MusicFormats maintainance guide

v0.9.61 – Feb 27, 2022

Jacques Menu

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

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

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

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

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

Abstract

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

Contents

Contents

Contents

Contents

Contents

List of Figures

Listings

Part I

Overview of MusicFormats

Chapter 1

Introduction

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

All the MusicXML examples mentioned can be downloaded from `https://github.com/jacques-menu/musicformats/tree/master/files/musicxml`.
They are grouped by subject in subdirectories, such as `basic/HelloWorld.xml`.

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

1.1 Acknowledgements

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

1.2 Prerequisites

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

- obtain a working knowledge of C++ programming. The code base of MusicFormats uses classes, simple inheritance, and templates;

- study the architecture of MusicFormats, which can be seen in figure 1.1, [Architecture], page 18, and is presented in more detail at:
  `https://github.com/jacques-menu/musicformats/blob/master/doc/musicformatsArchitecture/musicformatsArchitecture.pdf`

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

1.3   Chronology

Dominique Fober created `libmusicxml2` long before this author had the need for a library to read MusicXML data, in order to convert it to LilyPond. In the picture showing the architecture of MusicFormats in figure 1.1, [Architecture], page 18, Dom's work is essentially represented by the MusicXML, MXSRand Guido boxes at the top. He did more than this, of course, to provide `libmusicxml2` to users!

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

- perform as least as well as `musicxml2ly`, provided by LilyPond;

- provide as many options as needed to meet the user's needs.

The `*.cpp` files in `samples` were examples of the use of the library. Among them, `xml2guido` has been used since in various contexts. The diagram in figure 1.1, [Architecture], page 18, was created afterwards, and it would then have consisted of only MusicXML, MXSR and Guido, with passes 1, 2 and 3.

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

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

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

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

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

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

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

1.4   The GitHub repository

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

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

- the `v....` versions are the master versions frozen at some point in time.

Figure 1.1: The MusicFormats architecture
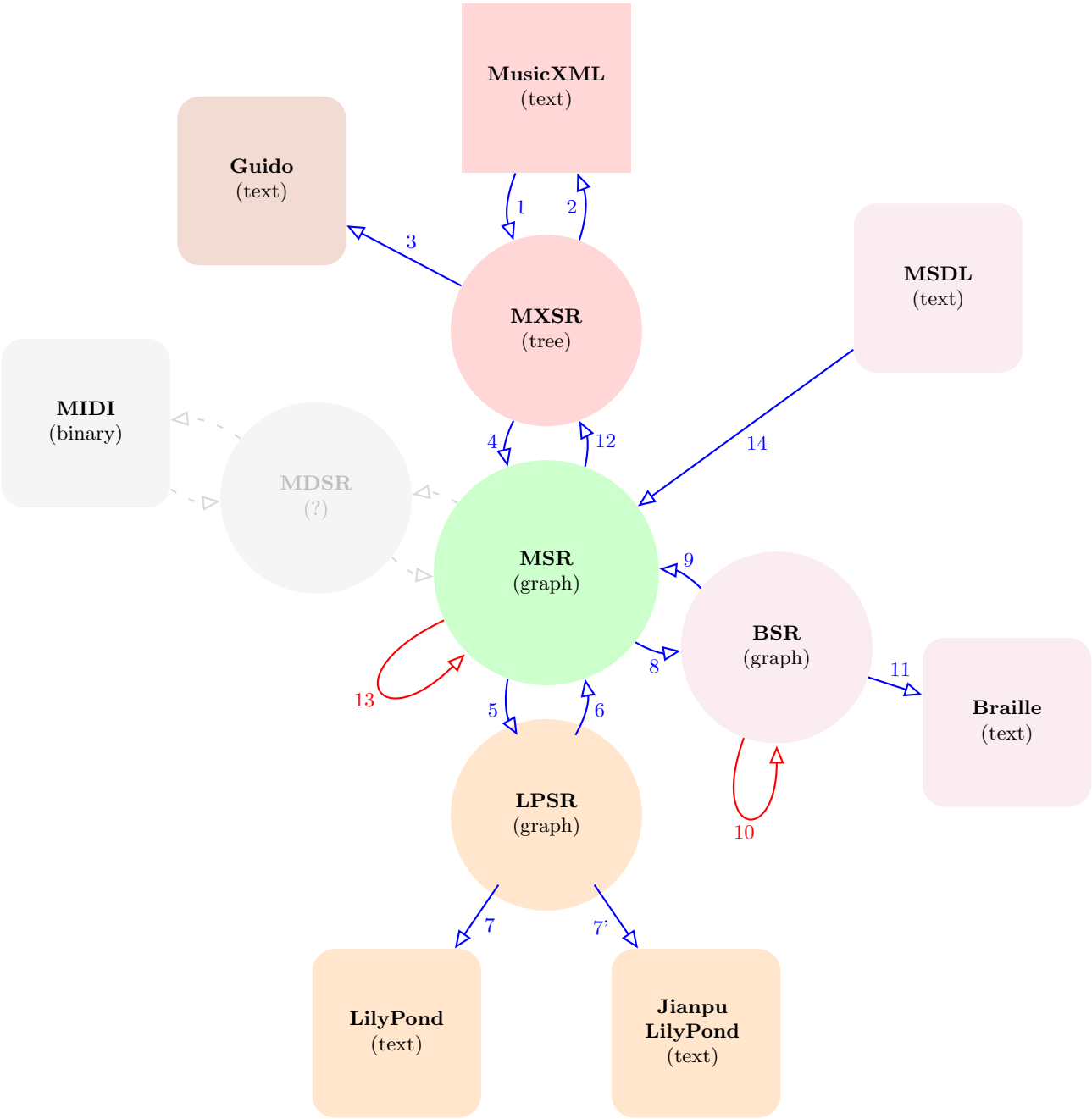
Chapter 2

Documentation

The MusicFormats documentation is written in LaTeX and contains:

- `documentation/MusicFormatsLaTeXSettings.tex`: a set of commands and settings used by all the documentation LaTeX files;

- `documentation/introductionToMusicxml/`: 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);

- `documentation/presentation/`: Dom Fober's original documentation describing `libmusicxml2`;

- `documentation/MusicFormatsArchitecture.tex`: the image describing the architecture of MusicFormats, used in other documents;

- `documentation/MusicFormatsUserGuide/`: the user's guide;

- `documentation/maintainersGuideToMusicFormats/`: this maintiainer's guide.

The pictures are created with the Ti*k*Z/PGF package, see `https://www.bu.edu/math/files/2013/08/tikzpgfmanual.pdf`.

LaTeX may need 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 file `documentation/MusicFormatsVersionNumber.txt` is used by the LaTeX source files to show the version number. It should be updated when a new version of MusicFormats is created.

2.1 LaTeX macros

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

For example:

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

```
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
       ]{#1.h/.cpp@{{tt *#1.h/.cpp}}}%
6  }
```

```
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
   ]{#1.h/.cpp@{{tt *#1.h/.cpp}}}%
13 }
```

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 is uses;

- then the two-phase visitors pattern, which is central to MusicFormats, is presented;

- the third part is dedicated to the programming style and conventions used throughout the code base;

- the OAH (Options and help), a pervasive feature in MusicFormats, is detailed;

- the fifth part details the formats provided by the library;

- the following parts are dedicated to passes, generators and converters, respectively;

- the ninth part presents the interfaces to the formats, passes and converters;

- the tenth part provides a longitudinal view of the handling of selected music score contents elements, grouped by such elements such as staves, tuplets and harmonies;

- and finally, the last part is dedicated to the implementation of the MSDL language.

2.3 The MusicFormats architecture

2.4 Command line user's guide

documentation/MusicFormatsUserGuide/MusicFormatsUserGuide.pdf is the usual user guide. It presents the command line use of MusicFormats.

2.5 API user's guide

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

2.6 Maintainance guide

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

Chapter 3

Building MusicFormats

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

- a C++11 compiler;

- the `cmake` tool.

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

The C++11 language is needed because MusicFormats uses $<$`regex`$>$ and the `auto` keyword. More recent versions should not be a problem.

3.1   Cloning the repository

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

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

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

3.2   One-shot partial `make` runs

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

3.3   `cmake` configuration

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

3.4 Library `Makefile`

This `Makefile` is build/Makefile.

3.5 CLI amples `Makefile`

This `Makefile` is src/clisamples/Makefile.

3.6 Building MusicFormats in practise

Once in the local repository clone, just execute:

```
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
       Mikrokosmos3Wandering*
7   8504 -rwxr-xr-x   1 jacquesmenu   staff   4350480 Sep 27 00:04:49 2021 MusicAndHarmonies*
8   8504 -rwxr-xr-x   1 jacquesmenu   staff   4350464 Sep 27 00:05:00 2021 RandomChords*
9   8504 -rwxr-xr-x   1 jacquesmenu   staff   4350448 Sep 27 00:05:01 2021 RandomMusic*
10  8696 -rwxr-xr-x   1 jacquesmenu   staff   4450928 Sep 27 00:04:56 2021 countnotes*
11 63904 -rwxr-xr-x   1 jacquesmenu   staff  32717248 Sep 27 00:04:57 2021
       libMultipleInitsTest*
12 76696 -rwxr-xr-x   1 jacquesmenu   staff  39266928 Sep 27 00:05:01 2021 msdlconverter*
13   144 -rwxr-xr-x   1 jacquesmenu   staff     70480 Sep 27 00:04:55 2021 musicformatsversion
       *
14 12616 -rwxr-xr-x   1 jacquesmenu   staff   6455376 Sep 27 00:04:59 2021 partsummary*
15  8920 -rwxr-xr-x   1 jacquesmenu   staff   4564864 Sep 27 00:04:59 2021 readunrolled*
16 81048 -rwxr-xr-x   1 jacquesmenu   staff  41496208 Sep 27 00:04:49 2021 xml2Any*
17 61232 -rwxr-xr-x   1 jacquesmenu   staff  31347456 Sep 27 00:04:53 2021 xml2brl*
18 63704 -rwxr-xr-x   1 jacquesmenu   staff  32615072 Sep 27 00:04:47 2021 xml2gmn*
19 17368 -rwxr-xr-x   1 jacquesmenu   staff   8891744 Sep 27 00:04:56 2021 xml2guido*
20 63896 -rwxr-xr-x   1 jacquesmenu   staff  32713936 Sep 27 00:04:50 2021 xml2ly*
21 12512 -rwxr-xr-x   1 jacquesmenu   staff   6403968 Sep 27 00:04:55 2021 xml2midi*
22 56384 -rwxr-xr-x   1 jacquesmenu   staff  28865024 Sep 27 00:04:59 2021 xml2xml*
23  9176 -rwxr-xr-x   1 jacquesmenu   staff   4695472 Sep 27 00:04:55 2021 xmlclone*
24  9320 -rwxr-xr-x   1 jacquesmenu   staff   4771024 Sep 27 00:05:00 2021 xmlfactory*
25  8912 -rwxr-xr-x   1 jacquesmenu   staff   4559072 Sep 27 00:04:57 2021 xmliter*
26  8752 -rwxr-xr-x   1 jacquesmenu   staff   4478336 Sep 27 00:04:55 2021 xmlread*
27 12104 -rwxr-xr-x   1 jacquesmenu   staff   6193216 Sep 27 00:04:54 2021 xmltranspose*
28  9320 -rwxr-xr-x   1 jacquesmenu   staff   4770128 Sep 27 00:05:02 2021 xmlversion*
```

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

```
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
       .3.2.0.dylib*
6      0 lrwxr-xr-x   1 jacquesmenu   staff        24 Sep 27 00:04:45 2021 libmusicxml2.3.
       dylib@ -> libmusicxml2.3.2.0.dylib
7  1055040 -rw-r--r--   1 jacquesmenu   staff 532838416 Sep 27 00:04:41 2021 libmusicxml2.a
```

```
 8    591776 -rw-r--r--    1 jacquesmenu   staff   302989312 Sep 21 09:05:55 2021 libmusicxml2.a.
         A93i4n
 9     57056 -rw-r--r--    1 jacquesmenu   staff    29212672 Sep 21 09:01:27 2021 libmusicxml2.a.
         KHrJT0
10     39968 -rw-r--r--    1 jacquesmenu   staff    20463616 Sep 21 09:11:20 2021 libmusicxml2.a.
         gZfmqe
11     39968 -rw-r--r--    1 jacquesmenu   staff    20463616 Sep 21 09:09:22 2021 libmusicxml2.a.
         tndUAV
12         0 lrwxr-xr-x    1 jacquesmenu   staff          20 Sep 27 00:04:45 2021 libmusicxml2.
         dylib@ -> libmusicxml2.3.dylib
```

Chapter 4

Code base structure

The code base of the MusicFormats library contains:

- **build** : a set of files to build the library in various environments with `make`

- **doc** : the documentation in LaTeX source and PDF formats

- **files** : a set of sample files for MusicXML and MSDL

- **javascript** : a set of files for the generation of Java Script, to allow the use of MusicFormats in Web sites

- **libmusicxml** : an embedded copy of the `libmusicxml2` code base

- **packages** : a set of files to create installable versions of the library, not yet operational

- **samples** : the main programs for examples supplide with MusicFormats, such as generators and converters

- **schemas** : a set of files defining the input languages, currently MusicXML, BMML and MEI, together with scripts to generated the set of classes definitions for analyzing them

- **src** the library code base, detailed below

- **validation** : a set of files including a `Makefile` for the validation of the library using the contents of `files`

- **win32** : Windows related support

4.1 The `libmusicxml` folder

This folder contains a version of Grame's `libmusicxml2` library, used by MusicFormats, to avoid the need for installing it separately. The only possible annoyance when installing both libraries is that the executables in `libmusicxml/build/bin` such as `countnotes` and `xml2guido` are installed twice: choosing which one to use can be handled in the `${PATH}` environment variable or its equivalent.

4.2   The doc folder

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

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

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

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

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

```
jacquesmenu@macmini: ~/musicformats-git-dev/documentation/presentation > ll
total 504
  0 drwxr-xr-x  6 jacquesmenu   staff     192 Apr 22 15:48:43 2021 ./
  0 drwxr-xr-x  9 jacquesmenu   staff     288 Jan  4 17:23:41 2022 ../
  0 drwxr-xr-x  6 jacquesmenu   staff     192 Apr 22 15:48:43 2021 imgs/
 88 -rw-r--r--  1 jacquesmenu   staff   41534 Apr 22 15:48:43 2021 libmusicxml2.odg
392 -rw-r--r--  1 jacquesmenu   staff  200524 Apr 22 15:48:43 2021 libmusicxml2.pdf
 24 -rw-r--r--@ 1 jacquesmenu   staff   11003 Oct 15 18:48:11 2021 libmusicxml2.tex
```

4.3   The schemas folder

This folder contains the definitions used to create the classes definitions to analyze textual data in the MusicXML, MEIand BMMLforat, and `elements.bash` scripts that compile the definitions into the C++ code files containing the classes:

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

```
 8     8 -rw-r--r--   1 jacquesmenu   staff        2502 Apr 22 15:49:15 2021 Makefile
 9     0 drwxr-xr-x   6 jacquesmenu   staff         192 May 21 18:30:08 2021 MusicXML/
10  2552 -rw-r--r--   1 jacquesmenu   staff     1305905 Apr 22 15:49:13 2021 mei-CMN.rng
11     0 drwxr-xr-x   3 jacquesmenu   staff          96 Apr 22 15:49:08 2021 scripts/
```

4.4 The src folder

The src folder has the following structure:

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

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

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

    – msdl2braille
    – msdl2guido
    – msdl2lilypond
    – msdl2musicxml
    – msdlconverter

    – msr2braille
    – msr2guido
    – msr2lilypond
    – msr2musicxml

    – musicxml2braille
    – musicxml2guido
    – musicxml2lilypond
    – musicxml2musicxml

- `generators` :

  - `LilyPondIssue34`

  - `Mikrokosmos3Wandering`

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

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

- `mfutilities` : various utilities, including indented 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 mfMusicformatsError.
     cpp
11  8 -rw-r--r--@  1 jacquesmenu   staff    629 Oct 15 20:34:47 2021 mfMusicformatsError.
     h
12  8 -rw-r--r--@  1 jacquesmenu   staff   2541 Nov  5 11:29:25 2021 oahOptionsVector.cpp
13  8 -rw-r--r--@  1 jacquesmenu   staff    972 Oct 15 20:16:51 2021 oahBasicTypes.h
14 64 -rw-r--r--@  1 jacquesmenu   staff  29773 Oct 15 18:48:10 2021 mfStringsHandling.
     cpp
15 16 -rw-r--r--@  1 jacquesmenu   staff   6269 Oct 15 18:55:46 2021 mfStringsHandling.h
16 16 -rw-r--r--@  1 jacquesmenu   staff   5028 Oct  7 20:03:27 2021 mfTiming.cpp
17  8 -rw-r--r--@  1 jacquesmenu   staff   3726 Oct  8 08:21:09 2021 mfTiming.h
```

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

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

```
13    8 -rw-r--r--@  1 jacquesmenu   staff      918 Nov 16 08:16:55 2021 oah2manPage.cpp
14    8 -rw-r--r--@  1 jacquesmenu   staff      912 Nov 15 12:56:16 2021 oah2manPage.h
15  344 -rw-r--r--@  1 jacquesmenu   staff   175094 Nov 16 08:16:55 2021 oahAtomsCollection
       .cpp
16  176 -rw-r--r--@  1 jacquesmenu   staff    87460 Nov 15 12:56:16 2021 oahAtomsCollection
       .h
17  336 -rw-r--r--@  1 jacquesmenu   staff   168969 Nov 16 08:16:55 2021 oahBasicTypes.cpp
18   96 -rw-r--r--@  1 jacquesmenu   staff    47228 Nov 15 12:56:16 2021 oahBasicTypes.h
19    8 -rw-r--r--@  1 jacquesmenu   staff     3258 Nov 16 08:16:55 2021 oahBrowsers.h
20   32 -rw-r--r--@  1 jacquesmenu   staff    14030 Nov 16 08:16:55 2021 oahElements.cpp
21   24 -rw-r--r--@  1 jacquesmenu   staff    10381 Nov 15 12:56:16 2021 oahElements.h
22    8 -rw-r--r--@  1 jacquesmenu   staff     2577 Nov 16 08:16:55 2021 oahInsiderHandlers
       .cpp
23    8 -rw-r--r--@  1 jacquesmenu   staff     2982 Nov 15 12:56:16 2021 oahInsiderHandlers
       .h
24   56 -rw-r--r--@  1 jacquesmenu   staff    25901 Nov 16 08:16:55 2021 oahOah.cpp
25   32 -rw-r--r--@  1 jacquesmenu   staff    13849 Nov 16 08:16:55 2021 oahOah.h
26    8 -rw-r--r--@  1 jacquesmenu   staff     1966 Nov 16 08:16:55 2021 oahOah2manPage.cpp
27    8 -rw-r--r--@  1 jacquesmenu   staff     1021 Nov 15 12:56:16 2021 oahOah2manPage.h
28   24 -rw-r--r--@  1 jacquesmenu   staff     8831 Nov 16 08:16:55 2021 oahRegularHandlers
       .cpp
29    8 -rw-r--r--@  1 jacquesmenu   staff     3855 Nov 15 12:56:16 2021 oahRegularHandlers
       .h
30    8 -rw-r--r--@  1 jacquesmenu   staff      568 Nov 15 12:56:16 2021 oahVisitor.cpp
31    8 -rw-r--r--@  1 jacquesmenu   staff      894 Nov 15 12:56:16 2021 oahVisitor.h
32   16 -rw-r--r--@  1 jacquesmenu   staff     5978 Nov 16 08:16:55 2021 outputFileOah.cpp
33    8 -rw-r--r--@  1 jacquesmenu   staff     3593 Nov 15 12:56:16 2021 outputFileOah.h
34    8 -rwxr--r--@  1 jacquesmenu   staff      236 Oct 23 12:02:12 2021 zsh_test.zsh*
```

- `formatsgeneration` : support for various output kinds

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

- `passes` : code for the individual passes

  - `bsr2braille`
  - `bsr2bsr`

  - `lpsr2lilypond`

  - `msr2bsr`
  - `msr2lpsr`
  - `msr2msr`
  - `msr2mxsr`

  - `mxsr2guido`
  - `mxsr2msr`
  - `mxsr2musicxml`

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

  - `bsr`

- – `lpsr`

- – `msdl`

- – `msdr`

- – `msr`

- – `msrapi`

- – `mxsr`

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

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

4.5   The `validation` folder

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

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

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

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

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

Chapter 5

Components

5.1   Terminology

In compiler writing terminology:

- an external format

- an internal representation is a data structure representing the program being compiled;

- there are often several internal representations, to simplify the compiler internal workings or for optimisation purposes;

- the output of the compiler, such as binary code for some physical or emulated processor, is a last 'representation' of the program;

- a pass converts an internal representation into another one, in a single step;

- a multi-pass converter is a chain of passes, reading the input, converting it into a first internal representation, then a pass to convert it into another internal representation, and so on until the compiler output is produced.

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

- internal representations (formats for short) of the music score: MSR, LSPR, BSR and MXSR;

- several passes are available to convert such formats into others;

- a set of multi-pass converters are supplied, such as `xml2ly xml2xml` and MSDL converter.

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

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

```
 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.h
 7  ./formats/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
```

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

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

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

5.2   Formats

The formats are in `src/formats`:

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

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

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

5.3   Representations

The representations are in `src/representations`:

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

5.4 Passes

The passs are in `src/passes`:

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

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

