 11 17:55:34 2022 +0100 : Distrib test 10
61 5113824 - Fri Feb 11 16:47:17 2022 +0100 : Distrib test 9
62 7f0fa8e - Fri Feb 11 16:45:21 2022 +0100 : Distrib test 8
63 93f72f4 - Fri Feb 11 16:32:33 2022 +0100 : Distrib test 6
64 121aa64 - Fri Feb 11 14:35:38 2022 +0100 : Distrib test 5
65 547556f - Fri Feb 11 11:30:17 2022 +0100 : Creation of MusicFormats repository

```

One can pick one of the commits, revert to it and check whether the bug is present in it.

70.2 Locating a bug in the commits with Git's bisection

Locating the particular push that introduced the bug can be facilitated by git's *bisect* facility. Here is how it works:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect start
2
3 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect good v0.9.60
4
5 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect bad v0.9.61
6 Bisecting: 7 revisions left to test after this (roughly 3 steps)
7 [0e85f994ab00ea2dd94ddcb1895cbae5a32f072a] Renames some documentation folders and files
8
9 jacquesmenu@macmini: ~/musicformats-git-dev > git branch
10 * (no branch, bisect started on v0.9.61)
11 master
12 v0.9.60
13 v0.9.61
14 v0.9.62

```

The bisection proposes commit 0e85f994ab00ea2dd94ddcb1895cbae5a32f072a as a middle point between v0.9.60 and v0.9.61. So let us check it out:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git checkout 0
   e85f994ab00ea2dd94ddcb1895cbae5a32f072a
2 HEAD is now at 0e85f99 Renames some documentation folders and files
3
4 jacquesmenu@macmini: ~/musicformats-git-dev > git branch
5 * (no branch, bisect started on v0.9.61)
6 master
7 v0.9.60
8 v0.9.61
9 v0.9.62

```

Then, building this intermediate development version leads to:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/build > make
2 make macos
3 cd libdir && cmake .. -G Xcode -Wno-dev
4 -- VERSION: v0.9.61
5
6 -- Configuring version v0
7 .v9
8 .v61
9
10 ... ..
11

```

```

12 /Users/jacquesmenu/musicformats-git-dev/src/components/mfcLibraryComponent.cpp:12:10:
    fatal error:
13   ' ../../../../MusicFormatsVersionNumber.h' file not found
14 #include " ../../../../MusicFormatsVersionNumber.h"
15   ~~~~~
16 1 error generated.
17
18 ... ..
19
20 ** BUILD FAILED **

```

Well, this dev version had been pushed to have new files and/or contents saved on the MusicFormats repository ...and we should try other commits around it.

A first possibility is to use `git bisect skip`, that moves to :

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect skip
2 Bisecting: 7 revisions left to test after this (roughly 3 steps)
3 [c7839a87549660963a8b1ef0898d5cbcce8305aa] Renamed some documentation folders and files

```

Checking commit `c7839a87549660963a8b1ef0898d5cbcce8305aa` out, we get:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git checkout
   c7839a87549660963a8b1ef0898d5cbcce8305aa
2 HEAD is now at c7839a8 Renamed some documentation folders and files

```

Building that leads to the same error as above. Let us skip one again:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect skip
2 Bisecting: 7 revisions left to test after this (roughly 3 steps)
3 [a880063c134a7ba49b31f5fb52b47f682058f64a] Switched from C++11 to C++17 for <filesystem>

```

It turns out the commit `a880063c134a7ba49b31f5fb52b47f682058f64a` does not build either. Let us skip to the next commit:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git bisect skip
2 Bisecting: 7 revisions left to test after this (roughly 3 steps)
3 [2cb4d5f2133d34a19fea8f47d9ed2ccfb24d0fad] Renamed some documentation folders and files
4
5 jacquesmenu@macmini: ~/musicformats-git-dev > git checkout 2
   cb4d5f2133d34a19fea8f47d9ed2ccfb24d0fad
6 HEAD is now at 2cb4d5f Renamed some documentation folders and files

```

Here the code base builds alright, and bug does not show up, so we should continue skipping.

70.3 Locating the bug in the code base

The bug we're after is found to have been introduced at this point:

- commit `2cb4d5f`, does not exhibit the bug;
- the next one, commit `662454a`, does.

The changes brought by commit can be shown with:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > git log --patch -1 662454a > patch_662454a.
   txt
2
3 jacquesmenu@macmini: ~/musicformats-git-dev > ls -sal patch_662454a.txt
4 256 -rw-r--r--@ 1 jacquesmenu staff 80782 May  4 10:59 patch_662454a.txt

```

The bug shows up with `files/musicxmlfiles/fullbarrests/MeasureRestWithoutBarLine.xml`, but not with others such as `files/musicxmlfiles/multistaff/SATBExample.xml`.

Analysing the patch description in `patch_662454a.txt`, we find that nothing in the differences between those two successive patches can explain the crash. The problem thus lies elsewhere...

Part XVII

Indexes