```
 1  jacquesmenu@macmini: ~/musicformats-git-dev/src > ll passes/mxsr2msr/
 2  total 1808
 3     0 drwxr-xr-x   8 jacquesmenu   staff      256 Jun 25 05:47:41 2021 ./
 4     0 drwxr-xr-x  16 jacquesmenu   staff      512 May 26 08:20:55 2021 ../
 5    96 -rw-r--r--@  1 jacquesmenu   staff    48389 Jun 21 07:43:20 2021 mxsr2msrOah.cpp
 6    40 -rw-r--r--@  1 jacquesmenu   staff    20327 Jun 16 10:41:37 2021 mxsr2msrOah.h
 7   192 -rw-r--r--@  1 jacquesmenu   staff    97896 Jun 25 08:58:38 2021
       mxsr2msrSkeletonBuilder.cpp
 8    48 -rw-r--r--@  1 jacquesmenu   staff    20942 Jun 25 07:36:29 2021
       mxsr2msrSkeletonBuilder.h
 9  1280 -rw-r--r--@  1 jacquesmenu   staff   651474 Jun 25 07:49:52 2021 mxsr2msrTranslator.cpp
10   152 -rw-r--r--@  1 jacquesmenu   staff    77039 Jun 21 07:43:20 2021 mxsr2msrTranslator.h
```

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

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

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

5.5   Generators

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

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

```
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 multiGenerationOah.
       cpp
6  16 -rw-r--r--@  1 jacquesmenu   staff    6750 Jun  6 06:38:55 2021 mfMultiGenerationOah
       .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:

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

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

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

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

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

  public:

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

                          mfServiceRunData (const string& serviceName);

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

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

    virtual               ~mfServiceRunData ();

  // .. .. .. ..

  private:

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

    // service name
    string                fServiceName;

    // conversion date
    string                fRunDateFull;
    string                fRunDateYYYYMMDD;

    // conversion command
    string                fCommandAsSupplied;

    string                fCommandWithLongOptionsNames;
    string                fCommandWithShortOptionsNames;

    // options and arguments
    mfOptionsAndArguments
                          fOptionsAndArguments;

    // command line
    string                fCommandLineAsSupplied;
```

```
64      // input source
65      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   string serviceName = argv [0];
12
13   // create the global output and log indented streams
14   // --------------------------------------------------------
15
16   createTheGlobalIndentedOstreams (cout, cerr);
17
18   // create the global run data
19   // --------------------------------------------------------
20
21   gGlobalServiceRunData =
22     mfServiceRunData::create (serviceName);
23
24   // ... ... ...
25 }
```

Then the various run data can be accessed easily:

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

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

```
1  lpsrScore::lpsrScore (
2    int                   inputLineNumber,
3    S_msrScore            theMsrScore,
4    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       stringstream s;
17
18       s <<
19         "Generated by " <<
20         gGlobalOahOahGroup->getOahOahGroupServiceName () <<
21         " " <<
22         fMultiComponent->
23           currentVersionNumberAndDateAsString () <<
24         endl <<
```

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

Chapter 6

Command line samples

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

```
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 maintainance tool to check that the MusicFormats library is not initialized more that once;

- `MusicAndHarmonies.cpp` creates a score at random with harmonies in it;

- `Mikrokosmos3Wandering.cpp` and `LilyPondIssue34.cpp` are generators;

- `xml2Any.cpp` uses the `oahOptionsVector` way to supply arguments instead of `arg/argv`;

- `xml2ly`, `xml2brl`, `xml2xml` and `xml2gmn` are converters from MusicXML to other formats;

- `msdlconverter.cpp` it the MSDL converter.

Chapter 7

Warning and errors (WAE)

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

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

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

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

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

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

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

#ifndef ABORT_TO_DEBUG_ERRORS
  #define ABORT_TO_DEBUG_ERRORS
#endif
```

Chapter 8

The trace facility

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

8.1 Activating the trace

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

```
#ifndef ___enableTracingIfDesired___
#define ___enableTracingIfDesired___

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

#endif
```

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

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

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

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

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

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

```
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      Score to voices          (-htstv, -help-trace-score-to-voices):
17        -t<SHORT_NAME>, -trace-<LONG_NAME>
18            Trace SHORT_NAME/LONG_NAME in books to voices.
19      The 10 known SHORT_NAMEs are:
20        book, scores, pgroups, pgroupsd, parts, staves, st, schanges,
21        .
22      The 10 known LONG_NAMEs are:
23        -books, -scores, -part-groups, -part-groups-details,
24        -parts, -staves, -staff-details, -staff-changes, -voices and
25        -voices-details.
26 ... ... ...
```

8.2 Trace categories

8.3 Using traces in practise

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

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

8.4 Debugging traces handling

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

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

the way to go is to:

- locate `gGlobalMxsr2msrOahGroup` in the `*.h` it is declared in;

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

Chapter 9

Multi-lingual support

MusicFormats supports multiple languages in various areas:

- note pitches names in MSR, LPSR and generated LilyPond code;

- LilyPond chord names;

- reserved keywords in MSDL.

Chapter 10

Textual input and output

10.1   Indented output streams

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

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

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

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

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

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

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

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

  // output the indenter
  fOutputSteam << fOutputIndenter;

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

  // reset the buffer
  str ("");

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

```

```
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 `osstream`, among them:

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

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

So when and where are `mfIndentedOstream` instances created?

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

The parameters to this function are:

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

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

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

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

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

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

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

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

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

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

10.3 Indenting the output

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

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

A typical sequence to produce indented output is:

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

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

  ++gIndenter;

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

```
23        "transposeDouble" << " = " << fTransposeDouble <<
24        endl << endl;
25
26    --gIndenter;
27  }
```

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 (ostream& os) const override;
2
3      string                  asString () const override;
4      string                  asStringShort () const override;
```

There are also more specific methods such as:

```
1      void                    printShort (ostream& os) const override;
2
3      void                    printSummary (ostream& os) const override;
```

Note that:

- virtual method **asString ()** produces a rather condensed view of the data to be displayed as part of a single line;

- virtual method **print ()** may produce its output on multiples lines, which always ends with an end of line.

Most classes in MusicFormats can be printed with the **<<** operator:

```
1  ostream& operator<< (ostream& os, const S_msrElement& elt)
2  {
3    elt->print (os);
4    return os;
5  }
```

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

```
1  void msrElement::print (ostream& os) const
2  {
3    os << asString () << 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  string msrTranspose::asString () const
2  {
3    stringstream s;
4
5    s <<
6      "[Transpose" <<
7      ", diatonic = " << fTransposeDiatonic <<
8      ", chromatic = " << fTransposeChromatic <<
9      ", transposeOctaveChange = " << fTransposeOctaveChange <<
10     ", transposeDouble = " << fTransposeDouble <<
11     ", line " << fInputLineNumber <<
12     "]";
```

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

A typical sequence to produce indented output is:

```cpp
1  void msrTranspose::print (ostream& os) const
2  {
3    const int fieldWidth = 22;
4
5    os <<
6      "Transpose" <<
7      ", line " << fInputLineNumber <<
8      endl;
9
10   ++gIndenter;
11
12   os << left <<
13     setw (fieldWidth) <<
14     "transposeDiatonic" << " = " << fTransposeDiatonic <<
15     endl <<
16     setw (fieldWidth) <<
17     "transposeChromatic" << " = " << fTransposeChromatic <<
18     endl <<
19     setw (fieldWidth) <<
20     "transposeOctaveChange" << " = " << fTransposeOctaveChange <<
21     endl <<
22     setw (fieldWidth) <<
23     "transposeDouble" << " = " << fTransposeDouble <<
24     endl << endl;
25
26   --gIndenter;
27 }
```

The main indented output streams are:

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

Chapter 11

Binary data output

Binary data output is done for Braille

Chapter 12

CPU measurements

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

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

           Handle the options and arguments from argc/argv          mandatory     0.01187
Pass 1     Create an MXSR reading a MusicXML file                   mandatory     0.00471
Pass 2a    Create an MSR skeleton from the MXSR                     mandatory     0.00222
Pass 2b    Populate the MSR skeleton from MusicXML data             mandatory     0.00405
Pass 4     Convert the MSR into an LPSR                             mandatory     0.00137
Pass 5     Convert the LPSR score to LilyPond code                 mandatory     0.00136

Total (sec)   Mandatory   Optional
-----------   ---------   ---------
0.02558       0.02558     0.00000
```

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

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

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

  public:

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

    static mfTimingItemsList        gGlobalTimingItemsList;

  public:

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

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

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

Chapter 12. CPU measurements

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

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

    return nullptr;
  }

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

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

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

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

  // create an msr2lpsrTranslator
  msr2lpsrTranslator
    translator (
      originalMsrScore);

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

  clock_t endClock = clock ();

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

Part II

The two-phase visitors pattern

Chapter 13

The two-phase visitors pattern

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

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

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

13.1   Basic mechanism

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

- first phase: visit the node for the fist time, top-down;

- visit the node contents, using the same two-phase visitors pattern;

- second phase: visit the node for the second time, bottom-up.

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

A visitor class should:

- inherit from `basevisitor`;

- inherit from the smart pointer classes it visits;

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

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

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

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

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

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

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

13.2   Browser template classes

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

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

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

    virtual              ~msrBrowser () {}

  public:

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

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

      t.browseData (fVisitor);

      leave (t);
    }

  protected:

    basevisitor*  fVisitor;

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

13.3   A first example: counting notes in MusicMXL data

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

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

Listing 13.1: `libmusicxml/samples/countnotes.cpp`

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

    countnotes() : fCount (0) {}

    virtual ~countnotes () {}

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

13.4   A more complex example

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

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

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

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

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

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

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

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

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

Listing 13.2: Visiting `<scaling/>`

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

  fCurrentMillimeters = (float)(*elt);
}

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

  fCurrentTenths = (float)(*elt);
}

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

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

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

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

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

13.5   Data browsing order

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

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

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

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

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

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

13.6   Selectively inhibiting data browsing

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

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

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

    void                  setInhibitFullMeasureRestsBrowsing ()
                            {
                              fInhibitFullMeasureRestsBrowsing = true;
                            }

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

Class `lpsr2lilypondTranslator` checks this setting:

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

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

```
11                     fVisitedLpsrScore ->
12                       getMsrScore () ->
13                         getInhibitFullMeasureRestsBrowsing ();
14
15                 if (inhibitFullMeasureRestsBrowsing) {
16 #ifdef TRACING_IS_ENABLED
17                   if (
18                     gGlobalTracingOahGroup ->getTraceNotes ()
19                       ||
20                     gGlobalTracingOahGroup ->getTraceFullMeasureRests ()
21                   ) {
22                     gLogStream <<
23                       "% ==> end visiting full measure rests is ignored" <<
24                       endl;
25                   }
26 #endif
27
28 #ifdef TRACING_IS_ENABLED
29                   if (gGlobalTracingOahGroup ->getTraceNotesDetails ()) {
30                     gLogStream <<
31                       "% ==> returning from visitEnd (S_msrNote&)" <<
32                       endl;
33                   }
34 #endif
35
36                   noteIsToBeIgnored = true;
37                 }
38               }
39            break;
40    // ... ... ...
41 }
```

Another example is in the class `lpsr2lilypondTranslator` constructor:

```
1  lpsr2lilypondTranslator :: lpsr2lilypondTranslator (
2    S_lpsrScore       lpsrScore ,
3    S_msrOahGroup&  msrOpts ,
4    S_lpsrOahGroup& lpsrOpts ,
5    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          fetchNoteScoreUpLink ();
 9
10      if (score) {
11        Bool
12          inhibitGraceNotesGroupsBeforeBrowsing =
13            score->
14              getInhibitGraceNotesGroupsBeforeBrowsing ();
15
16        if (inhibitGraceNotesGroupsBeforeBrowsing) {
17 #ifdef TRACING_IS_ENABLED
18          if (
19            gGlobalMsrOahGroup->getTraceMsrVisitors ()
20              ||
21            gGlobalTracingOahGroup->getTraceNotes ()
22              ||
23            gGlobalTracingOahGroup->getTraceGraceNotes ()
24          ) {
25            gLogStream <<
26              "% ==> visiting grace notes groups before is inhibited" <<
27              endl;
28          }
29 #endif
30        }
31        else {
32          // browse the grace notes group before
33          msrBrowser<msrGraceNotesGroup> browser (v);
34          browser.browse (*fNoteGraceNotesGroupBefore);
35        }
36      }
37    }
38
39    // ... ... ...
40 }
```

13.7 Adapting visitors to data browsing order with booleans

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

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

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

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

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

  fOnGoingSystemLayout = true;
}

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

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

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

  fOnGoingSystemLayout = false;
}
```

13.8   Adapting visitors to data browsing order with stacks

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

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

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

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

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

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

    S_msrTuplet
      containingTuplet =
        fOnGoingTupletsStack.top ();

    // 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  }
```

13.9 Avoiding the visiting pattern by cascading

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

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

```
1  void msrPart::appendStaffDetailsToPart (
2    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       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 cascadingthrough `printSlices ()` methods, see chapter 20, [MSR time-oriented represension], page 186.

Part III

Programming style and conventions

Chapter 14

Programming style and conventions

14.1 Files naming conventions

Most file names start with an identification of the component they belong to, such as 'oah', 'mxsr', 'msr', 'lpsr', 'lilypond', 'bsr', 'braille', 'xml2ly', 'xml2brl' and msdl.
The ancillary files such as src/utilities/mfIndentedTextOutput.h/.cpp follow this rule too, with an mf prefix.

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

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

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

```
 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 mfMusicformatsError.cpp
11  8 -rw-r--r--@  1 jacquesmenu  staff    629 Oct 15 20:34:47 2021 mfMusicformatsError.h
12  8 -rw-r--r--@  1 jacquesmenu  staff   2541 Nov  5 11:29:25 2021 oahOptionsVector.cpp
13  8 -rw-r--r--@  1 jacquesmenu  staff    972 Oct 15 20:16:51 2021 oahBasicTypes.h
14 64 -rw-r--r--@  1 jacquesmenu  staff  29773 Oct 15 18:48:10 2021 mfStringsHandling.cpp
15 16 -rw-r--r--@  1 jacquesmenu  staff   6269 Oct 15 18:55:46 2021 mfStringsHandling.h
16 16 -rw-r--r--@  1 jacquesmenu  staff   5028 Oct  7 20:03:27 2021 mfTiming.cpp
17  8 -rw-r--r--@  1 jacquesmenu  staff   3726 Oct  8 08:21:09 2021 mfTiming.h
```

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

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

```
 9  formats/lpsr/lpsrBarNumbers.h:85:class EXP lpsrBarCommand : public lpsrElement
10  formats/lpsr/lpsrLyrics.h:31:class EXP lpsrNewLyricsBlock : public lpsrElement
11  formats/lpsr/lpsrComments.h:25:class EXP lpsrComment : public lpsrElement
12  formats/lpsr/lpsrVoices.h:29:class EXP lpsrUseVoiceCommand : public lpsrElement
13  formats/lpsr/lpsrParts.h:27:class EXP lpsrPartBlock : public lpsrElement
14  formats/lpsr/lpsrPapers.h:32:class EXP lpsrPaper : public lpsrElement
15  formats/lpsr/lpsrPartGroups.h:29:class EXP lpsrPartGroupBlock : public lpsrElement
16  formats/lpsr/lpsrParallelMusic.h:28:class EXP lpsrParallelMusicBLock : public lpsrElement
17  formats/lpsr/lpsrLayouts.h:23:class EXP lpsrLayout : public lpsrElement
18  formats/lpsr/lpsrHeaders.h:27:class EXP lpsrHeader : public lpsrElement
19  formats/lpsr/lpsrScheme.h:29:class EXP lpsrSchemeVariable : public lpsrElement
20  formats/lpsr/lpsrScheme.h:140:class EXP lpsrSchemeFunction : public lpsrElement
21  formats/lpsr/lpsrBookBlockElements.h:35:class EXP lpsrBookBlockElement : public
        lpsrElement
22  formats/lpsr/lpsrBookBlockElements.h:237:class EXP lpsrBookBlock : public lpsrElement
23  formats/lpsr/lpsrContexts.h:30:class EXP lpsrContext : public lpsrElement
```

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

```
1  ./formats/msdl/msdlBasicTypes.h
2  ./formats/msdl/msdlBasicTypes.cpp
3  ./formats/msr/msrBasicTypes.cpp
4  ./formats/msr/msrBasicTypes.h
5  ./formats/lpsr/lpsrBasicTypes.cpp
6  ./formats/lpsr/lpsrBasicTypes.h
7  ./formats/bsr/bsrBasicTypes.h
8  ./formats/bsr/bsrBasicTypes.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

14.2   Adding C++ files

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

When building MusicFormats with:

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

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

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

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

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

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

- git mv oldName newName

- git commit "..." -a

- git push

14.3   Renaming C++ files

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

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

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

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

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

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

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

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

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

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

14.4   Source code layout

The following text-editing conventions are used:

- tabs are not used before the first non-space character in a line, two spaces are used instead;

- the code is not tightly packed: declarations in classes have the members' names aligned vertically, with many spaces before them if needed, and empty lines are used to separate successive activities in methods.

14.5   Defensive programming

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

- identifiers are explicit and long if needed – only very local ones are short, such as iteration loops indexes;

- the code is organized in sections, with an initial comment documenting what the code does;

- the C++11's `auto` declaration feature is used only for enumeration type `s`, see below. Writing the explicit types in a large code base helps the maintainer mastering the code;

- function `mfAssert ()` is used to perform sanity checks, such as detect a null pointer prior to using it.

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

```
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:

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

    class Iterator
    {
      public:

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

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

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

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

      private:

        int fIterationIndex;
    };
};
```

14.6   Sanity checks

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

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

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

14.7   JMI comments

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

14.8   Exported symbols

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

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

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

14.9   Smart pointers

libmusicxml2 provides what Dominique Fober named smart pointers, because:

- a smart pointer is an instance of a class that contains the actual pointer in the usual C++ sense;

- the actual pointer is guaranteed to be initialized to **nullptr**;

- garbage collection is implicit, using reference counts.

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

The reference counting is done in class **smartable**:

```
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`:

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

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

    static SMARTP<waeHandler> create ();

  public:

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

                         waeHandler ();

    virtual              ~waeHandler ();

  public:

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

  public:

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

  public:

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

    string               asString () const;

    void                 print (ostream& os) const;

  private:

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

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

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

14.10   Files contents layout

Indentation is done by two spaces, avoiding TAB characters.

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

- public data types, usually enumeration type **s**, if any;

- public static class `create*` () methods, except for pure virtual classes, in which case they are commented out;

- constructors ad destructor;

- public `set*` () and `get*` () methods;

- public services if any;

- public visiting methods, i.e. `acceptIn` (), `acceptOut` () and `browseData` (), if the class contains browsable data such as STL lists, vectors, maps and sets;

- public print methods, such as `asString` () and `print` ();

- private methods if any;

- private fields.

- private work methods if any;

- private work fields.

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

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

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

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

14.11   `#define DEBUG*` code sections

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

- `//#define DEBUG_EARLY_OPTIONS`::in src/oah/oahEarlyOptions.cpp;

- `//#define DEBUG_INDENTER`::in src/oah/mfIndentedTextOutput.cpp;

- `//#define DEBUG_SPLITTING`::in src/mfutilities/mfStringsHandling.cpp

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

14.12   Identifiers naming conventions

The following rules apply:

- all enumeration type names describing variants in classes end in 'Kind';

- all enumeration constants start with 'k';

- all classes names have a prefix indicating which part of MusicFormats there belong to, such as class msrTimeSignature, oahAtomStoringAValue and msdlKeywordsLanguageAtom;

- all classes member fields start with 'f';

- all global variables start with 'gGlobal';

- all variables private to methods start with 'pPrivate';

- some K_* constants are defined with #define because creating global constants variables and accessing them would be too cumbersome, such as:

```
1    \#define msrPart::K_PART_HARMONIES_STAFF_NUMBER      10
2    \#define msrPart::K_PART_HARMONIES_VOICE_NUMBER      11
3
4    \#define msrPart::K_PART_FIGURED_BASS_STAFF_NUMBER   20
5    \#define msrPart::K_PART_FIGURED_BASS_VOICE_NUMBER   21
```

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

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

or

```
1                          fStringToDalSegnoKindMapVariable;
```

or

```
1    map<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`.

14.13   Exceptions and warnings/errors reporting

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

All the exception classes are derived from , that contains:

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

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

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

  // ... ... ...

  private:

    string                fExceptionDescription;

    int                   fExceptionNumber;
    int                   fExceptionLevel;
};
```

An example of exception is:

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

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

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

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

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

```
11  }
```

and:

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

Another one is:

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

14.14 Exporting symbols for Windows DLLs

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

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

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

```
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
```

14.15   Dynamic type checking

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

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

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

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

14.16   Input line numbers

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

```
    int                     fInputLineNumber;
```

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

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

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

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

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

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

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

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

```
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.

14.17 Static declarations

They are used for:

- classes methods such as method `msrTie::create ()`, method `msrTempo::createTempoPerMinute ()` and method `msrTemp::tempoKindAsString ()`;

- classes constant fields such as constant `msrStanza::K_NO_STANZA_NUMBER`, to be prefered to C-style `#define` preprocessor specifications for type safety;

- functions of methods remanent variables such as function private variable `pPrivateThisMethodHasBeenRun` in function `initializeMsrGenerationAPI ()`;

- library-wide variables such as global variable `gGlobalOStreamIndenter` and global variable `gGlobalTimingItem` that would be too cumbersome to pass to each and every method or function that uses them.

14.18 Avoiding MusicFormats multiple initializations

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

```
1  void initializeBSR ()
2  {
3    // protect library against multiple initializations
4    static Bool pPrivateThisMethodHasBeenRun (false);
5
6    if (! pPrivateThisMethodHasBeenRun) {
7      initializeBsrBasicTypes ();
8
9      pPrivateThisMethodHasBeenRun = true;
10   }
11 }
```

14.19   Enumeration types

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

```
1  enum class msrSlurTypeKind {
2    k_NoSlur ,
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_No*` are used to indicate a value that *has not been set yet*. There are always the first on in the corresponding enumeration, to benefit from the the C++11 implicit initialization to the the equivalent of 0.

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

```
1      // repeat winged
2      enum 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      string winged = elt->getAttributeValue ("winged");
6
7      fCurrentBarLineRepeatWingedKind =
8        msrBarLine::kBarLineRepeatWingedNone; // default value
9
10     if (winged.size ()) {
11       if        (winged == "none") {
12         fCurrentBarLineRepeatWingedKind =
13           msrBarLine::kBarLineRepeatWingedNone;
14       }
```

```
15        else if (winged == "straight") {
16          fCurrentBarLineRepeatWingedKind =
17            msrBarLine::kBarLineRepeatWingedStraight;
18        }
19        else  if (winged == "curved") {
20          fCurrentBarLineRepeatWingedKind =
21            msrBarLine::kBarLineRepeatWingedCurved;
22        }
23        else  if (winged == "double-straight") {
24          fCurrentBarLineRepeatWingedKind =
25            msrBarLine::kBarLineRepeatWingedDoubleStraight;
26        }
27        else  if (winged == "double-curved") {
28          fCurrentBarLineRepeatWingedKind =
29            msrBarLine::kBarLineRepeatWingedDoubleCurved;
30        }
31        else {
32          stringstream s;
33
34          s <<
35            "repeat winged \"" << winged <<
36            "\" is unknown";
37
38          musicxmlError (
39            gGlobalServiceRunData->getInputSourceName (),
40            inputLineNumber,
41            __FILE__, __LINE__,
42            s.str ());
43        }
44      }
45
46      // ... ... ...
```

14.20   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 if far from obvious.

This is why we use enum classes such as:

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

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

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

14.21   Boolean values anyway

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

Moreover, a `bool` variable not explicitly initialized in the 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.

14.22   Iterating over numeration types

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

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

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

  // MusicXML harmonies

  kHarmonyMajor, kHarmonyMinor,
  kHarmonyAugmented, kHarmonyDiminished,

  // ... ... ...

  // other

  kHarmonyOther,

  kHarmonyNone,

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

  AllFirst = k_NoHarmony,
  AllLast  = kHarmonyNone,

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

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

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

    class Iterator
    {
```

```
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        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 (ostream& os)
2 {
3   os <<
4     "All the known harmonies structures are:" <<
5     endl << 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      endl;
20  } // for
21
22  --gIndenter;
23 }
```

14.23   Rational numbers

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

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

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

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

14.24   Default values

The guide lines for MusicFormats in this matter are:

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

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

- all variables and classes fields of non-class types, such as `int`, `float` and enumeration type `s`, are to be initialized explicitly;

- MusicFormats functions and methods parameters never have default values: overloading is used instead.

14.25   `create*` methods

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

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

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

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

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

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

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

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

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

```
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:

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

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

  return result;
}
```

14.27   `initialize*()` and `finalize*()` methods

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

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

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

  // set part number
  fPartAbsoluteNumber = ++gPartsCounter;

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

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

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

14.28   `*asString()` and `*fromString()` functions

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

```cpp
// placement
//------------------------------------------------------------------------
enum class msrPlacementKind {
  k_NoPlacement,

  kPlacementAbove, kPlacementBelow
};
```

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

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

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

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

  return result;
}
```

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

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

  return result;
}
```

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

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

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

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

```cpp
//_____
class EXP msr2msrTranslator :

  // MSR score

  public visitor<S_msrScore>,

  // ... ... ...

{
  public:

                          msr2msrTranslator ();

    virtual               ~msr2msrTranslator ();

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

};
```

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

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

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

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

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

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

  return fResultingNewMsrScore;
}
```

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

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

```
 6    const 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  }
```

14.30 `context` arguments

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

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

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

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

14.31 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 multiset<S_oahElement, compareOahElements>&
3 oah/oahBasicTypes.h:    multiset<S_oahElement, compareOahElements>
4 oah/oahElements.h:struct compareOahElements {
5 formats/msr/msrParts.h:    static bool
    compareStavesToHaveFiguredBassElementsBelowCorrespondingPart (
6 formats/msr/msrNotes.h:    static bool            compareNotesByIncreasingPositionInMeasure
     (
7 formats/msr/msrMeasureElements.h:    static bool
    compareMeasureElementsByIncreasingPositionInMeasure (
8 formats/msr/msrStaves.h:    static bool            compareVoicesByIncreasingNumber (
9 formats/msr/msrStaves.h:    static bool
    compareVoicesToHaveHarmoniesAboveCorrespondingVoice (
10 formats/msr/msrStaves.h:    static bool
    compareVoicesToHaveFiguredBassElementsBelowCorrespondingVoice (
11 formats/msr/msrMeasuresSlices.h:    static bool
    compareNotesEventsByIncreasingPositionInMeasure (
12 formats/msr/msrMeasuresSlices.h:    static bool
    compareSimultaneousNotesChunksByIncreasingPositionInMeasure (
13 formats/lpsr/lpsrParts.h:    static bool
    compareElementsToHaveHarmoniesAboveCorrespondingStaff (
14 formats/lpsr/lpsrParts.h:    static bool            compareStaffBlockWithOtherElement (
15 formats/lpsr/lpsrParts.h:    static bool
    compareChordNamesContextWithOtherElement (
16 utilities/mfutilities.h:    // compare indentation value
```

An example is:

```
1 bool msrPart::compareStavesToHaveFiguredBassElementsBelowCorrespondingPart (
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 }
```

Part IV

MusicFormats components

Chapter 15

musicformats components (MFC)

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

15.1   Versions numbers

The basic data structure is class `mfcVersionNumber`:

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

  public:

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

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

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

    Bool                  operator<  (const mfcVersionNumber& other) const;

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

    Bool                  operator>  (const mfcVersionNumber& other) const;

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

  public:

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

    string                asString () const;

    void                  print (ostream& os) const;

  private:

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

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

```
39    string                   fPreRelease;
40 };
```

15.2   Versions descriptions

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

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

15.3   Versions histories

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

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

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

```
1  S_mfcVersionDescr mfcVersionsHistory::fetchCurrentVersion () const
2  {
3    // sanity check
4    mfAssert (
5      __FILE__, __LINE__,
6      fVersionsList.size () > 0,
```

```
7      "fVersionsList is empty");
8
9    return fVersionsList.back ();
10 }
```

15.4   Components descriptions

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

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

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

```
1 class mfcComponentDescr : public smartable
2 {
3   // ... ... ...
4
5   public:
6
7     // public services
8     // ------------------------------------------------------
9
10    S_mfcVersionDescr      fetchComponentCurrentVersion () const
11                              {
12                                return
13                                  fVersionsHistory->
14                                    fetchCurrentVersion();
15                              }
16
17  // ... ... ...
18
19  public:
20
21    // print
22    // ------------------------------------------------------
23
24    string                asString () const;
25
26    string                currentVersionNumberAndDateAsString () const;
27
28    virtual void          print (ostream& os) const;
29
30    virtual void          printVersion (ostream& os) const;
31    virtual void          printHistory (ostream& os) const;
32
33  protected:
34
35    // protected services
36    // ------------------------------------------------------
37
38    virtual void          printOwnHistory (ostream& os) const;
39
40  protected:
41
42    // protected fields
43    // ------------------------------------------------------
```

```
44
45     string                 fComponentName;
46
47     mfcComponenKind        fComponenKind;
48
49     S_mfcVersionsHistory   fVersionsHistory;
50  };
```

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

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

- class `mfcRepresentationComponent` for formats;

- class `mfcPassComponent` for passes;

- purely virtual class `mfcMultiComponent` for the generators, converters and MusicFormats library, itself the superclass of:

  - class `mfcGeneratorComponent`;
  - class `mfcConverterComponent`;
  - class `mfcLibraryComponent`.

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

15.5   Multi-components

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

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

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

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

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

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

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

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

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

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

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

```
1  jacquesmenu@macmini: ~ > xml2brl -history
2  Command line version of musicxml2braille converter v0.9.51 (October 12 2021)
3
4  Own history:
5    v0.9.5 (October 6 2021):
6      Start of sequential versions numbering
7
8    v0.9.51 (October 12 2021):
9      Fixed trace OAH issue
10
11 Representations history:
12   MXSR
13     v0.9.5 (October 6 2021):
14       Start of sequential versions numbering
15
16   MSR
17     v0.9.5 (October 6 2021):
18       Start of sequential versions numbering
19
20     v0.9.51 (October 14 2021):
```

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

15.6   Versions history creation

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

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

15.6.1   Representations and passes components creation

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

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

```
1 S_mfcRepresentationComponent EXP createMsrRepresentationComponent ()
2 {
3   static S_mfcRepresentationComponent pRepresentationComponent;
4
5   // protect library against multiple initializations
6   if (! pRepresentationComponent) {
7
8 #ifdef TRACING_IS_ENABLED
9     if (gGlobalOahEarlyOptions.getEarlyTraceComponents ()) {
10       gLogStream <<
11         "Initializing MSR format component" <<
12         endl;
13     }
14 #endif
15
16     // create the format component
17     pRepresentationComponent =
18       mfcRepresentationComponent::create (
19         "MSR");
```

```
20
21     // populate it
22     pRepresentationComponent ->
23       appendVersionDescrToComponent (
24         mfcVersionDescr::create (
25           mfcVersionNumber::createFromString ("0.9.50"),
26           "October 6, 2021",
27           list<string> {
28             "Start of sequential versions numbering"
29           }
30       ));
31
32     pRepresentationComponent ->
33       appendVersionDescrToComponent (
34         mfcVersionDescr::create (
35           mfcVersionNumber::createFromString ("0.9.51"), // JMI
36           "October 14, 2021",
37           list<string> {
38             "Refined MSR names and summary display options"
39           }
40       ));
41   }
42
43   return pRepresentationComponent;
44 }
```

The conversion of MusicXML to MXSRdoes not belong to MusicFormats since it is provided by `libmusicxml2`.

15.6.2   Generators and converters components creation

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

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

```
1  S_mfcConverterComponent EXP createMusicxml2brailleConverterComponent ()
2  {
3    static S_mfcConverterComponent pConverterComponent;
4
5    // protect library against multiple initializations
6    if (! pConverterComponent) {
7
8  #ifdef TRACING_IS_ENABLED
9      if (gGlobalOahEarlyOptions.getEarlyTraceComponents ()) {
10       gLogStream <<
11         "Creating the musicxml2braille component" <<
12         endl;
13     }
14 #endif
15
16     // create the converter component
17     pConverterComponent =
18       mfcConverterComponent::create (
19         "musicxml2braille",
20         mfcMultiComponentEntropicityKind::kComponentEntropicityNo,
21         mfcMultiComponentUsedFromTheCLIKind::kComponentUsedFromTheCLIYes); // JMI ???
22
23     // populate the converter's own history
24     pConverterComponent ->
25       appendVersionDescrToComponent (
26         mfcVersionDescr::create (
27           mfcVersionNumber::createFromString ("0.9.50"),
28           "October 6, 2021",
```

```
29          list<string> {
30            "Start of sequential versions numbering"
31          }
32        ));
33
34      pConverterComponent->
35        appendVersionDescrToComponent (
36          mfcVersionDescr::create (
37            mfcVersionNumber::createFromString ("0.9.51"),
38            "October 12, 2021",
39            list<string> {
40              "Fixed trace OAH issue"
41            }
42        ));
43
44      // populate the converter's formats list
45      pConverterComponent->
46        appendRepresentationToMultiComponent (
47          createMxsrRepresentationComponent ());
48      pConverterComponent->
49        appendRepresentationToMultiComponent (
50          createMsrRepresentationComponent ());
51      pConverterComponent->
52        appendRepresentationToMultiComponent (
53          createBsrRepresentationComponent ());
54
55      pConverterComponent->
56        appendPassToMultiComponent (
57          createMxsr2msrComponent ());
58
59      pConverterComponent->
60        appendPassToMultiComponent (
61          createMsr2msrComponent ());
62
63      pConverterComponent->
64        appendPassToMultiComponent (
65          createMsr2bsrComponent ());
66
67      pConverterComponent->
68        appendPassToMultiComponent (
69          createBsr2bsrComponent ());
70
71      pConverterComponent->
72        appendPassToMultiComponent (
73          createBsr2brailleComponent ());
74    }
75
76    return pConverterComponent;
77  }
```

15.6.3   MusicFormats library component creation

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

```
1  S_mfcLibraryComponent EXP createLibraryComponent ()
2  {
3    static S_mfcLibraryComponent pLibraryComponent;
4
5    // protect library against multiple initializations
6    if (! pLibraryComponent) {
7
8  #ifdef TRACING_IS_ENABLED
9      if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
10       gLogStream <<
```

```
11          "Creating the  MFC library component" <<
12          endl;
13      }
14  #endif
15
16      // create the library's history
17      pLibraryComponent =
18        mfcLibraryComponent::create (
19          "musicformats",
20          mfcMultiComponentEntropicityKind::kComponentEntropicityNo,
21          mfcMultiComponentUsedFromTheCLIKind::kComponentUsedFromTheCLIYes); // JMI ???
22
23      // populate the library's own history
24      pLibraryComponent->
25        appendVersionDescrToComponent (
26          mfcVersionDescr::create (
27            mfcVersionNumber::createFromString ("0.9.50"),
28            "October 6, 2021",
29            list<string> {
30              "Start of sequential versions numbering"
31            }
32        ));
33
34      pLibraryComponent->
35        appendVersionDescrToComponent (
36          mfcVersionDescr::create (
37            mfcVersionNumber::createFromString ("0.9.51"),
38            "October 12, 2021",
39            list<string> {
40              "Adding a version number to the MusicFormats library",
41              "Fixed trace OAH issue in the musicxml2* converters)"
42            }
43        ));
44
45      pLibraryComponent->
46        appendVersionDescrToComponent (
47          mfcVersionDescr::create (
48            mfcVersionNumber::createFromString ("0.9.52"),
49            "October 12, 2021",
50            list<string> {
51              "Added MusicFormats library versions history to '-hist, -history'"
52            }
53        ));
54
55      pLibraryComponent->
56        appendVersionDescrToComponent (
57          mfcVersionDescr::create (
58            mfcVersionNumber::createFromString ("0.9.53"),
59            "October 22, 2021",
60            list<string> {
61              "Replaced bool by class Bool in variables and fields",
62              "Created MFC (MusicFormats components)"
63            }
64        ));
65
66      pLibraryComponent->
67        appendVersionDescrToComponent (
68          mfcVersionDescr::create (
69            mfcVersionNumber::createFromString ("0.9.54"),
70            "Novermber 6, 2021",
71            list<string> {
72              "Replaced cout and cerr by gOutputStream and gLogStream respectively in the
    CLI samples",
73              "Finalized components numbering (MFC)"
74            }
75        ));
76
```

```
77      // populate the library's components history
78      pLibraryComponent->
79        appendRepresentationToMultiComponent (
80          createMsrRepresentationComponent ());
81      pLibraryComponent->
82        appendRepresentationToMultiComponent (
83          createLpsrRepresentationComponent ());
84      pLibraryComponent->
85        appendRepresentationToMultiComponent (
86          createBsrRepresentationComponent ());
87      pLibraryComponent->
88        appendRepresentationToMultiComponent (
89          createMxsrRepresentationComponent ());
90
91      pLibraryComponent->
92        appendPassToMultiComponent (
93          createMsr2msrComponent ());
94
95      pLibraryComponent->
96        appendPassToMultiComponent (
97          createMsr2lpsrComponent ());
98      pLibraryComponent->
99        appendPassToMultiComponent (
100         createLpsr2lilypondComponent ());
101
102     pLibraryComponent->
103       appendPassToMultiComponent (
104         createMsr2bsrComponent ());
105     pLibraryComponent->
106       appendPassToMultiComponent (
107         createBsr2bsrComponent ());
108     pLibraryComponent->
109       appendPassToMultiComponent (
110         createBsr2brailleComponent ());
111
112     pLibraryComponent->
113       appendPassToMultiComponent (
114         createMsr2mxsrComponent ());
115
116     pLibraryComponent->
117       appendPassToMultiComponent (
118         createMxsr2musicxmlComponent ());
119
120     pLibraryComponent->
121       appendPassToMultiComponent (
122         createMxsr2guidoComponent ());
123   }
124
125   return pLibraryComponent;
126 }
```

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

15.6.4   Version and history options handling

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

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

```
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` subclasses constructors by a call to the overriden `initializeHandlerMultiComponent ()`.

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

```
1  xml2xmlInsiderHandler::xml2xmlInsiderHandler (
2    const string& serviceName,
3    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 overriden `initializeHandlerMultiComponent ()` methods merely get the atom or compound versions to assign it to field `oahHandler::fHandlerMultiComponent`.

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

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

15.7   Accessing versions in regular handlers

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

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

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

  protected:

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

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

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

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

15.8   Getting current version numbers

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

15.8.1 Current version numbers in options

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  public:

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

    void                      applyElement (ostream& os) override;

  // ... ... ...

  public:

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

    // ... ... ...

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

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

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

  int saveIndent = gIndenter.getIndent ();

  gIndenter.resetToZero ();

  printVersion (os);

  gIndenter.setIndent (saveIndent);
}
```

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

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

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

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

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

15.8.2   Current version numbers in formats

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

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

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

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

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

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

Class `lpsrScore` contains an MFC component field:

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

  private:

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

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

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

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

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

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

  fMsrScore = theMsrScore;

  fMultiComponent = multiComponent;

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

```
20      s <<
21        "Generated by " <<
22        gGlobalOahOahGroup ->getOahOahGroupServiceName () <<
23        " " <<
24        fMultiComponent ->
25          currentVersionNumberAndDateAsString () <<
26        endl <<
27
28        "% on " <<
29        gGlobalServiceRunData ->getTranslationDateFull () <<
30        endl <<
31
32        "% from ";
33
34      if (gGlobalServiceRunData ->getInputSourceName () == "-") {
35        s << "standard input";
36      }
37      else {
38        s << "\"" << gGlobalServiceRunData ->getInputSourceName () << "\"";
39      }
40
41      fInputSourceNameComment =
42        lpsrComment ::create (
43          inputLineNumber ,
44          s.str (),
45          lpsrComment ::kGapAfterwardsNo);
46    }
47
48    // ... ... ... ...
49  }
50
51  // ... ... ... ...
52 }
```

15.8.3   Current version numbers in passes

Another case is that of the generation of MusicXML output:

```
1 jacquesmenu@macmini: ~/musicformats -git-dev/files/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   string
12     passComponentCurrentVersionNumberAndDateAsString =
13       passComponent->
14         currentVersionNumberAndDateAsString ();
15
16   // create the initial creation comment
17   stringstream s;
18   s <<
19     endl <<
20     "=================================================" <<
21     endl <<
22     "Created by " <<
23     gGlobalOahOahGroup->getOahOahGroupServiceName () <<
24     " " <<
25     passComponentCurrentVersionNumberAndDateAsString <<
26     endl <<
27
28     "on " <<
29     gGlobalServiceRunData->getTranslationDateFull () <<
30     endl <<
31
32     "from " <<
33     gGlobalServiceRunData->getInputSourceName () <<
34     endl <<
35
36     "=================================================" <<
37     endl;
38
39   // append the initial creation comment to the score part wise element
40   fResultingMusicxmlelement->push (createMxmlelement (kComment, s.str ()));
41
42   // create a software element
43   Sxmlelement
44     softwareElement =
45       createMxmlelement (
46         k_software,
47         gGlobalOahOahGroup->getOahOahGroupServiceName ()
48           + " "
49           + passComponentCurrentVersionNumberAndDateAsString +
50           ", https://github.com/jacques-menu/musicformats");
51
52   // append it to the identification encoding
53   appendToScoreIdentificationEncoding (softwareElement);
54
55   // ... ... ... ...
56  }
```

Part V

Options and help

Chapter 16

Options and help (OAH)

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

- there is a great number of options in MusicFormats;

- attaching the help to the options in a clean, neat way was highly desirable;

- more important still, the re-use of options whenever translators are combined into converters could only be achieved with an object oriented handling of the options and help.

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

16.1   OAH basics

- OAH (Options And Help) is supposed to be pronounced something close to "whaaaah!" The intonation is left to the speaker, though... And as the saying goes: "OAH? why not!"

- options handling is organized as a hierarchical, introspective set of classes. An options and its corresponding help are grouped in a single object.

- the options can be supplied thru:

  - the command line, in `argv`. This allows for mixed options and arguments in any order, à la GNU;
  - the API functions such as function `musicxmlfile2lilypond ()`, in an options and arguments.

- class `oahElement` is the super-class of all options types, including groups and subgroups. It contains a short name and a long name, as well as a description. Short and long names can be used and mixed at will in the command line and in option vectors (API), as well as '-' and '--'. The short name is mandatory, but the long name may be empty if the short name is explicit enough.

- prefixes such '-t=' and `-help=`' allow for a contracted form of options. For example, `-t=meas,notes` is short for '-t-meas, -tnotes'. A `oahPrefix` contains the prefix name, the ersatz by which to replace it, and a description.

- a class `oahHandler` contains a list of `oahGroup`'s, each handled in a pair of `.h/.cpp` files such as `src/formats/msr/msrOah.h` and `src/formats/msr/msrOah.cpp`, and a list of options prefixes.

- a class `oahGroup` contains a list of `oahSubGroup`'s and an upLink to the containing `oahHandler`.

- a class `oahSubGroup` contains a list of `oahAtom`'s and an upLink to the containing `oahGroup`.

- each class `oahAtom` contains an atomic option and the corresponding help, and an upLink to the containing `oahSubGroup`.

16.2   Features

- the values of the various options can be displayed with the option `-display-options-values, -dov` option;

- partial help can be obtained, i.e. help about any group, subgroup or atom, showing the path in the hierarchy down to the corresponding option;

- there are various subclasses of class `oahAtom` such as class `oahIntegerAtom`, class `oahBooleanAtom` and class `oahRationalAtom`, to control options values of common types;

- class `oahThreeBooleansAtom`, for example, allows for three boolean settings to be controlled at once with a single option;

- class `oahAtomStoringAValue` describes options for which a value is supplied in the command line or in option vectors (API);

- a class such as class `lpsrPitchesLanguageAtom` is used to supply a string value to be converted into an internal enumerated type;

- a class `oahCombinedBooleansAtom` contains a list of boolean atoms to manipulate several such atoms as a single one, see the 'class `cubase`' combined booleans atom in `src/passes/mxsr2msr/mxsr2msrOah.cpp`;

- class `oahMultiplexBooleansAtom` contains a list of boolean atoms sharing a common prefix to display such atoms in a compact manner, see the '`ignore-redundant-clefs`' multiplex booleans atom in `src/passes/mxsr2msr/mxsr2msrOah.cpp`;

- storing options and the corresponding help in class `oahGroup`'s makes it easy to re-use them. For example, file `xml2ly` and file `xml2lbr` have their three first passes in common, (up to obtaining the MSR description of the score), as well as the corresponding options and help;

- `src/oah/oahAtomsCollection` contains a bunch of general purpose options such as class `oahContactAtom`, class `oahFloatAtom` and class `oahLengthAtom`;

- a regular handler (used by default unless the option `-insider` is used), presents the options and help grouped by subject, such as voices and tuplets. It uses an insider handler, which groups them by internal representation and conversion pass. This is how options groups are re-used for various converters such as file `xml2ly`, file `xml2brl` and file `xml2xml`.

16.3   Atoms expecting a value

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

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

```
class EXP oahAtomExpectingAValue : public oahAtom
/*
  a purely virtual common ancestor for all atom classes
  that take a value from argv or an oahOptionsVector
*/
{
  // ... ... ...

  public:

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

```
13
14     void                    applyElement (ostream& os) override;
15                              // reports an error
16
17     virtual void            applyAtomWithValue (
18                               const string& theString,
19                               ostream&      os) = 0;
20
21     virtual void            applyAtomWithDefaultValue (ostream& os);
22                              // used only if fElementValueKind
23                              // is oahElementValueKind::kElementValueImplicit
24                              // or oahElementValueKind::kElementValueOptional
25 };
```

The classes derived from **oahAtomExpectingAValue** are:

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

16.3.1   The `oahAtomStoringAValue` class

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

```
1 class EXP oahAtomStoringAValue : public oahAtomExpectingAValue
2 {
3   // ... ... ...
4
5   public:
6
7     // print
8     // ------------------------------------------------------
9
10     virtual void            printAtomWithVariableEssentials (
11                               ostream& os,
12                               int fieldWidth) const;
13     virtual void            printAtomWithVariableEssentialsShort (
14                               ostream& os,
15                               int fieldWidth) const;
16
17     void                    print (ostream& os) const override;
18     void                    printShort (ostream& os) const override;
19
20     void                    printHelp (ostream& os) const override;
21
22     virtual void            printAtomWithVariableOptionsValues (
23                               ostream& os,
24                               int      valueFieldWidth) const override;
25
26   protected:
27
28     // protected fields
29     // ------------------------------------------------------
30
31     string                  fValueSpecification;
32
33     string                  fVariableName;
34     Bool                    fSetByUser;
```

```
35  };
```

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

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

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

16.3.2   The `oahBooleanAtom` special case

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

```
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      string                fVariableName;
19      Bool&                 fBooleanVariable;
20      Bool                  fSetByUser;
21  };
```

16.3.3   Checking whether an option has been selected

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

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

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

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

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

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

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

16.3.4   The `oahAtomStoringAValue` subclasses

The classes derived from `oahAtomStoringAValue` are:

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

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

16.4 Pure help atoms

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

16.4.1 Pure help atoms without a value

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

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

The actual pure help atoms without a value are:

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

16.4.2 Pure help atoms expecting a value

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

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

The actual pure help atoms expecting a value are:

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

16.5   Options implicitly storing a value

There are options in multiGeneration to select the generated output :

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

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

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

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

  protected:

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

    string                fVariableName;
    Bool                  fSetByUser;
};
```

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

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

  private:

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

    mfMultiGenerationOutputKind&
                      fMultiGenerationOutputKindVariable;
};
```

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

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

16.6 Options and help handling

- each option short name and non-empty long name must be unique in a given handler, to avoid ambiguities;

- an service `main ()` calls method `oahHandler::handleOptionsAndArgumentsFromArgcArgv ()`, in which:

- method `oahHandler::handleOptionNameCommon ()` handles the option names;

- `handleOptionValueOrArgument()` and the arguments to the service.

- contracted forms are expanded in method `oahHandler::handleOptionNameCommon ()` before the resulting, uncontracted options are handled;

- options handling works in two passes:

- the first one creates a list of class `oahElementUse` instances from `argc`/`argv` or an options and arguments;

- the second one traverses this list to apply the options that are used.

- the options are applied by virtual method `applyElement ()`, virtual method `applyAtomWithValue ()` and virtual method `applyAtomWithDefaultValue ()`;

- method `oahHandler::handleKnownArgvAtom ()` associatiates the value to the (preceding) field `oahHandler::fl` if not null, or appends it to field `oahHandler::fHandlerArgumentsVector` to otherwise;

- `fPendingArgvAtomExpectingAValue` is used in `argv` contents handling to associate an option name with it value, which is the next element in `argv`.

16.7 Basic OAH types

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

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

```
 8  enum class oahHandlerUsedThruKind {
 9  class oahPrefix;
10  class EXP oahPrefix : public smartable
11    a common ancestor for all atom classes ,
12    this class contains  only an uplink to the containing subgroup
13  class EXP oahAtom : public oahElement
14  /* this class is purely virtual
15  class EXP oahAtomExpectingAValue : public oahAtom
16    a common ancestor for all atom classes
17  /* this class is purely virtual
18    a common ancestor for all atom classes
19  class EXP oahAtomStoringAValue : public oahAtomExpectingAValue
20  /* this class is purely virtual
21  class EXP oahPureHelpAtomWithoutAValue : public oahAtom
22  /* this class is purely virtual
23  class EXP oahPureHelpAtomExpectingAValue : public oahAtomExpectingAValue
24  /* this class is purely virtual
25  class EXP oahSubGroup : public oahElement
26  class EXP oahGroup : public oahElement
27  class EXP oahHandler : public smartable
28   /* this class is purely virtual
29  enum class oahOptionalValuesStyleKind {
30  class EXP oahAtomWithoutAValue : public oahAtom
31  / * this class is purely virtual
```

16.8   Prefixes handling

16.9   `argc/argv` versus `oahOptionsVector`

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

- command line tools get them from `argc/argv` as usual;

- application using the library through the API should place them in an `oahOptionsVector`, defined in `src/mflibrarymfMusicformatsError.h`:

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

Using an `oahOptionsVector` can be done for example:

- in Web sites;

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

In these tools, and `oahOptionsVector` is instantiated ans populated from `argc/argv` with `convertArgcArgvToOption` 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                          string                  serviceName,
5                          const oahOptionsVector& theOptionsVector);
6
7    oahElementHelpOnlyKind
8                        handleOptionsAndArgumentsFromArgcArgv (
9                          int   argc,
10                         char* argv[]);
11
12   virtual void        checkOptionsAndArgumentsConsistency ();
13
14   virtual void        checkOptionsAndArguments () const = 0;
```

16.10 Applying options

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

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

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

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

There are two methods for that:

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

```
```

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

16.11 A OAH atoms collection

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

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class oahAtomsCollection.h
2  class EXP oahAtomAlias : public oahAtom
3  class EXP oahMacroAtom : public oahAtom
4  class EXP oahOptionsUsageAtom : public oahPureHelpAtomWithoutAValue
5  class EXP oahHelpAtom : public oahPureHelpAtomWithoutAValue
6  class EXP oahHelpSummaryAtom : public oahPureHelpAtomWithoutAValue
7  class EXP oahAboutAtom : public oahPureHelpAtomWithoutAValue
8  class EXP oahVersionAtom : public oahPureHelpAtomWithoutAValue
9  class EXP oahContactAtom : public oahPureHelpAtomWithoutAValue
10 class EXP oahBooleanAtom : public oahAtom
11 class EXP oahTwoBooleansAtom : public oahBooleanAtom
12 class EXP oahThreeBooleansAtom : public oahBooleanAtom
13 class EXP oahCombinedBooleansAtom : public oahAtom
```

```
14 class EXP oahCommonPrefixBooleansAtom : public oahAtom
15 class EXP oahIntegerAtom : public oahAtomStoringAValue
16 class EXP oahTwoIntegersAtom : public oahIntegerAtom
17 class EXP oahFloatAtom : public oahAtomStoringAValue
18 class EXP oahStringAtom : public oahAtomStoringAValue
19 class EXP oahFactorizedStringAtom : public oahAtom
20 class EXP oahStringWithDefaultValueAtom : public oahStringAtom
21 class EXP oahRationalAtom : public oahAtomStoringAValue
22 class EXP oahNaturalNumbersSetElementAtom : public oahAtomStoringAValue
23 class EXP oahRGBColorAtom : public oahAtomStoringAValue
24 class EXP oahIntSetElementAtom : public oahAtomStoringAValue
25 class EXP oahStringSetElementAtom : public oahAtomStoringAValue
26 class EXP oahStringToIntMapElementAtom : public oahAtomStoringAValue
27 class EXP oahStringAndIntegerAtom : public oahAtomStoringAValue
28 class EXP oahStringAndTwoIntegersAtom : public oahAtomStoringAValue
29 class EXP oahLengthUnitKindAtom : public oahAtomStoringAValue
30 class EXP oahLengthAtom : public oahAtomStoringAValue
31 class EXP oahMidiTempoAtom : public oahAtomStoringAValue
32 class EXP oahOptionNameHelpAtom : public oahStringWithDefaultValueAtom
33 class EXP oahQueryOptionNameAtom : public oahPureHelpAtomExpectingAValue
34 class EXP oahFindStringAtom : public oahPureHelpAtomExpectingAValue
```

See chapter 37, [The OAH atoms collection], page 237, for more details.

16.12   An option and help example

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

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

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

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

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

And that's it.

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

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

16.13   Options and help introspection

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

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

```
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     4:
13     -sluragn, -slur-all-grace-notes
14     Add a slur to all grace notes
15     5:
16     -beamagn, -beam-all-grace-notes
17     Add a beam to all grace notes
```

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

16.14 Insider versus regular handlers

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

- the full view, named 'insider', contains the options grouped by format or pass;

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

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

Class `oahRegularHandler` relies on the corresponding insider handler:

```
protected:

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

  S_oahHandler          fInsiderHandler;
```

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

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

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

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

  // subgroup

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

  // atoms from the insider handler

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

```

```
39    registerAtomInRegularSubgroup ("display-cpu-usage", subGroup);
40  }
```

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

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

16.15 Deciphering the options and arguments

16.15.1 Options and arguments multi-pass analysis

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

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

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

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

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

Class `oahHandler` contains:

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

16.15.2 Pure help runs

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

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

16.15.3 Applying options

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

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

The heart of it is:

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

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

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

        else {
          // this is an atom

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

          oahElementValueKind
            atomValueKind =
              atom->
                getElementValueKind ();

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

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

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

```cpp
                break;

            case oahElementValueKind::kElementValueImplicit:
              atomExpectingAValue->
                applyAtomWithDefaultValue (
                  gOutputStream);
              break;

            case oahElementValueKind::kElementValueMandatory:
              if (valueUsed.size ()) {
                atomExpectingAValue->
                  applyAtomWithValue (
                    valueUsed,
                    gOutputStream);
              }
              else {
                stringstream s;

                s <<
                  "Atom expecting a value " <<
                  atomExpectingAValue->fetchNamesBetweenQuotes () <<
                  " needs a non-empty value";

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

            case oahElementValueKind::kElementValueOptional:
              if (valueUsed.size ()) {
                atomExpectingAValue->
                  applyAtomWithValue (
                    valueUsed,
                    gOutputStream);
              }
              else {
                atomExpectingAValue->
                  applyAtomWithDefaultValue (
                    gOutputStream);
              }
              break;
          } // switch
        }

        else {
#ifdef TRACING_IS_ENABLED
          if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
            gLogStream <<
              "**** Handling atom not expecting a value:" <<
              endl;

            ++gIndenter;

            gLogStream <<
              atom <<
              endl;

            --gIndenter;
          }
#endif

          atom->
            applyElement (
              gOutputStream);
        }
      }

      // has a help-only been applied?
```

```
125            switch (elementUsed->getElementHelpOnlyKind ()) {
126              case oahElementHelpOnlyKind::kElementHelpOnlyYes:
127                // a help option has been applied
128                this->
129                  setOahHandlerFoundAHelpOption (
130                    elementUsed->
131                      fetchNamesBetweenQuotes ());
132                break;
133              case oahElementHelpOnlyKind::kElementHelpOnlyNo:
134                break;
135            } // switch
136          }
137
138          else {
139            stringstream s;
140
141            s <<
142              "Element from the from the atom uses list for \"" <<
143              nameUsed <<
144              "\" is null";
145
146            oahInternalError (s.str ());
147
148      // ... ... ...
149  }
```

16.15.4   Early handling of some options

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

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

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

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

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

```
1  // ... ... ...
2
3  // insider
4  EXP extern const string K_INSIDER_OPTION_LONG_NAME;
5  EXP extern const string K_INSIDER_OPTION_SHORT_NAME;
6
7  // ... ... ...
8
9  class EXP oahEarlyOptions
10 {
11    // ... ... ...
12
13    public:
14
15      // set and get
16      // ------------------------------------------------------
17
18      void                setEarlyInsiderOption ();
19      Bool                getEarlyInsiderOption () const
```

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

Method oahEarlyOptions::applyEarlyOptionIfRelevant () performs the analysis:

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

16.16   Adding new options

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

- choose a short name and possibly a long name for the option;

- choose an atom class in src/oah/oahAtomsCollection.h/.cpp or write a new one if needed;

- decide in which subgroup and group the option should be made available in an insider OAH group, and create the latter two if needed;

- create a suitable atom and append it to the desired OAH subgroup;

- check the use of the atom wherever needed in the code base;

- add the new atom's long name to the corresponding regular OAH group;

- and last but not least ... test the result.

16.16.1 Representations' vs. passes' options

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

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

16.16.2 Using an existing OAH atom class

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

- class `oahBooleanAtom` was ready to be used;

- it was decided to place it in the global variable `gGlobalOahOahGroup` OAH group, in its `Options help` sub group;

- class `oahOahGroup` in `src/oah/oahOah.h/.cpp` got a new `fReverseNamesDisplayOrder` field:

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

    // ... ... ...

    Bool                    fReverseNamesDisplayOrder;

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

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

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

  // reverse names display order

  fReverseNamesDisplayOrder = false;

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

```
14  R"(Write the short names before the long ones.)",
15          "reverseNamesDisplayOrder",
16          fReverseNamesDisplayOrder));
17
18    // ... ... ...
19  }
```

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

```
1  void oahOahGroup::printOahOahValues (int valueFieldWidth)
2  {
3    gLogStream <<
4      "The basic options are:" <<
5      endl;
6
7    // ... ... ...
8
9    // options and help display
10    // ------------------------------------
11
12    gLogStream << left <<
13      setw (valueFieldWidth) << "Options trace and display:" <<
14      endl;
15
16    ++gIndenter;
17
18    gLogStream << left <<
19      setw (valueFieldWidth) << "fReverseNamesDisplayOrder" << " : " <<
20      fReverseNamesDisplayOrder <<
21      endl <<
22
23    // ... ... ...
```

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

```
1  string oahElement::fetchNames () const
2  {
3    stringstream s;
4
5    if (
6      fShortName.size ()
7          &&
8      fLongName.size ()
9    ) {
10      if (gGlobalOahOahGroup->getReverseNamesDisplayOrder ()) {
11        s <<
12          "-" << fShortName <<
13          ", " <<
14          "-" << fLongName;
15      }
16      else {
17        s <<
18          "-" << fLongName <<
19          ", " <<
20          "-" << fShortName;
21      }
22    }
23
24    else {
25      if (fShortName.size ()) {
26        s <<
27          "-" << fShortName;
28      }
29      if (fLongName.size ()) {
30        s <<
```

```
31          "-" << fLongName;
32      }
33    }
34
35    return s.str ();
36 }
```

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

```
1  void msdl2brailleRegularHandler::createOahRegularGroup ()
2 {
3    // ... ... ...
4
5    registerAtomInRegularSubgroup ("reverse-names-display-order", subGroup);
6
7    // ... ... ...
8 }
```

16.16.3   Creating a new OAH atom class without a value

When class oahHistoryAtom was added to OAH, the first thing has been to add a printHistory () in class mfcMultiComponent in src/mfutilities/mfcBasicTypes.h:

```
1 class mfcMultiComponent : public smartable
2 {
3   // ... ... ...
4
5   public:
6
7     // print
8     // ------------------------------------------------------
9
10    void                  print (ostream& os) const;
11    void                  printHistory (ostream& os) const;
12
13   // ... ... ...
14 };
```

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

```
1 //_____
2 class EXP oahHistoryAtom : public oahPureHelpAtomWithoutAValue
3 {
4   // ... ... ...
5
6   public:
7
8     // public services
9     // ------------------------------------------------------
10
11    void                  applyElement (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 (ostream& os) const override;
27
28     void                    printHistory (ostream& os) const;
29 };
```

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

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

method :: () has the be adapted as:

```
1 void oahHistoryAtom::printHistory (ostream& os) const
2 {
3   // get the handler history
4   S_mfcMultiComponent
5     handlerMultiComponent =
6       fetchAtomHandlerUpLink ()->
7         getHandlerMultiComponent ();
8
9   // sanity check
10   mfAssert (
11     __FILE__, __LINE__,
12     handlerMultiComponent != nullptr,
13     "handlerMultiComponent is null");
14
15   handlerMultiComponent->
16     printHistory (os);
17 }
```

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

```
1   // history
2
3   subGroup->
4     appendAtomToSubGroup (
5       oahHistoryAtom::create (
6         "hist", "history",
7         regex_replace (
8 R"(Display EXECUTABLE_NAME's history.)",
9          regex ("EXECUTABLE_NAME"),
10          serviceName),
11        serviceName));
```

And the new option long name `version` has been added to all regular OAH handlers that already contained the option `-version, -v`, such as in method `xml2lyRegularHandler::createInformationsRegularGroup ()`, alongside the existing option `-version, -v`:

```
1   registerAtomInRegularSubgroup ("version", subGroup);
2   registerAtomInRegularSubgroup ("version-full", subGroup);
3   registerAtomInRegularSubgroup ("history", subGroup);
4   registerAtomInRegularSubgroup ("mf-version", subGroup);
5   registerAtomInRegularSubgroup ("mf-history", subGroup);
```

16.16.4   Creating a new OAH atom class expecting a value

Let's look at how class `oahLengthAtom` is implemented.

class `msrLength` is defined in src/formats/msr/msrBasicTypes.h/.cpp:

```
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
      BLARK
11                                    return
12                                      fLengthUnitKind == other.fLengthUnitKind
13                                      &&
14                                      fLengthValue == other.fLengthValue;
15                                }
16
17      Bool                    operator!= (const msrLength& other) const
18                                { return ! ((*this) == other); }
19
20      void                    convertToLengthUnit (
21                                msrLengthUnitKind lengthUnitKind);
22
23    // ... ... ...
24
25    private:
26
27      // private fields
28      // ------------------------------------------------------
29
30      msrLengthUnitKind    fLengthUnitKind;
31      float                fLengthValue;
32 };
```

Enumeration type 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                                    fSetByUser = true;
19                                 }
20
21     msrLength               getLengthVariable () const
22                                 { return fLengthVariable; }
23
24    public:
25
26     // public services
27     // ----------------------------------------------------------
28
29     void                    applyAtomWithValue (
30                               const string& theString,
31                               ostream&      os) override;
32
33    // ... ... ...
34
35    public:
36
37     // print
38     // ----------------------------------------------------------
39
40    // ... ... ...
41
42     void                    printAtomWithVariableOptionsValues (
43                               ostream& os,
44                               int      valueFieldWidth) const override;
45
46    private:
47
48     // private fields
49     // ----------------------------------------------------------
50
51     msrLength&              fLengthVariable;
52  };
```

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

```
1  void oahLengthAtom::applyAtomWithValue (
2    const string& theString,
3    ostream&      os)
4  {
5    // ... ... ...
6
7    regex e (regularExpression);
8    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        stringstream s;
21        // concatenate the integer and decimal parts
22        s << sm [ 1 ] << sm [ 2 ];
23        s >> floatValue;
24      }
25
26      string lengthUnitName = sm [ 3 ];
27
28      // is lengthUnitName known in the length unit names map?
29      map<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        stringstream s;
38
39        s <<
40          "length unit name \"" << lengthUnitName <<
41          "\" is unknown" <<
42          endl <<
43          "The " <<
44          gGlobalMsrLengthUnitKindsMap.size () <<
45          " known length unit names are:" <<
46          endl;
47
48        ++gIndenter;
49
50        s <<
51          existingMsrLengthUnitKinds (K_NAMES_LIST_MAX_LENGTH);
52
53        --gIndenter;
54
55        oahError (s.str ());
56      }
57
58      setLengthVariable (
59        msrLength (
60          (*it).second,
61          floatValue));
62    }
63
64    else {
65      stringstream s;
66
67      s <<
68        "length value \"" << theString <<
69        "\" for option '" << fetchNames () <<
70        "' is ill-formed";
71
72      oahError (s.str ());
73    }
74  }
```

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

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

Then an option to set the LilyPond paper height can be added to the relevant OAH options group in method `lpsrOahGroup::initializeLpsrPaperOptions ()` in `src/formats/lpsr//lpsrOah.cpp` by:

```
  // paper height

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

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

16.17 Extra options

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

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

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

```
1  #ifdef EXTRA_OAH_IS_ENABLED
2    // create the extra OAH group
3    appendGroupToHandler (
4      createGlobalHarmoniesExtraOahGroup ());
5  #endif
```

Macro `EXTRA_OAH_IS_ENABLED` is defined or not in `src/oah/enableHarmoniesExtraOahIfDesired.h`:

```
1  // comment the following definition if no extra options are wanted
2
3  #ifndef EXTRA_OAH_IS_ENABLED
4    #define EXTRA_OAH_IS_ENABLED
5  #endif
```

16.18   `man` pages generation

MusicFormats can create man pages for its command line tools by browing their OAH hierarchy. This has not been finalized yet.

16.19   Specific global OAH groups

Some informations need to be available globally in the MusicFormats library, such a the conversion date and command line. They are groupeds in `src/oah/generalOah.h/.cpp`:

```
1  class EXP generalOahGroup : public oahGroup
2  {
3    // ... ... ...
4
5    private:
6
7      // translation date
8      // -------------------------------------
9
10     string                fTranslationDateFull;
11     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 cakeicing on the cake independently of any conversion activity:

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/src/oah > grep class harmoniesExtraOah.h
2  class EXP extraShowAllHarmoniesStructuresAtom : public oahAtomStoringAValue
3  class EXP extraShowAllHarmoniesContentsAtom : public oahAtomStoringAValue
4  class EXP extraShowHarmonyDetailsAtom : public oahAtomStoringAValue
5  class EXP extraShowHarmonyAnalysisAtom : public oahAtomStoringAValue
6  class EXP harmoniesExtraOahGroup : public oahGroup
```

```
1  class EXP harmoniesExtraOahGroup : public oahGroup
2  {
3    // ... ... ...
4
5    public:
6
7      // fields
8      // ------------------------------------------------------
9
10     string                    fHarmoniesRootAsString;
11 };
```

16.20 Visiting OAH groups

As an internal representation, OAH can be browsed with the two-phase visitors. This is useful:

- to produce man pages automatically from the options available;

- to create the code that proposes the options to the user in a Web site, also automatically.

Part VI

Representations

Chapter 17

Representations general principles

17.1 Trees vs graphs

17.2 Denormalization

In databases, denormalization means that some data is present in several places. This is usually done for speed, at the cost of making updates more complex, since no such place should be ignored in an update.

MSR uses denormalization explicitly, with smart pointers to class instances being stored in other instances.

In particular, class `msrChord` contains elements that are actually detained by the notes it is composed of:

```
1    // articulations
2    list<S_msrArticulation>
3                        fChordArticulations;
4
5    // spanners
6    list<S_msrSpanner>    fChordSpanners;
7
8    // single tremolo
9    S_msrSingleTremolo    fChordSingleTremolo;
```

This is to avoid having to browse the chord's components to obtain the corresponding information each time it is needed.

All such denormalization is done in MSR internally: the code using MSR does not have to denormalize itself. It can use whichever occurrence of any given denormalized data safely, though.

17.3 Newborn clones

The multi-pass structure of the converters build with `musicformat` leads to a question: should an existing description, such as that of a barLine or a note, be used as is, or should it be built again?

Depending of the kind of description, both possibilities are used:

- the description is used as is if it is shallow, i.e. t doesn't contains smart-pointers to data;

- otherwise, a new description is built, sharing some some non smart-pointers fieds with the existing one. This newborn clone is then populated with whatever is needed.

For example, in src/passes/msr2lpsr/, the S_msrBarLine values found in the MSR data are used also in the LPSR data:

```
void msr2lpsrTranslator::visitStart (S_msrBarLine& elt)
{
#ifdef TRACING_IS_ENABLED
  int inputLineNumber =
    elt->getInputLineNumber ();
#endif

#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrBarLine" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  // ... ... ...

  // append the barLine to the current voice clone
  fCurrentVoiceClone->
    appendBarLineToVoice (elt);
}
```

On the opposite, a new S_msrVoice description is built for use by LPSR: this is how the LilyPond #34 issue is circumvented, adding skip notes where needed in the voices that don't have grace notes at their beginning.

Such new descriptions are created by *NewbornClone () methods, such as:

```
S_msrTuplet msrTuplet::createTupletNewbornClone ()
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceTuplets ()) {
    gLogStream <<
      "Creating a newborn clone of tuplet " <<
      asString () <<
      endl;
  }
#endif

  S_msrTuplet
    newbornClone =
      msrTuplet::create (
        fInputLineNumber,
        fMeasureElementMeasureNumber,
        fTupletNumber,
        fTupletBracketKind,
        fTupletLineShapeKind,
        fTupletShowNumberKind,
        fTupletShowTypeKind,
        fTupletFactor,
        fMemberNotesSoundingWholeNotes,
        fMemberNotesDisplayWholeNotes);

  return newbornClone;
}
```

17.4   Deep clones

Some classes in MusicFormats, such as class `msrVoice` in `src/formats/msr/msrVoices.h/.cpp`, have a `*DeepClone ()` method:

```
SMARTP<msrVoice> createVoiceDeepClone (
                        int          inputLineNumber,
                        msrVoiceKind voiceKind,
                        int          voiceNumber,
                        S_msrStaff   containingStaff);
```

Deep copies of the MSR data is not used currently. This can be changed should the need arise in the future.

17.5   Inheritance

17.5.1   Single inheritance

Many classes in MusicFormats use single inheritance. For example, in `src/formats/msr/msrTimeSignature.h`:

```
class EXP msrTimeSignature : public msrMeasureElement
{
  public:

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

    static SMARTP<msrTimeSignature> create (
                          int                       inputLineNumber,
                          msrTimeSignatureSymbolKind timeSignatureSymbolKind);

    // creation from the applications
    // ------------------------------------------------------

    static SMARTP<msrTimeSignature> createTwoEightsTime (
                          int inputLineNumber);

    // ... ... ...

    // creation from the applications
    // ------------------------------------------------------

    static SMARTP<msrTimeSignature> createTimeFromString (
                          int     inputLineNumber,
                          string timeString);

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

The definitions in in `src/formats/msr/msrTimeSignature.cpp` are:

```
S_msrTimeSignature msrTimeSignature::create (
  int                       inputLineNumber,
  msrTimeSignatureSymbolKind timeSignatureSymbolKind)
{
  msrTimeSignature* o =
    new msrTimeSignature (
      inputLineNumber, timeSignatureSymbolKind);
  assert (o != nullptr);
  return o;
}

msrTimeSignature::msrTimeSignature (
  int                       inputLineNumber,
```

```
14    msrTimeSignatureSymbolKind timeSignatureSymbolKind)
15        : msrMeasureElement (inputLineNumber)
16  {
17    fTimeSignatureSymbolKind = timeSignatureSymbolKind;
18
19    fTimeIsCompound = false;
20  }
```

17.5.2   Single inheritance for smart pointers

All classes for which smart pointers are needed should inherit from class `smartable`, such as in `src/formats/msdl/msd`

```
1  class msdlScanner : public smartable
2  {
3    public:
4
5      // creation
6      // -------------------------------------------------------
7
8      static SMARTP<msdlScanner> create (istream& inputStream);
9
10   public:
11
12      // constructors/destructor
13      // -------------------------------------------------------
14
15                          msdlScanner (istream& inputStream);
16
17    // ... ... ...
18  };
```

This leads to the following in in `src/formats/msdl/msdlScanner.cpp`:

```
1  S_msdlScanner msdlScanner::create (istream& inputStream)
2  {
3    msdlScanner* o =
4      new msdlScanner (inputStream);
5    assert (o != nullptr);
6    return o;
7  }
8
9  msdlScanner::msdlScanner (istream& inputStream)
10     : fInputStream (
11         inputStream),
12       fCurrentToken (
13         ),
14       fCurrentTokenKind (
15         fCurrentToken.getTokenKindToModify ()),
16       fCurrentTokenDescription (
17         fCurrentToken.getTokenDescriptionToModify ())
18  {
19    // trace
20  #ifdef TRACING_IS_ENABLED
21    fTraceTokens        = gGlobalMsdl2msrOahGroup->getTraceTokens ();
22    fTraceTokensDetails = gGlobalMsdl2msrOahGroup->getTraceTokensDetails ();
23  #endif
24
25    // ... ... ...
26  }
```

17.5.3 Multiple inheritance for visitors

Multiple inheritance is used extensively in visitors, which is the way to specify what elements are it seen by the visitor. For example, in src/formats/msr/msr2msrTranslator.h, there is:

```
class EXP msr2msrTranslator :

  public visitor<S_msrScore>,

  // rights

  public visitor<S_msrIdentification>,

  public visitor<S_msrCredit>,
  public visitor<S_msrCreditWords>,

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

Then there are visitStart () and/or visitEnd () methods to handle the corresponding elements:

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

  ++gIndenter;

  // set the current identification
  fCurrentIdentification = elt;

  // store it in the resulting MSR score
  fResultingNewMsrScore->
    setIdentification (
      fCurrentIdentification);

  fOnGoingIdentification = true;
}
```

```
void msr2msrTranslator::visitEnd (S_msrIdentification& elt)
{
  fOnGoingIdentification = false;

  --gIndenter;

#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> End visiting msrIdentification" <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif
}
```

Forgetting to define those visit* () methods causes *no error message whatsoever*: the corresponding elements are just not handled by the visitor.
The visitors trace options are useful to detect such cases:

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

17.5.4   Multiple inheritance in other classes

The only such case is class `mfIndentedOstream` in src/utilities/mfIndentedTextOutput.cpp.h/.cpp:

```
1   class EXP mfIndentedOstream: public 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 (cout);
9
10    myStream <<
11      1 << 2 << 3 << endl <<
12      5 << 6 << endl <<
13      7 << 8 << endl;
14  */
15
16    public:
17
18      // creation
19      // ------------------------------------------------------
20
21      static SMARTP<mfIndentedOstream> create (
22        ostream&         theOStream,
23        mfOutputIndenter& theIndenter)
24      {
25        mfIndentedOstream* o = new mfIndentedOstream (
26          theOStream,
27          theIndenter);
28        assert (o != nullptr);
29
30        return o;
31      }
32
33      // constructors/destructor
34      // ------------------------------------------------------
35
36                          mfIndentedOstream (
37                            ostream&         theOStream,
38                            mfOutputIndenter& theIndenter)
39                            : 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                    incrIdentation ()
62                                  { ++ (fIndentedStreamBuf.getOutputIndenter ()); }
63
64      void                    decrIdentation ()
65                                  { -- (fIndentedStreamBuf.getOutputIndenter ()); }
66
67    private:
68
69      // private fields
70      // --------------------------------------------------------
71
72      // mfIndentedOstream just uses an mfIndentedStreamBuf
73      mfIndentedStreamBuf     fIndentedStreamBuf;
74
75  };
76  typedef SMARTP<mfIndentedOstream> S_indentedOstream;
```

17.5.5 Reversibility

All formats in MusicFormats that can be obtained by a conversion from another one should be convertible
back in the latter, without information loss.
Thus:

- MXSR contains nearly everything that can be described in MusicXML data. The main exception at
  the time of this writing is the MIDI information, see subsection 25.1.1, [MusicXML coverage], page 205;

- MSR contains MusicXML-related informations, so as to convert it back to MXSR;

- LSPR and BSR contain an MSR component. This is why converting those formats back to MSR is
  merely getting the corresponding field.

Chapter 18

Displaying formats

MusicFormats is equipped with option `-display*` options as a help to the maintainer.

18.1 Display categories

18.2 Displaying in practise

```
%void lpsr2lilypondTranslator::generateCodeForNoteRegularInMeasure (
%  S_msrNote note)
%{
%  int inputLineNumber =
%    note->getInputLineNumber ();
%
%#ifdef TRACING_IS_ENABLED
%  if (gGlobalTracingOahGroup->getTraceNotes ()) {
%    stringstream s;
%
%    s <<
%      endl <<
%      "% --> generating code for noteRegularInMeasure " <<
%      note->asString () <<
%      ", line " << inputLineNumber <<
%      endl;
%
%    gLogStream          << s.str ();
%    fLilypondCodeStream << s.str ();
%  }
%#endif
%
%
```

Chapter 19

Music Scores Representation (MSR)

MSR is the central format of music scores in MusicFormats. It contains a very detailed representation of western notation music score elements. Most of it is handling music in a sequential way. See chapter 20, [MSR time-oriented represention], page 186, for a presentation of how it handles time-oriented concerns.

Some of the data in MSR are supplied by the code that uses MSR, as in class `msrSlur`:

```
1    static SMARTP<msrSlur> create (
2                            int              inputLineNumber,
3                            int              slurNumber,
4                            msrSlurTypeKind  slurTypeKind,
5                            msrLineTypeKind  slurLineTypeKind,
6                            msrPlacementKind slurPlacementKind);
7
8    // ... ... ...
9
10   // private fields
11   // --------------------------------------------------------
12
13
14   int                 fSlurNumber;
15
16   msrSlurTypeKind     fSlurTypeKind;
17
18   msrLineTypeKind     fSlurLineTypeKind;
19
20   msrPlacementKind    fSlurPlacementKind;
```

Other data are computed by the MSR private methods. For example, in `src/formats/msr/msrVoices.h`:

```
1    // there can only be 4 regular voices in a staff
2    // (those that can contain beamed notes)
3    // and we need a number for the orientation of beams
4    int                 fRegularVoiceStaffSequentialNumber;
5
6    // ... ... ...
7
8    // fVoiceShortestNoteDuration and fVoiceShortestNoteTupletFactor
9    // are used to compute a number of divisions per quarter note
10   // if needed, such as when generating MusicXML from MSR
11   rational            fVoiceShortestNoteDuration;
12   msrTupletFactor     fVoiceShortestNoteTupletFactor;
```

There are also data that varies during the lifetime of the object, while it is being populated for example. One such case is class `msrMeasure`:

```
1     rational                    fCurrentMeasureWholeNotesDuration;
2                                 // this increases when musical elements
3                                 // are appended to the measure
```

MSR has been designed to be as general as possible, leading it to contain informations fitted to the various textual formats that can be converted to it or output from it by MusicFormats tools.

It is a *very fine-grained* representation of scores:

- some informations it contains are present as such in the textual formats;

- others are computed when the representation is populated, such as, in `src/formats/msr/msrVoices.h`:

```
1     rational                    fVoiceShortestNoteDuration;
```

This information is used when generating MusicXML output to set the `<divisions/>` value.

LPSR and BSR contain an MSR as a sub-component, in order to allow for easy two-way conversion. This avoids the loss of information. This is why converting LPSR and BSR to MSR is done at no cost: just get the MSR component.

Both LPSR and BSR complement their MSR sub-component with whatever is needed for their purpose:

- LPSR contains a description of the structure of the score for the needs of LilyPond output and export from LilyPond when this becomes available;

- BSR contains a description of how to layout the braille cell on the embossed page, in terms of cells per line and lines per page.

19.1   MSR basic types

Some types used thoughout MSR are defined in `src/formats/msr/msrBasicTypes.h/.cpp`:

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr >  egrep -rIn  '^// '
     msrBasicTypes.h
2  msrBasicTypes.h:29:// input line numbers
3  msrBasicTypes.h:34:// names lists max length
4  msrBasicTypes.h:35:// -----------------------------------------------------
5  msrBasicTypes.h:39:// XMLLang
6  msrBasicTypes.h:52:// diatonic pitches
7  msrBasicTypes.h:69:// alterations
8  msrBasicTypes.h:90:// accidentals
9  msrBasicTypes.h:124:// editorial accidentals
10 ... ... ...
11 msrBasicTypes.h:1840:// moments
12 msrBasicTypes.h:1938:// tuplet factors
13 msrBasicTypes.h:2024:// harmonies intervals
14 msrBasicTypes.h:2134:// harmonies structure
15 msrBasicTypes.h:2231:// harmonies contents
16 msrBasicTypes.h:2320:// harmonies details and analysis
17 msrBasicTypes.h:2333:// RGB colors
18 msrBasicTypes.h:2391:// AlphaRGB colors
19 msrBasicTypes.h:2444:// score notation kinds
20 msrBasicTypes.h:2455:// global variables
21 msrBasicTypes.h:2500:// initialization
```

19.2   Data matching across formats

Choices have to be made regarding the way we represent music scores elements, since this varies across formats.

In particular, the way MusicXML structures the elements is not what MSR does. For example, class `msrIdentification` in `src/formats/msr/msrIdentification.h` contains:

```
class EXP msrIdentification : public msrElement
{
  // ... ... ...

  private:

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

    // work
    string              fWorkNumber;

    // ... ... ...

    // creators

    // ... ... ...

    list<string>        fSoftwaresList;

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

This information is stored in distinct elements in MusicXML:

```
<score-partwise>
  <work>
    <work-number>K. 331</work-number>
    <work-title>Piano Sonata in A Major</work-title>
  </work>
  <identification>
    <creator type="composer">Wolfgang Amadeus Mozart</creator>
    <rights>Copyright © 2003 Recordare LLC</rights>
    <encoding>
      <software>Finale 2003 for Windows</software>
      <software>Dolet for Finale 1.3</software>
      <encoding-date>2003-03-14</encoding-date>
    </encoding>
```

The same occurs for MusicXML's `<direction/>` elements, that contain distinct subelements `<words/>` and `<metronome/>`:

```
        <direction>
          <direction-type>
            <words>Adagio</words>
          </direction-type>
          <direction-type>
            <metronome>
              <beat-unit>long</beat-unit>
              <per-minute>100</per-minute>
            </metronome>
          </direction-type>
        </direction>
```

Note that two `<direction-type/>` elements are needed, since only one of `<words/>` and `<metronome/>` can be present in a given instance, as stated in `direction.mod`:

```
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 | 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 41, [Tempos handling], page 268 and section 19.18, [Tempos], page 161 for more details.

19.3   Lengths

There are several cases where a length is used in MSR, hence:

```
1  enum class msrLengthUnitKind {
2    kUnitInch, kUnitCentimeter, kUnitMillimeter
3  };
```

```
1  class EXP msrLength : public smartable
2  {
3      // ... ... ...
4
5      msrLengthUnitKind     fLengthUnitKind;
6      float                 fLengthValue;
```

19.4   Positions in measures and moments

Such positions are represented by rational numbers such as `3/8`, `1/1` being a whole note.

Class `msrMoment` stores a position in a measure, with a relative offset since harmonies can be placed on a note during its sounding time:

```
1      rational              fWrittenPositionInMeseasure;
2      rational              fSoundingRelativeOffset;
```

19.5   Rests and skips

A skip is an invisible rest, i.e. the meaning is the same as that in LilyPond. Skips are created to it fill the holes between notes wherever needed, in order for all voices to be notes/rests/skips sequences.

Skips are also created in `src/passes/msr2lpsr/` to circumvent the LilyPond #34 issue.

19.6   Solo notes and rests

A solo note or rest is characterized as sounding alone in its multi-voice staff for its whole duration.

In the case of a solo rests, such detection allows for better output, in particular when LilyPond code is generated.
An example is at figure 19.1, [The solo rests problem], page 154 : the eighth rests in the second measure of voice 1 of the first staff should be be placed on the middle line of the staff, as `MuseScore` does.

Figure 19.1: The solo rests problem

19.7   Linear versus time-oriented representation

Most music scoring GUI applications handle music as containing voices, which are made of sequences of notes, chords, tuplets and such. This is a horizontal, linear view of the music in the score.

Another view of the music is time-oriented, i.e., what are are notes being played at a given moment in time? This is a vertical view of the music, which is highlighted in piano roll views.

MSR stores descriptions of so-called 'measures slice' through class `msrMeasuresSlice`, defined in `src/formats/msr/m` Then a time-oriented view of a voice, staff or part is a sequence of such measure slices, defined in class `msrMeasuresSlicesSequence`.

An class `msrMeasuresSlice` contains basically a slice measures vector:

```
// the measures in the slice
vector<S_msrMeasure>  fSliceMeasuresVector;
```

From this, the following other descriptions are derived:

```
// notes flat list
list<S_msrNote>       fSliceNotesFlatList;

// note events list
list<S_msrNoteEvent>  fSliceNoteEventsList;

// simultaneous notes chunks list
list<S_msrSimultaneousNotesChunk>
                      fSliceSimultaneousNotesChunksList;
```

Note events are distinguished with enumeration type `msrNoteEventKind`:

```
//_____
enum class msrNoteEventKind {
  kNoteEventStart,
  kNoteEventStop
};
```

Class `msrNoteEvent` contains:

```
rational              fNoteEventPositionInMeasure;
S_msrNote             fNoteEventNote;
msrNoteEventKind      fNoteEventKind;
```

19.8 Spanners

A spanner... spans from one note or rest to another one. A choice to be made about when to use spanners: should wedges < and > be handled as spanners, or simply as being attached to notes? It has been chosen to use spanners only for ligatures apart from true spanners.

MusicXML uses `"start"`, `"start"` and `"start"` attributes, which need to be present in MSR for MusicXML generation. They are reflected in MSR as enumeration type enumeration type `msrSpannerTypeKind`, defined this way:

```
// spanner types
//_____
enum class msrSpannerTypeKind {
  k_NoSpannerType,

  kSpannerTypeStart, kSpannerTypeContinue, kSpannerTypeStop
};
```

19.9 Uplinks and sidelinks

An uplink is a direct pointer from one class instance to one that contains it. class `msrNote` contains:

```
    S_msrChord            fNoteDirectChordUpLink;
    S_msrGraceNotesGroup  fNoteDirectGraceNotesGroupUpLink;
    S_msrTuplet           fNoteDirectTupletUpLink;
    S_msrMeasure          fNoteDirectMeasureUpLink;
```

A sidelink is used in ligatures and spanners, so that each end of the structure can reference the other one:

```
msrLigatures.h:    S_msrLigature           fLigatureOtherEndSideLink; // two-way
```

```
msrSpanners.h:    S_msrSpanner            fSpannerOtherEndSideLink; // two-way
```

19.10 Sounding and displayed durations

All durations are represented by rational numbers whose denominators are powers of 2, such as `rational (3, 16`, and relative to the duration of a whole note.

This information is a field of class `msrMeasureElement`:

```
    rational                fMeasureElementSoundingWholeNotes;
```

In a tuplet, the sounding durations is different than the written durations, so we store the sounding

```
    // whole notes
    rational              fNoteDisplayWholeNotes;

    int                   fNoteDotsNumber;

    msrDurationKind       fNoteGraphicDurationKind;

    msrTupletFactor       fNoteTupletFactor;

    msrQuarterTonesPitchKind
                          fNoteQuarterTonesDisplayPitchKind;
    msrOctaveKind         fNoteDisplayOctaveKind;
                                // for unpitched notes
                                // and pitched rests
```

19.11 Printing descriptions

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

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

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

There are also more specific methods such as:

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

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

Note that:

- virtual method `asString ()` produces a rather condensed view of the data to be displayed as part of a single line;

- virtual method `print ()` may produce its output on multiples lines, which always ends with an end of line.

Most classes in MusicFormats can be printed with the `<<` operator:

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

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

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

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

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

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

  return s.str ();
}
```

A typical sequence to produce indented output is:

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

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

  ++gIndenter;

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

  --gIndenter;
}
```

The main indented output streams are:

```cpp
#define gOutputStream *gGlobalOutputIndentedOstream
#define gLogStream    *gGlobalLogIndentedOstream
```

19.12   Pitches

MSR handle diatonic, semitone and quarter tone pitches, defined in `src/formats/msr/msrBasicTypes.h` as shown below. All pitches data is represented internally as quater tones pitches, and conversions are done wherever needed.

```cpp
// diatonic pitches
//_____
enum class msrDiatonicPitchKind {
  k_NoDiatonicPitch,

  // starting at C for LilyPond relative octave calculations
  kDiatonicPitchC,
  kDiatonicPitchD, kDiatonicPitchE, kDiatonicPitchF,
  kDiatonicPitchG, kDiatonicPitchA, kDiatonicPitchB
};
```

```cpp
// semi tones pitches
//_____
enum class msrSemiTonesPitchKind {
  kSTP_NoSemiTonesPitch,

  kSTP_C_TripleFlat,
  kSTP_C_DoubleFlat, kSTP_C_Flat,
  kSTP_C_Natural,
  kSTP_C_Sharp, kSTP_C_DoubleSharp,
  kSTP_C_TripleSharp,
```

```
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_DoubleleFlat , 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    k_NoQuarterTonesPitch ,
5
6    kQTP_Rest , kQTP_Skip ,
7
8    kQTP_A_TripleFlat ,
9    kQTP_A_DoubleFlat , kQTP_A_SesquiFlat , kQTP_A_Flat , kQTP_A_SemiFlat ,
10   kQTP_A_Natural ,
11   kQTP_A_SemiSharp , kQTP_A_Sharp , kQTP_A_SesquiSharp , kQTP_A_DoubleSharp ,
12   kQTP_A_TripleSharp ,
13
14   kQTP_B_TripleFlat ,
15   kQTP_B_DoubleFlat , kQTP_B_SesquiFlat , kQTP_B_Flat , kQTP_B_SemiFlat ,
16   kQTP_B_Natural ,
17   kQTP_B_SemiSharp , kQTP_B_Sharp , kQTP_B_SesquiSharp , kQTP_B_DoubleSharp ,
18   kQTP_B_TripleSharp ,
19
20   kQTP_C_TripleFlat ,
21   kQTP_C_DoubleFlat , kQTP_C_SesquiFlat , kQTP_C_Flat , kQTP_C_SemiFlat ,
22   kQTP_C_Natural ,
23   kQTP_C_SemiSharp , kQTP_C_Sharp , kQTP_C_SesquiSharp , kQTP_C_DoubleSharp ,
24   kQTP_C_TripleSharp ,
25
26   kQTP_D_TripleFlat ,
27   kQTP_D_DoubleFlat , kQTP_D_SesquiFlat , kQTP_D_Flat , kQTP_D_SemiFlat ,
28   kQTP_D_Natural ,
29   kQTP_D_SemiSharp , kQTP_D_Sharp , kQTP_D_SesquiSharp , kQTP_D_DoubleSharp ,
```

```
30    kQTP_D_TripleSharp ,
31
32    kQTP_E_TripleFlat ,
33    kQTP_E_DoubleFlat , kQTP_E_SesquiFlat , kQTP_E_Flat , kQTP_E_SemiFlat ,
34    kQTP_E_Natural ,
35    kQTP_E_SemiSharp , kQTP_E_Sharp , kQTP_E_SesquiSharp , kQTP_E_DoubleSharp ,
36    kQTP_E_TripleSharp ,
37
38    kQTP_F_TripleFlat ,
39    kQTP_F_DoubleFlat , kQTP_F_SesquiFlat , kQTP_F_Flat , kQTP_F_SemiFlat ,
40    kQTP_F_Natural ,
41    kQTP_F_SemiSharp , kQTP_F_Sharp , kQTP_F_SesquiSharp , kQTP_F_DoubleSharp ,
42    kQTP_F_TripleSharp ,
43
44    kQTP_G_TripleFlat ,
45    kQTP_G_DoubleFlat , kQTP_G_SesquiFlat , kQTP_G_Flat , kQTP_G_SemiFlat ,
46    kQTP_G_Natural ,
47    kQTP_G_SemiSharp , kQTP_G_Sharp , kQTP_G_SesquiSharp , kQTP_G_DoubleSharp ,
48    kQTP_G_TripleSharp
49 };
```

19.13   Octaves

They are represented with enumeration type :

```
1  // octaves
2  //_____
3  enum class msrOctaveKind {
4    k_NoOctave ,
5
6    kOctave0 , kOctave1 , kOctave2 , kOctave3 ,
7    kOctave4 , // that of middle C
8    kOctave5 , kOctave6 , kOctave7 , kOctave8 , kOctave9
9  };
```

For the needs of LilyPond and MSDL, MSR also contains a description of how to enter octaves:

```
1  // octave entry
2  //_____
3  enum class msrOctaveEntryKind {
4    kOctaveEntryRelative ,
5    kOctaveEntryAbsolute ,
6    kOctaveEntryFixed
7  };
```

19.14   Durations

MusicFormats represents durations with enumeration type `msrDurationKind`, defined in `src/formats/msr/msrBasic`

```
1  // durations
2  //_____
3  enum class msrDurationKind {
4    k_NoDuration ,
5
6    // from longest to shortest for the algorithms
7    kMaxima , kLonga , kBreve , kWhole , kHalf ,
8    kQuarter ,
9    kEighth , k16th , k32nd , k64th , k128th , k256th , k512th , k1024th
10 };
```

19.15   Alterations

```
1  // alterations
2  //_____
3  enum class msrAlterationKind {
4    k_NoAlteration ,
5
6    kAlterationTripleFlat , kAlterationDoubleFlat , kAlterationSesquiFlat ,
7    kAlterationFlat , kAlterationSemiFlat ,
8    kAlterationNatural ,
9    kAlterationSemiSharp , kAlterationSharp , kAlterationSesquiSharp ,
10   kAlterationDoubleSharp , kAlterationTripleSharp
```

19.16   Accidentals

```
1  // accidentals
2  //_____
3  enum class msrAccidentalKind {
4    kAccidentalNone ,
5
6    kAccidentalSharp , kAccidentalNatural ,
7    kAccidentalFlat , kAccidentalDoubleSharp ,
8    kAccidentalSharpSharp ,
9    kAccidentalFlatFlat , kAccidentalNaturalSharp ,
10   kAccidentalNaturalFlat , kAccidentalQuarterFlat ,
11   kAccidentalQuarterSharp ,kAccidentalThreeQuartersFlat ,
12   kAccidentalThreeQuartersSharp ,
13
14   kAccidentalSharpDown , kAccidentalSharpUp ,
15   kAccidentalNaturalDown , kAccidentalNaturalUp ,
16   kAccidentalFlatDown , kAccidentalFlatUp ,
17   kAccidentalTripleSharp , kAccidentalTripleFlat ,
18   kAccidentalSlashQuarterSharp , kAccidentalSlashSharp ,
19   kAccidentalSlashFlat , kAccidentalDoubleSlashFlat ,
20   kAccidentalSharp_1 , kAccidentalSharp_2 ,
21   kAccidentalSharp_3 ,  kAccidentalSharp_5 ,
22   kAccidentalFlat_1 , kAccidentalFlat_2 ,
23   kAccidentalFlat_3 , kAccidentalFlat_4 ,
24   kAccidentalSori , kAccidentalKoron ,
25
26   kAccidentalOther
27 };
```

19.17   Durations

They are represented in MSR with the enumeration type `msrDurationKind` enumeration type, defined in `src/formats/msr/msrBasicTypes.h`:

```
1  // durations
2  //_____
3  enum class msrDurationKind {
4    k_NoDuration ,
5
6    // from longest to shortest for the algorithms
7    kMaxima , kLonga , kBreve , kWhole , kHalf ,
8    kQuarter ,
9    kEighth , k16th , k32nd , k64th , k128th , k256th , k512th , k1024th
10 };
```

Class `msrDottedDuration` contains:

```
msrDurationKind        fDurationKind;
int                    fDotsNumber;
```

19.18   Tempos

There are thus several kinds of tempos in MSR, with variants represented by enumeration type `msrTempoKind` in `src/formats/msr/msrTempos.h`:

```
class EXP msrTempo : public msrMeasureElement
{
  public:

    // data types
    // ------------------------------------------------------

    enum msrTempoKind {
      k_NoTempoKind,
      kTempoBeatUnitsWordsOnly,
      kTempoBeatUnitsPerMinute,
      kTempoBeatUnitsEquivalence,
      kTempoNotesRelationship
    };

    // ... ... ...

    enum msrTempoParenthesizedKind {
      kTempoParenthesizedYes, kTempoParenthesizedNo
    };

    // ... ... ...

    enum msrTempoNotesRelationshipKind {
      kTempoNotesRelationshipNone, kTempoNotesRelationshipEquals
    };

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

19.18.1   Tempos notes

A tempo indication can contain a note a notes in a tuplet. Such notes are described by class `msrTempoNote`:

```
class EXP msrTempoNote : public msrElement
{
  public:

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

    static SMARTP<msrTempoNote> create (
                          int            inputLineNumber,
                          const rational& tempoNoteWholeNotes,
                          Bool           tempoNoteBelongsToATuplet);

  protected:

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

```
18                            msrTempoNote (
19                              int              inputLineNumber,
20                              const rational& tempoNoteWholeNotes,
21                              Bool             tempoNoteBelongsToATuplet);
22
23     // ... ... ...
24
25   private:
26
27     // private fields
28     // ------------------------------------------------------
29
30     rational              fTempoNoteWholeNotes;
31
32     list<S_msrBeam>       fTempoNoteBeams;
33
34     Bool                  fTempoNoteBelongsToATuplet;
35 };
```

19.18.2   Tempos tuplets

A tuplet in a tempo representation is described by class `msrTempoTuplet`:

```
1  //_____
2  class EXP msrTempoTuplet : public msrElement
3  {
4    public:
5
6      // data types
7      // ------------------------------------------------------
8
9      enum msrTempoTupletTypeKind {
10       kTempoTupletTypeNone,
11       kTempoTupletTypeStart, kTempoTupletTypeStop
12     };
13
14     // ... ... ...
15
16     enum msrTempoTupletBracketKind {
17       kTempoTupletBracketYes, kTempoTupletBracketNo
18     };
19
20     // ... ... ...
21
22     enum msrTempoTupletShowNumberKind {
23       kTempoTupletShowNumberActual,
24       kTempoTupletShowNumberBoth,
25       kTempoTupletShowNumberNone
26     };
27
28     // ... ... ...
29
30     // creation from MusicXML
31     // ------------------------------------------------------
32
33     static SMARTP<msrTempoTuplet> create (
34       int                          inputLineNumber,
35       int                          tempoTupletNumber,
36       msrTempoTupletBracketKind    tempoTupletBracketKind,
37       msrTempoTupletShowNumberKind tempoTupletShowNumberKind,
38       msrTupletFactor              tempoTupletFactor,
39       rational                     memberNotesDisplayWholeNotes);
40
41   protected:
```

```
42
43      // constructors/destructor
44      // ------------------------------------------------------
45
46      msrTempoTuplet (
47        int                       inputLineNumber ,
48        int                       tempoTupletNumber ,
49        msrTempoTupletBracketKind    tempoTupletBracketKind ,
50        msrTempoTupletShowNumberKind tempoTupletShowNumberKind ,
51        msrTupletFactor              tempoTupletFactor ,
52        rational                     memberNotesDisplayWholeNotes );
53
54      // ... ... ...
55
56   private :
57
58      // private fields
59      // ------------------------------------------------------
60
61      int                    fTempoTupletNumber ;
62
63      msrTempoTupletBracketKind
64                             fTempoTupletBracketKind ;
65
66      msrTempoTupletShowNumberKind
67                             fTempoTupletShowNumberKind ;
68
69      msrTupletFactor        fTempoTupletFactor ;
70
71      rational               fMemberNotesDisplayWholeNotes ;
72
73      rational               fTempoTupletDisplayWholeNotes ;
74
75      list < S_msrElement >   fTempoTupletElements ;
76 };
```

19.18.3 Tempos description

The private fields in class `msrTempo` are:

```
1  class EXP msrTempo : public msrMeasureElement
2  {
3    // ... ... ...
4
5    private :
6
7      // private fields
8      // ------------------------------------------------------
9
10     msrTempoKind         fTempoKind ;
11
12     list < S_msrWords >   fTempoWordsList ;
13
14     msrDottedDuration    fTempoBeatUnit ;
15
16     string               fTempoPerMinute ; // '90' or '132-156' for example
17     msrDottedDuration    fTempoEquivalentBeatUnit ;
18
19     S_msrTempoNotesRelationshipshipElements
20                          fTempoNotesRelationshipLeftElements ;
21     msrTempoNotesRelationshipKind   fTempoNotesRelationshipKind ;
22     S_msrTempoNotesRelationshipshipElements
23                          fTempoNotesRelationshipRightElements ;
24
```

```
25     msrTempoParenthesizedKind
26                         fTempoParenthesizedKind ;
27
28     msrPlacementKind        fTempoPlacementKind ;
29 };
```

Among these fields:

- field `msrTempo::fTempoKind` denotes the variant;

- field `msrTempo::fTempoWordsList` contains the words that can be present, such as 'adagio molto';

- field `msrTempo::fTempoBeatUnit` is a dotted duration, as in '4.';

- field `msrTempo::fTempoPerMinute` is a string, since it can contain ranges indication as in '4. = 60-66';

- field `msrTempo::fTempoEquivalentBeatUnit` is a dotted duration;

- field `msrTempo::fTempoNotesRelationshipLeftElements`, field `msrTempo::fTempoNotesRelationshipKind`
  and field `msrTempo::fTempoNotesRelationshipRightElements` are used when a relationship is present,
  such as '2. = 1', in which case field `msrTempo::fTempoNotesRelationshipKind` contains field `msrTempo::kTemp`

- field `msrTempo::fTempoParenthesizedKind` indicates whether the tempo indication is parenthesized;

- field `msrTempo::fTempoPlacementKind` tells whether the tempo is to be places above or below the
  staff, constant `msrPlacementKind::kPlacementAbove` by default.

19.19   Clefs

Clefs are distinguished using enumeration type `msrClefKind`:

```
1  // clefs
2  //_____
3
4  enum class msrClefKind {
5    k_NoClef ,
6
7    kClefTreble ,
8    kClefSoprano , kClefMezzoSoprano , kClefAlto , kClefTenor , kClefBaritone , kClefBass ,
9    kClefTrebleLine1 ,
10   kClefTrebleMinus15 , kClefTrebleMinus8 , kClefTreblePlus8 , kClefTreblePlus15 ,
11
12   kClefBassMinus15 , kClefBassMinus8 , kClefBassPlus8 , kClefBassPlus15 ,
13
14   kClefVarbaritone ,
15
16   kClefTablature4 , kClefTablature5 , kClefTablature6 , kClefTablature7 ,
17
18   kClefPercussion ,
19
20   kClefJianpu
21 };
```

Class `msrClef` contains:

```
1     msrClefKind          fClefKind ;
2     int                  fClefStaffNumber ;
```

19.20　Keys

MSR, as MusicXML, suports Humdrum-Scot keys as well as traditional key such as `C` and `6/8`.

A Humdrum-Scot key is composed of items represented by class **msrHumdrumScotKeyItem**, each containing:

```
msrDiatonicPitchKind    fKeyDiatonicPitchKind;
msrAlterationKind       fKeyAlterationKind;
msrOctaveKind           fKeyOctaveKind;
```

An example is at figure 19.2, [Humdrum-Scot keys], page 165. It has been produced by:

```
xml2ly -auto-output-file-name keys/HumdrumScotKeys.xml
```
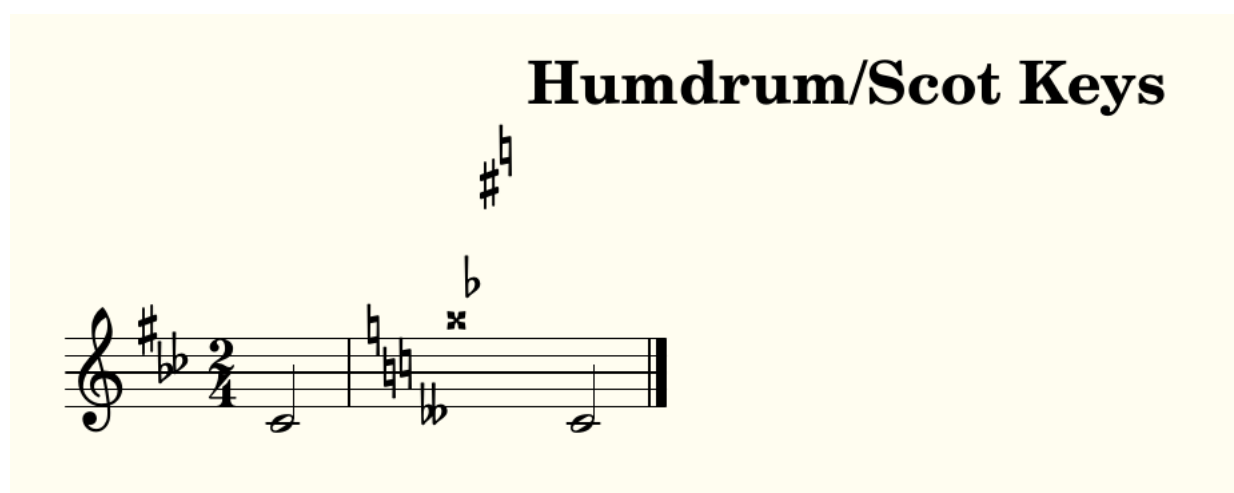

Figure 19.2: Humdrum-Scot keys

Class `msrKey` thus contains:

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

msrKeyKind              fKeyKind;

// traditional keys

msrQuarterTonesPitchKind
                       fKeyTonicQuarterTonesPitchKind;
msrModeKind            fModeKind;
int                    fKeyCancel;

// Humdrum/Scot keys

vector<S_msrHumdrumScotKeyItem>
                       fHumdrumScotKeyItemsVector;
Bool                   fKeyItemsOctavesAreSpecified;
```

19.21　Time signatures

The variants in time signatures are distinguished by enumeration type enumeration type **msrTimeSignatureSymbolKir**

```
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      vector<int>              fTimeSignatureBeatsNumbersVector; // 5+3+1 is possible
2      int                      fTimeSignatureBeatValue;
```

Class `msrTimeSignature` contains:

```
1      msrTimeSignatureSymbolKind
2                           fTimeSignatureSymbolKind;
3
4      vector<S_msrTimeSignatureItem>
5                           fTimeSignatureItemsVector;
6
7      // a time is compound if it contains several items
8      // or if the only one has several beats numbers
9      // i.e. 3/4 is not, (3+4)/8 is, and 2/4+3/4 is too
10     Bool                 fTimeIsCompound;
```

19.22 MSR classes inheritance

The picture at figure 19.3, [The MSR classes hierarchy], page 167, shows the hierarchy of the main MSR classes. The colors are used as follows:

The background colors are used as follows:

- green : a score element that is expected to be found in a score representation, such as class `msrStaff` and class `msrChord`;

Figure 19.3: The MSR classes hierarchy
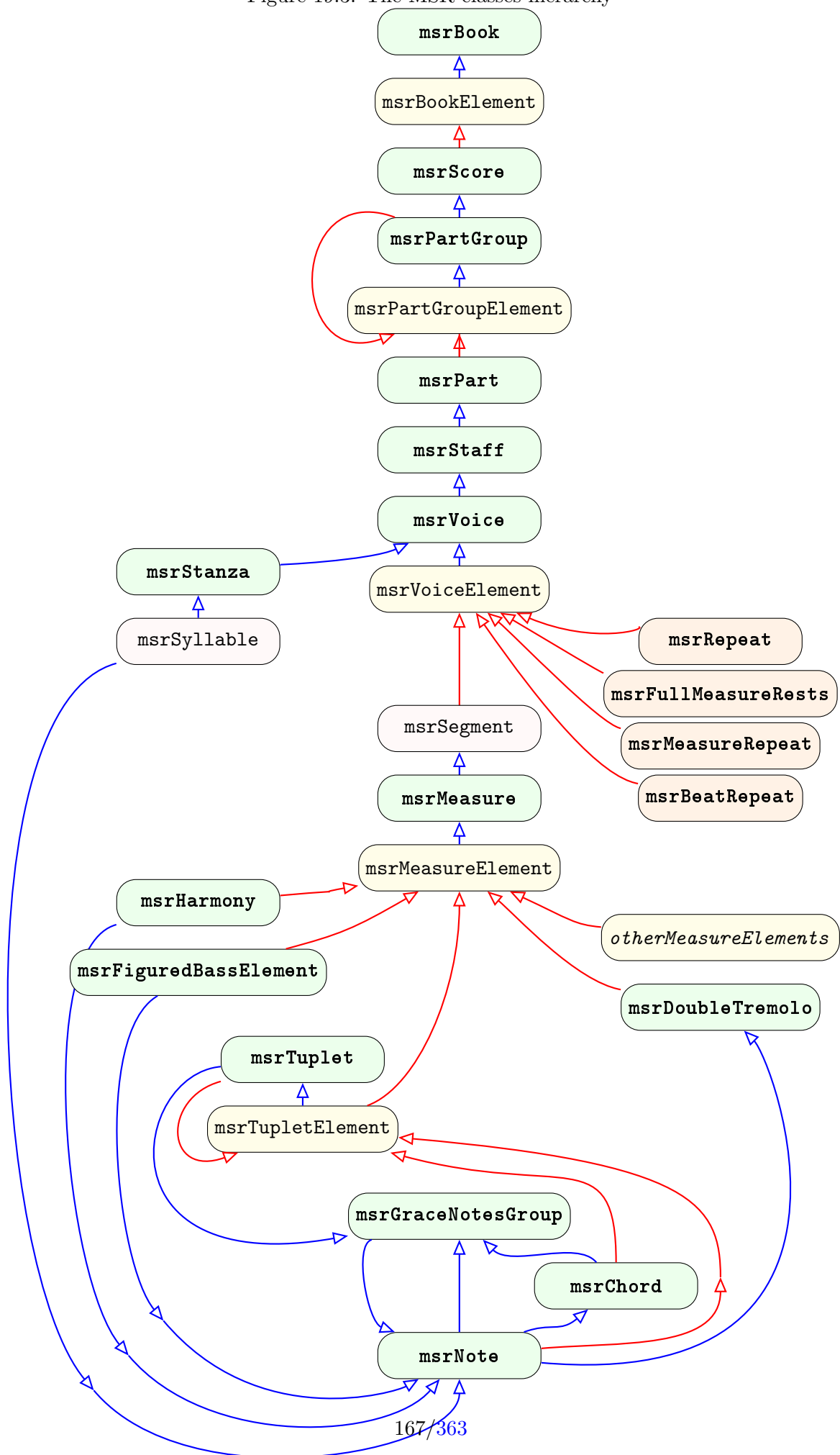
- pink : a element needed in MSR to structure the representation, such as class `msrSegment` and class `msrSyllable`;

- yellow : a base class with name class `msr*Element` for elements that can be used in another class, such as class `msrVoiceElement`;

The arrows colors have the following meaning:

- red: a link from a class to its base class. For example, class `msrPart` is derived from class `msrPartGroupElement`;

- blue: a link from a class to another that uses smart pointers to one or more instances of the former. For example, an class `msrTuplet` may be an element of an class `msrGraceNotesGroup`.

When not shown for clarity, the common base class of all these classes is class `msrElement`, that contains an integer input line number.

The `otherMeasureElements` classes are:

- bars:

  - class `msrBarCheck`
  - class `msrBarNumberCheck`
  - class `msrBarLine`
  - class `msrHiddenMeasureAndBarLine`

- breaks:

  - class `msrLineBreak`
  - class `msrPageBreak`

- notes:

  - class `msrVoiceStaffChange`
  - class `msrOctaveShift`

- clefs, keys, times, tempo:

  - class `msrClef`
  - class `msrKey`
  - class `msrTime`
  - class `msrTempo`

- instruments:

  - class `msrStaffDetails`
  - class `msrScordatura`
  - class `msrAccordionRegistration`
  - class `msrHarpPedalsTuning`
  - class `msrPedal`
  - class `msrDamp`
  - class `msrDampAll`

- lyrics:

  - class `msrSyllable`

- rehearsals, segno and coda:

  - class `msrRehearsalMark`
  - class `msrSegno`
  - class `msrDalSegno`
  - class `msrCoda`

- others:

  - class `msrPrintLayout`
  - class `msrEyeGlasses`
  - class `msrStaffLevelElement`
  - class `msrTranspose`
  - class `msrTupletElement`

19.23 Books

Books handling is presented at section 53, [Books handling], page 290.

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    set<S_msrBookElement> fBookElementsSet;
3
4    list<S_msrBookElement>fBookElementsList;
```

Currently, the only book element used is the class `msrScore`, but others might come, such as texts, which LilyPond allows as `\markup {...}`:

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrBook' *
2  formats/msr/msrScores.h:class EXP msrScore : public msrBookElement
```

19.24 Scores

Scores handling is presented at section **??**, [Scores handling], page **??**.

A score in MSR is the usual music score concept. It contains a set and a list of `S_msrPartGroup`:

```
1    // part groups
2    set<S_msrPartGroup>   fScorePartGroupsSet;
3
4    list<S_msrPartGroup>  fPartGroupsList;
```

19.25 Part groups

Part groups handling is presented at section 19.25, [Part groups], page 170.

A part group in MSR contains parts or other part groups. This concept is recursive, as it is in music score: the winds part group can oboes and horns part group, for example. An implicit part group exists in MSR if the score does not contain explicit part groups.

An class `msrPartGroup` thus contains parts and part groups in any order, as is found in symphonic music scores:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrPartGroupElement' *
formats/msr/msrParts.h:class EXP msrPart : public msrPartGroupElement
formats/msr/msrPartGroups.h:class EXP msrPartGroup : public msrPartGroupElement
```

which are stored in a list:

```
    // allowing for both parts and (sub-)part groups as elements
    list<S_msrPartGroupElement>
                        fPartGroupElementsList;
```

19.26 Parts

Parts handling is presented at section 51, [Parts handling], page 287.

A part in MSR is composed of voices, stored in:

```
    // staves

    map<int, S_msrStaff>  getPartStaveNumbersToStavesMap;
    list<S_msrStaff>      fPartAllStavesList;

    // harmonies

    S_msrStaff            fPartHarmoniesStaff;
    S_msrVoice            fPartHarmoniesVoice;

    // figured bass

    S_msrStaff            fPartFiguredBassStaff;
    S_msrVoice            fPartFiguredBassVoice;

    // voices

    list<S_msrVoice>      fPartAllVoicesList;
```

19.27 Staves

Staves handling is presented at section ??, [Staves handling], page ??.

A stave contains at most 4 numbered voices, stored in:

```
1      // the mapping of all the voices in the staff,
2      // including harmonies and figured bass voices
3      map<int, S_msrVoice>  fStaffVoiceNumbersToAllVoicesMap;
4
5      // the mapping of voice numbers to regular voices
6      map<int, S_msrVoice>  fStaffVoiceNumbersToRegularVoicesMap;
7
8      // we need to handle the regular voice specifically
9      // to assign them sequencing numbers from 1 to gMaxStaffVoices,
10     // needed to set the beams orientation (up or down)
11     int                    fStaffRegularVoicesCounter;
12
13     // harmonies and figured bass elements should be placed %%%JMI
14     // in the first regular voice of the staff, hence:
15     list<S_msrVoice>       fStaffRegularVoicesList;
16
17     // we need to sort the voices by increasing voice numbers,
18     // but with harmonies voices right before the corresponding regular voices
19     list<S_msrVoice>       fStaffAllVoicesList;
```

19.28 Voice elements

Voices contain instances of class `msrVoiceElement`, defined in `src/formats/msr/msrVoiceElements.h/.cpp`:

```
1  //------------------------------------------------------------------------------
2  /*
3    Various elements can found in voices,
4    hence class msrVoiceElement
5  */
6
7  class EXP msrVoiceElement : public msrElement
8  {
9    public:
10
11      // creation from MusicXML
12      // ------------------------------------------------------
13
14      // cloning
15      // ------------------------------------------------------
16
17    protected:
18
19                        msrVoiceElement (
20                          int inputLineNumber);
21
22      virtual           ~msrVoiceElement ();
23
24    /*
25      The voice uplink is declared in the sub-classes,
26      to allow for separate *.h files, C++ constraint
27    */
28  };
```

The classes derived from class `msrVoiceElement` are:

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep 'public msrVoiceElement
        ' *.h
2  msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
3  msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
4  msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
5  msrFullMeasureRests.h:class EXP msrFullMeasureRests : public msrVoiceElement
6  msrSegments.h:class EXP msrSegment : public msrVoiceElement
```

They are describes in specific sections below.

19.29   Voices

Voices handling is presented at section 49, [Voices handling], page 285.

A voice is conceptually a sequence of `S_msrVoiceElement`, that may be:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > grep -r 'public msrVoiceElement' *
formats/msr/msrMeasureRepeats.h:class EXP msrMeasureRepeat : public msrVoiceElement
formats/msr/msrRepeats.h:class EXP msrRepeat : public msrVoiceElement
formats/msr/msrFullMeasureRests.h:class EXP msrFullMeasureRests : public msrVoiceElement
formats/msr/msrBeatRepeats.h:class EXP msrBeatRepeat : public msrVoiceElement
formats/msr/msrSegments.h:class EXP msrSegment : public msrVoiceElement
```

More precisely and for technical reasons, an class `msrVoice` contains:

```
    // voice initial elements list

    list<S_msrVoiceElement>
                        fVoiceInitialElementsList;

    // voice first and last segments

    // fVoiceLastSegment contains the music
    // not yet stored in fVoiceInitialElementsList,
    // it is thus logically the end of the latter,
    // and is created implicitly for every voice.
    // It is needed 'outside' of the 'list<S_msrElement>'
    // because it is not a mere S_msrElement, but a S_msrSegment
    S_msrSegment        fVoiceLastSegment;

    // fVoiceFirstSegment is used to work around LilyPond issue #34
    S_msrSegment        fVoiceFirstSegment;
```

Each voice is described by a field of enumeration type `msrVoiceKind`, defined in
`src/formats/msr/msrBasicTypes.h`:

```
enum class msrVoiceKind {
  kVoiceKindRegular,
  kVoiceKindDynamics,
  kVoiceKindHarmonies,   // for MusicXML <harmony/>, LilyPond ChordNames
  kVoiceKindFiguredBass // for MusicXML <figured-bass/>, LilyPond FiguredBass
};
```

As stated in the comment above, `fVoiceLastSegment` is used because it because `fVoiceInitialElementsList` can contain any class `msrVoiceElement`, whereas all MSR elements appended to the voice are to be placed in a segment.

An class `msrSegment` instance should thus be created and stored in `fVoiceLastSegment` before class `msrVoiceElement` instances can be appended to the voice.

When repeats are handled, an class `msrRepeat` instance is created. Then the contents of field `msrVoice::fVoiceLastS` is moved into it and a new segment is created, see section 19.36, [Repeats], page 178.

Wether the last segment should be created right when the voice is created is controlled with enumeration type `msrVoiceCreateInitialLastSegmentKind`, defined in `src/formats/msr/msrVoices.h`:

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

19.30 Measures

Measures handling is presented at section 38, [Measures handling], page 242.

A measure is a linear, flat sequence of class `msrMeasureElements`, some of which are structured, such as class `msrChord`. Class `msrMeausre` is defined in `src/formats/msr/msrMeausre.h/.cpp`.

The measure elements are defined in `src/formats/msr/`:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep  'public
    msrMeasureElement' *.h
msrBars.h:class EXP msrBarCheck : public msrMeasureElement
msrBars.h:class EXP msrBarNumberCheck : public msrMeasureElement
msrBars.h:class EXP msrBarLine : public msrMeasureElement
msrBreaks.h:class EXP msrLineBreak : public msrMeasureElement
msrBreaks.h:class EXP msrPageBreak : public msrMeasureElement
msrClefs.h:class EXP msrClef : public msrMeasureElement
msrCodas.h:class EXP msrCoda : public msrMeasureElement
msrDoubleTremolos.h:class EXP msrDoubleTremolo : public msrMeasureElement
msrEyeGlasses.h:class EXP msrEyeGlasses : public msrMeasureElement
msrFiguredBassElements.h:class EXP msrFiguredBassElement : public msrMeasureElement
msrHarmonies.h:class EXP msrHarmony : public msrMeasureElement
msrHiddenMeasureAndBarLines.h:class EXP msrHiddenMeasureAndBarLine : public
    msrMeasureElement
msrInstruments.h:class EXP msrScordatura : public msrMeasureElement
msrInstruments.h:class EXP msrAccordionRegistration : public msrMeasureElement
msrInstruments.h:class EXP msrHarpPedalsTuning : public msrMeasureElement
msrInstruments.h:class EXP msrPedal : public msrMeasureElement
msrInstruments.h:class EXP msrDamp : public msrMeasureElement
msrInstruments.h:class EXP msrDampAll : public msrMeasureElement
msrKeys.h:class EXP msrKey : public msrMeasureElement
msrLyrics.h:class EXP msrSyllable : public msrMeasureElement
msrMusicXMLSpecifics.h:class EXP msrPrintLayout : public msrMeasureElement
msrRehearsalMarks.h:class EXP msrRehearsalMark : public msrMeasureElement
msrSegnos.h:class EXP msrSegno : public msrMeasureElement
msrSegnos.h:class EXP msrDalSegno : public msrMeasureElement
msrStavesDetails.h:class EXP msrStaffDetails : public msrMeasureElement
msrTempos.h:class EXP msrTempo : public msrMeasureElement
msrTimeSignatures.h:class EXP msrTimeSignature : public msrMeasureElement
msrTranspositions.h:class EXP msrOctaveShift : public msrMeasureElement
msrTranspositions.h:class EXP msrTranspose : public msrMeasureElement
msrTupletElements.h:class EXP msrTupletElement : public msrMeasureElement
msrVoiceStaffChanges.h:class EXP msrVoiceStaffChange : public msrMeasureElement
```

In order to perform a time-wise analysis of the scores, MSR contains class `msrMeaussre` linear flat lists, without the class `msrRepeat` and such being represented.
This is used when identifying rest notes that are not 'heard' simultaneously with other notes or rests: this way, the rest can ignore the current voice number and be placed in the vertical middle of the staff.

Apart from the cloning methods, only one method creates measures, namely method `msrSegment::createAMeasureAndAppendItToSegment ()`, defined in `src/formats/msr/msrSegments.h/.cpp`:

```
S_msrMeasure msrSegment::createAMeasureAndAppendItToSegment (
  int     inputLineNumber,
  string measureNumber,
  msrMeasureImplicitKind
        measureImplicitKind)
{
  // ... ... ...

  ++gIndenter;

  // determine new measure 'first in segment' kind
```

```
12    msrMeasure::msrMeasureFirstInSegmentKind
13       measureFirstInSegmentKind;
14
15    if (fSegmentMeasuresList.size () == 0) {
16       // this is the first measure in the segment
17       measureFirstInSegmentKind =
18          msrMeasure::kMeasureFirstInSegmentKindYes;
19    }
20    else {
21       // this is not the first measure in the segment
22       measureFirstInSegmentKind =
23          msrMeasure::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          fSegmentVoiceUpLink->
40             incrementVoiceCurrentMeasureOrdinalNumber ());
41
42    // append result to the segment
43    appendMeasureToSegment (result);
44
45    --gIndenter;
46
47    return result;
48 }
```

19.31   Repeats patterns and replicas

MSR represents repeated beats and measures this way:

- a pattern describes what is repeated;

- there are as many replicas of the music as needed.

This leads to:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep Pattern *.h | grep
     class
2 msrBeatRepeats.h:class EXP msrBeatRepeatPattern : public msrElement
3 msrMeasureRepeats.h:class EXP msrMeasureRepeatPattern : public msrElement
4 jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msr > grep Replicas *.h | grep
     class
5 msrBeatRepeats.h:class EXP msrBeatRepeatReplicas : public msrElement
6 msrMeasureRepeats.h:class EXP msrMeasureRepeatReplicas : public msrElement
```

These two repeat cases are described in the sections below.

19.32 Beat repeats

Beat repeats handling is presented at section 45, [Beat repeats handling], page 276.

Class `msrBeatRepeat`, defined in `src/formats/msr/msrBeatRepeats.h/.cpp`, contains a pattern and replicas:

```
class EXP msrBeatRepeat : public msrVoiceElement
{
  // ... ... ...

  private:

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

    // upLinks
    S_msrVoice            fBeatRepeatVoiceUpLink;

    // numbers
    int                   fBeatRepeatMeasuresNumber;
    int                   fBeatRepeatSlashesNumber;

    // measures repeat pattern
    S_msrBeatRepeatPattern
                          fBeatRepeatPattern;

    // measures repeat replicas
    S_msrBeatRepeatReplicas
                          fBeatRepeatReplicas;

    // measures repeat build phase, used when building the measures repeat
    msrBeatRepeatBuildPhaseKind
                          fCurrentBeatRepeatBuildPhaseKind; // unused??? JMI
};
```

Class `msrBeatRepeatPattern` contains a segment and an uplink:

```
class EXP msrBeatRepeatPattern : public msrElement
{
  // ... ... ...

  private:

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

    // upLinks
    S_msrBeatRepeat     fBeatRepeatUpLink;

    // segment
    S_msrSegment          fBeatRepeatPatternSegment;
 };
```

Class `msrBeatRepeatReplicas` contains a segment and an uplink:

```
class EXP msrBeatRepeatReplicas : public msrElement
{
  // ... ... ...

  private:

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

```

```
10    // upLinks
11    S_msrBeatRepeat        fBeatRepeatUpLink;
12
13    // segment
14    S_msrSegment           fBeatRepeatReplicasSegment;
15 };
```

19.33 Measure repeats

Measure repeats handling is presented at section 46, [Measure repeats handling], page 277.

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             fMeasureRepeatVoiceUpLink;
12
13     // numbers
14     int                    fMeasureRepeatMeasuresNumber;
15     int                    fMeasureRepeatSlashesNumber;
16
17     // measures repeat pattern
18     S_msrMeasureRepeatPattern
19                            fMeasureRepeatPattern;
20
21     // measures repeat replicas
22     S_msrMeasureRepeatReplicas
23                            fMeasureRepeatReplicas;
24
25     // measures repeat build phase, used when building the measures repeat
26     msrMeasureRepeatBuildPhaseKind
27                            fCurrentMeasureRepeatBuildPhaseKind;
28 };
```

Class `msrMeasureRepeatPattern` contains a segment and an uplink:

```
1  class EXP msrMeasureRepeatPattern : public msrElement
2  {
3    // ... ... ...
4
5   private:
6
7      // private fields
8      // -------------------------------------------------------
9
10     // upLinks
11     S_msrMeasureRepeat    fMeasureRepeatUpLink;
12
13     // segment
14     S_msrSegment          fMeasureRepeatPatternSegment;
15 };
```

Class `msrMeasureRepeatReplicas` contain a segment and an uplink:

```cpp
class EXP msrMeasureRepeatReplicas : public msrElement
{
  // ... ... ...

  private:

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

    // upLinks
    S_msrMeasureRepeat    fMeasureRepeatUpLink;

    // segment
    S_msrSegment          fMeasureRepeatReplicasSegment;
};
```

19.34   Full measure rests

Full measure rests handling is presented at section 65, [Full measure rests handling], page 330.

Class `msrFullMeasureRests`, defined in `src/formats/msr/msrFullMeasureRests.h/.cpp`, essentially contains a class `msrFullMeasureRestsContents` instance and a mesure rests number:

```cpp
class EXP msrFullMeasureRests : public msrVoiceElement
{
  // ... ... ...

  private:

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

    S_msrVoice            fFullMeasureRestsVoiceUpLink;

    int                   fFullMeasureRestsNumber;

    S_msrFullMeasureRestsContents
                          fFullMeasureRestsContents;

    string                fFullMeasureRestsNextMeasureNumber;
    int                   fFullMeasureRestsLastMeasurePuristNumber;

    // shortcut for efficiency
    rational              fFullMeasureRestsMeasureSoundingNotes;
};
```

Class `msrFullMeasureRestsContents` contains an class `msrSegment` instance and an uplink:

```cpp
class EXP msrFullMeasureRestsContents : public msrElement
{
  // ... ... ...

  private:

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

    // upLink
    S_msrFullMeasureRests fFullMeasureRestsContentsFullMeasureRestsUpLink;

    S_msrSegment          fFullMeasureRestsContentsSegment;
};
```

19.35 Barlines

19.36 Repeats

Repeats handling is presented at section 48, [Repeats handling], page 279.

Contrary to MusicXML, MusicFormats represents the full structure of repeated music, not just barlines.

The following classes are defined in `src/formats/msr/msrRepeats.h/.cpp`, contains:

```
jacquesmenu@macmini: ~/musicformats -git -dev/src/formats/msr > grep class msrRepeats.h
class msrRepeat;
class msrFullMeasureRests;
class msrMeasureRepeat;
class msrNote;
class EXP msrRepeatCommonPart : public msrElement
class EXP msrRepeatEnding : public msrElement
class EXP msrRepeat : public msrVoiceElement
class EXP msrRepeatDescr : public smartable
class EXP msrRepeatElement : public msrElement
```

Class `msrRepeat`, defined in msrBothmsrRepeats, contains an class `msrRepeatCommonPart`, followed by zero or more instances of class `msrRepeatEnding`:

```
class EXP msrRepeat : public msrVoiceElement
{
  public:

    // data types
    // --------------------------------------------------------

    enum msrRepeatExplicitStartKind {
      kRepeatExplicitStartNo ,
      kRepeatExplicitStartYes
    };

    // ... ... ...

    // common part
    S_msrRepeatCommonPart fRepeatCommonPart;

    // repeat endings
    vector<S_msrRepeatEnding>
                         fRepeatEndings;
    int                  fRepeatEndingsInternalCounter;

    // immediately preceding and following repeats
    // detecting several repeats in a row helps LilyPond code generation
    // depending on the options JMI
    S_msrRepeat          fImmediatelyPrecedingRepeat;
    S_msrRepeat          fImmediatelyFollowingRepeat;
};
```

Class `msrRepeatCommonPart` contains a list of class `msrVoiceElement`:

```
  private:

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

    // upLinks
    S_msrRepeat          fRepeatCommonPartRepeatUpLink;

    // elements list
```

```
10      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           fRepeatEndingRepeatUpLink;
8
9     // numbers
10    string                fRepeatEndingNumber; // may be "1, 2"
11    int                   fRepeatEndingInternalNumber; // internally assigned
12
13    // kind
14    msrRepeatEndingKind   fRepeatEndingKind;
15
16    // elements list
17    list<S_msrVoiceElement>
18                          fRepeatEndingElementsList;
```

19.37 Segments

Segments handling is presented at section 44, [Segments handling], page 271.

Segment are not explicit in music scores, but they are there alright and we have to represent them in MSR:

- it is a sequence of music elements not containing a repeat. This is equivalent to so-called *basic blocs* in compiler technology, that are linear sequences of instructions without jumps, i.e. there is exactly one entry and one exit.

For example, at figure 19.4, [Three segments in a voice], page 180, there are three segments:

- the first one contains the `c1`, and belongs to a first repeat;

- the second one contains the `d1`, and is a member of the voice;

- the last one contains the `e1` and belongs to a second repeat.

Figure 19.4: Three segments in a voice

19.38   Notes and rests

Class `msrNote` is complex class: it handles many variants, but using classes to represent the variants would be too cumbersone. As shown at figure 19.3, [The MSR classes hierarchy], page 167:

- a note can be a standalone (regular) note or rest;

- it can belong to a grace notes group;

- it can belong to chord, which can itself belong to a grace notes group or a tuplet;

- it can belong to a tuplet;

- it can belong to double tremolo;

- and finally, a rest can be unpiched.

class `msrNote` thus uses enumeration type enumeration type `msrNoteKind`, defined in `src/formats/msr/msrBasicTyp` to distinguish them:

```
enum class msrNoteKind {
  k_NoNote,

  // in measures
  kNoteRegularInMeasure,
  kNoteRestInMeasure,
  kNoteSkipInMeasure, // an invisible rest
  kNoteUnpitchedInMeasure,

  // in chords
  kNoteRegularInChord,

  // in tuplets
  kNoteRegularInTuplet,
  kNoteRestInTuplet,
  kNoteUnpitchedInTuplet,

  // in grace notes groups
  kNoteRegularInGraceNotesGroup,
  kNoteSkipInGraceNotesGroup, // used to circumvent LilyPond issue #34

  // in chords in grace notes groups
  kNoteInChordInGraceNotesGroup,

  // in tuplets in grace notes groups
  kNoteInTupletInGraceNotesGroup,

  // in double-tremolos
  kNoteInDoubleTremolo
};
```

19.39   Articulations

19.40   Ornaments

19.41   Ties

19.42   Dynamics

19.43   Beams

19.44   Slurs

19.45   Grace notes groups

Grace notes groups handling is presented at section 59, [Grace notes groups handling], page 296.

19.46   Chords

A chord contains notes only, and can occur in measures, tuplets and grace notes groups, hence:

```
// chords
//------------------------------------------------------------------------------

enum class msrChordInKind {
  k_NoChordIn,

  kChordInMeasure,
  kChordInTuplet,
  kChordInGraceNotesGroup
};
```

19.47   Tuplets

Tuplets handling is presented at section 61, [Tuplets handling], page 298.

A tuplet can contain:

- notes and rests;
- chords;
- other tuplets.

Tuplets can occur in measures and other tuplets, hence enumeration type `msrTupletInKind`:

```
enum class msrTupletInKind {
  k_NoTupletIn,

  kTupletInMeasure,
  kTupletInTuplet
};
```

Tuplets factors are represented by class `msrTupletFactor`, defined in
`src/formats/msr/msrBasicTypes.h/.cpp`.

```cpp
class EXP msrTupletFactor
{
  // ... ... ...

  public:

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

    Bool                  isEqualToOne () const
                              {
                                return
                                  fTupletActualNotes == fTupletNormalNotes;
                              }

    rational              asRational () const
                            {
                                return
                                  rational (
                                    fTupletActualNotes,
                                    fTupletNormalNotes);
                              }

  // ... ... ...

  private:

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

    int                   fTupletActualNotes;
    int                   fTupletNormalNotes;
};
```

19.48   Harmonies and figured bass similarities

Harmonies and figured bass handling is presented at section 62, [Harmonies handling], page 299 and section 63, [Figured bass elements handling], page 315, respectively.

In MusicXML, harmonies and figured bass occur at the measure level:

```xml
    <harmony print-frame="no">
      <root>
        <root-step>C</root-step>
        </root>
      <kind text="m">minor</kind>
      </harmony>
    <note default-x="75.17" default-y="-35.00">
      <pitch>
        <step>F</step>
        <octave>4</octave>
        </pitch>
      <duration>2</duration>
      <voice>1</voice>
      <type>quarter</type>
      <stem>up</stem>
      </note>
```

```
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 `msrFiguredBassElement` 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     list<S_msrHarmony>     fNoteHarmoniesList;
14
15     // figured bass
16     // --------------------------------------------------------
17
18     list<S_msrFiguredBassElement>
19                            fNoteFiguredBassElementsList;
20
21   // ... ... ...
22 };
```

- each class `msrPart` instance contains a harmonies staff and voice, as well as a figured bass staff and voice:

```
1  class EXP msrPart : public msrPartGroupElement
2  {
3    // ... ... ...
4
5    private:
6
7      // private fields
8      // --------------------------------------------------------
9    // harmonies
10
11     S_msrStaff            fPartHarmoniesStaff;
12     S_msrVoice            fPartHarmoniesVoice;
13
14     // figured bass
15
16     S_msrStaff            fPartFiguredBassStaff;
17     S_msrVoice            fPartFiguredBassVoice;
```

```
18
19    // ... ... ...
20  };
```

The way harmonies and figured bass elements are represented in MusicFormats is presented in the next two sections.

19.49   Harmonies

Harmonies handling is presented at section 62, [Harmonies handling], page 299.

19.50   Figured bass elements

Figured bass elements handling is presented at section 63, [Figured bass elements handling], page 315.

19.51   Lyrics

Lyrics handling is presented at section 64, [Lyrics handling], page 329.

Lyrics are handled in rather a special way in music scores:

- they have a linear structure, independent of the repeats structure of the staff they belong too;

- the can be several lyrics stanzas associated to a given staff;

- the syllables in lyrics can apply to more that one note, and the subdivisions of words have to be handled.

The basic building block for lyrics in MSR is class `msrSyllable`, whose variants are distinguished by enumeration type enumeration type `msrSyllableKind`:

```
1     enum msrSyllableKind {
2       kSyllableNone,
3       kSyllableSingle,
4       kSyllableBegin, kSyllableMiddle, kSyllableEnd,
5
6       kSyllableOnRestNote,
7       kSyllableSkipRestNote,
8       kSyllableSkipNonRestNote,
9
10      kSyllableMeasureEnd,
11      kSyllableLineBreak, kSyllablePageBreak
12    };
```

Extensions are described by enumeration type :

```
1     enum msrSyllableExtendKind {
2       kSyllableExtendNone,
3       kSyllableExtendEmpty,
4       kSyllableExtendSingle,
5       kSyllableExtendStart, kSyllableExtendContinue, kSyllableExtendStop
6     };
```

Class `msrSyllable` contains:

```
1    // syllable kind
2    msrSyllableKind       fSyllableKind;
3
4    // texts list
5    list<string>          fSyllableTextsList;
6
7    // extend kind
8    msrSyllableExtendKind fSyllableExtendKind;
9
10   // stanza number, may contain non-digits
11   string                fSyllableStanzaNumber;
12
13   // syllable whole notes
14   rational              fSyllableWholeNotes;
15
16   // syllable tuplet factor
17   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    vector<S_msrSyllable> fSyllables;
3
4    Bool                  fStanzaTextPresent;
```

19.52　MIDI

MIDI handling is presented at section .

Chapter 20

MSR time-oriented represention

In order to represent the music according to simultaneous sounding time, MSR builds:

- a flat list of measures at the voice and staff levels;

- from this, a vector of measures slices at the voice, staff, part, part group and score levels.

The source files are in `src/formats/msr/msrMeasuresSlices.h/.cpp`.

20.1   Note events

Notes start and stop are represented by enumeration type `msrNoteEventKind`:

```
enum class msrNoteEventKind {
  kNoteEventStart,
  kNoteEventStop
};
```

A note event is described in class :

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

  private:

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

    rational              fNoteEventPositionInMeasure;
    S_msrNote             fNoteEventNote;
    msrNoteEventKind      fNoteEventKind;
};
```

20.2   Simultaneous notes chunks

Such a chunk is a set of notes or rests played simultaneously, i.e. that start and stop at the same time. The set is stored as a list actually:

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

  private:

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

    rational            fChunkPositionInMeasure;
    list<S_msrNote>     fChunkNotesList;
    rational            fChunkWholeNotes;
};
```

20.3   Measures slices

A measures slice, described by class **msrMeasuresSlice**, is a 'vertical' cut in the score across voices: is contains all the measures starting at the same time, one per voice:

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

  protected:

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

    int                 fSlicePuristMeasureNumber;
    string              fSliceMeasureNumber;

    // the measures in the slice
    vector<S_msrMeasure>  fSliceMeasuresVector;

    // notes flat list
    list<S_msrNote>     fSliceNotesFlatList;

    // note events list
    list<S_msrNoteEvent>  fSliceNoteEventsList;

    // simultaneous notes chunks list
    list<S_msrSimultaneousNotesChunk>
                        fSliceSimultaneousNotesChunksList;
};
```

20.4   Measures slices sequences

A class **msrMeasuresSlicesSequence** contains a vector of **S_msrMeasuresSlice** instances:

```
1  class EXP msrMeasuresSlicesSequence : public smartable
2  {
3    // ... ... ...
4
5    private:
6
7      // private fields
8      // ------------------------------------------------------
9
10     string                 fMeasuresOrigin;
11
12     vector<S_msrMeasuresSlice>
13                            fMeasuresSlicesVector;
14 };
```

A smart pointer to am `msrMeasuresSlicesSequence` instance is stored in `msrVoice`, `msrStaff`, `msrPart`, `msrPartGroup` and `msrScore`.

20.5 Building the measures slices

20.5.1 Part measures slices

At the part level, this is done in method `msrPart::collectPartMeasuresSlices ()`:

```
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       stringstream s;
24
25       s <<
26         "The staffMeasuresSlicesSequence of staff \"" <<
27         staff->getStaffName () <<
28         "\" is null";
29
30       musicxmlWarning (
31         gGlobalServiceRunData->getInputSourceName (),
32         inputLineNumber,
33         s.str ());
34     }
35     else {
36       fPartMeasuresSlicesSequence->
37         mergeWithMeasuresSlicesSequence (
38           inputLineNumber,
```

```
39          getPartCombinedName (),
40          staffMeasuresSlicesSequence);
41      }
42
43      --gIndenter;
44    } // for
45
46    // ... ... ...
47 }
```

20.5.2   Staff measures slices

Method `msrStaff::collectStaffMeasuresSlices ()` builds them:

```
 1 void msrStaff::collectStaffMeasuresSlices (
 2   int inputLineNumber)
 3 {
 4   // ... ... ...
 5
 6   // create the staff measures slices sequence
 7   fStaffMeasuresSlicesSequence =
 8     msrMeasuresSlicesSequence::create (
 9       fStaffName); // origin
10
11   // populate it
12   for (S_msrVoice voice : fStaffAllVoicesList) {
13     // ... ... ...
14
15     // get the voice measures slices sequence
16     S_msrMeasuresSlicesSequence
17       voiceMeasuresSlicesSequence =
18         voice->
19           getVoiceMeasuresSlicesSequence ();
20
21     // merge it with the voice measures slices sequence
22     if (voiceMeasuresSlicesSequence) { // JMI
23       fStaffMeasuresSlicesSequence =
24         fStaffMeasuresSlicesSequence->
25           mergeWithMeasuresSlicesSequence (
26             inputLineNumber,
27             fStaffName,
28             voiceMeasuresSlicesSequence);
29     }
30
31     // identify the solo notes and rests in the staff
32     fStaffMeasuresSlicesSequence->
33       identifySoloNotesAndRests ();
34
35     --gIndenter;
36   } // for
37
38   // ... ... ...
39 }
```

20.6   Solo notes and rests

A solo note or rest is one that occurs alone at some point in time for its whole duration, without any other
note being played at the same time.

Identifying such solo notes or rests is done in method `msrMeasuresSlicesSequence::identifySoloNotesAndRests` (
using the measures slices of the staff they occur in, called method `msrStaff::collectStaffMeasuresSlices ()`
as shown above:

```cpp
void msrMeasuresSlicesSequence::identifySoloNotesAndRests ()
{
  // ... ... ...

  // collect the notes from the sequence's measures slices
  for (
    vector<S_msrMeasuresSlice>::const_iterator i =
      fMeasuresSlicesVector.begin ();
    i != fMeasuresSlicesVector.end ();
    ++i
  ) {
    S_msrMeasuresSlice measuresSlice = (*i);

    measuresSlice->
      collectNonSkipNotesFromMeasuresSliceMeasures ();
  } // for
}
```

20.7   A measures slices example

Chapter 21

Path to voice

src/formats/msr/msrPathToVoice.h.h/.cpp defines class msrPathToVoice, used to create partial clones of class msrBook retaining only certain staves and/or voices, or to create new class msrScore instances containing each of them only:

```cpp
class EXP msrPathToVoice : public smartable
{
  // ... ... ...

  public:

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

    void                  appendPartGroup (S_msrPartGroup partGroup)
                              {
                                fPartGroupsList.push_back (partGroup);
                              }

  // ... ... ...

  private:

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

    S_msrBook             fBook;

    S_msrScore            fScore;

    // part groups can be nested
    list<S_msrPartGroup>  fPartGroupsList;

    S_msrPart             fPart;

    S_msrStaff            fStaff;

    S_msrVoice            fVoice;
};
```

Chapter 22

LilyPond Scores Representation (LPSR)

An LPSR description contains two components:

- the first one is an MSR, thus the whole music score description;

- the second one is a description of the structure of the score mirroring LilyPond's specific blocks such as \book {...} and \layout {...} .

Class lpsrScore thus contains:

```
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   list<S_msrElement>   fScoreElementsList;
13
14   // score LPSR book blocks list
15   list<S_lpsrBookBlock> fScoreBookBlocksList;
16   S_lpsrScoreBlock      fScoreScoreBlock; // JMI ???
```

22.1   LPSR basic types

Some types used thoughout LSPR are defined in src/formats/lpsr//lpsrBasicTypes.h/.cpp:

```
1    jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/lpsr >  egrep -rIn  '^// '
        lpsrBasicTypes.h
2    lpsrBasicTypes.h:28:// score output kinds
3    lpsrBasicTypes.h:50:// accidental styles
4    lpsrBasicTypes.h:53:// JMI there are new ones in LilyPond 2.22
5    lpsrBasicTypes.h:87:// chords languages
6    lpsrBasicTypes.h:107:// whole notes
7    lpsrBasicTypes.h:118:// dotted durations
8    lpsrBasicTypes.h:129:// rests measures
9    lpsrBasicTypes.h:135:// texts lists
10   lpsrBasicTypes.h:141:// pitches and octaves
11   lpsrBasicTypes.h:151:// lyrics durations
12   lpsrBasicTypes.h:168:// initialization
```

22.2   Adapting LilyPond code generation to the target version number

As of version 2.22, `compressFullBarRests` has been replaced by `compressEmptyMeasures` for clarity.

Such is done specific methods:

Chapter 23

Braille Scores Representation (BSR)

BSR represents braille scores as composed of lines of 6-dot cells.

23.1 BSR basic types

Some types used thoughout BSR are defined in `src/formats/bsr/bsrBasicTypes.h/.cpp`:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/bsr >  egrep -rIn  '^// '
    bsrBasicTypes.h
bsrBasicTypes.h:23:// cell kinds
bsrBasicTypes.h:107:// lower case letters
bsrBasicTypes.h:139:// capitals
bsrBasicTypes.h:143://  kCellCapitalsSequenceSign, // { kCellCapitalsSign,
    kCellCapitalsSign };
bsrBasicTypes.h:145:// decimal digits
bsrBasicTypes.h:160:// lower decimal digits
bsrBasicTypes.h:174:// alterations
bsrBasicTypes.h:181:// augmentation dots
bsrBasicTypes.h:186:// arithmetic operators
bsrBasicTypes.h:195:// words
bsrBasicTypes.h:205:// braille cells
bsrBasicTypes.h:212:// braille output kinds
bsrBasicTypes.h:231:// chords languages
bsrBasicTypes.h:251:// brailling numbers
bsrBasicTypes.h:255:// brailling characters and strings
bsrBasicTypes.h:261:// writing UTF-16 to ostreams
bsrBasicTypes.h:273:// initialization
bsrBasicTypes.h:971:// constants
bsrBasicTypes.h:975:// computations
```

23.2 Representing cells

This is done basically with enumeration type enumeration type `bsrCellKind`, defined in `src/formats/bsr/bsrBasicT`

```
// cell kinds
//_____
enum class bsrCellKind {
  kCellUnknown,

  // non 6dots values
  kCellEOL      , // L'\u000a'
  kCellEOP      , // L'\u000c'

  // 6dots values for Braille
```

```
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   kCell1 = kCellA,
6   kCell2 = kCellB,
7   kCell3 = kCellC,
8   kCell4 = kCellD,
9   kCell5 = kCellE,
10   kCell6 = kCellF,
11   kCell7 = kCellG,
12   kCell8 = kCellH,
13   kCell9 = kCellI,
14   kCell0 = 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:

```
// arithmetic operators
//_____
const bsrCellKind
  kCell_ac_plus     = bsrCellKind::kDots235,
  kCell_ac_minus    = bsrCellKind::kDots36,
  kCell_ac_times    = bsrCellKind::kDots35,
  kCell_ac_dividedBy = bsrCellKind::kDots25,
  kCell_ac_equals   = bsrCellKind::kDots2356;
```

Words:

```
// words
//_____
const bsrCellKind
  kCellWordSign       = bsrCellKind::kDots345,

  kCellWordApostrophe  = bsrCellKind::kDots6,

  kCellParenthesis     = bsrCellKind::kDots2356,
  kCellQuestionMark    = bsrCellKind::kDots26;
```

Chapter 24

MusicXML Scores Representation (MXSR)

This format is provided by `libmusicxml2`, even though Dominique Fober didn't give it that name. It is a tree of class `mxmlelement` nodes, mapped one to one to the MusicXML markups.

The files in `libmusicxml/src`.

A set of interface functions is contained in `src/formats/mxsr/mxsr.h/.cpp`:

```
jacquesmenu@macmini: ~/musicformats-git-dev/src > ll formats/mxsr/
total 48
 0 drwxr-xr-x   6 jacquesmenu  staff   192 May 26 08:20:55 2021 ./
 0 drwxr-xr-x  10 jacquesmenu  staff   320 Jun 25 05:39:49 2021 ../
 8 -rw-r--r--@  1 jacquesmenu  staff  3292 Jun  6 06:35:19 2021 mxsr.cpp
 8 -rw-r--r--@  1 jacquesmenu  staff  1555 Jun  6 06:35:19 2021 mxsrGeneration.h
16 -rw-r--r--@  1 jacquesmenu  staff  7781 Jun  6 06:35:19 2021 mxsrOah.cpp
16 -rw-r--r--@  1 jacquesmenu  staff  4829 Jun  6 06:35:19 2021 mxsrOah.h
```

24.1 MusicXML elements and attributes

MusicXML data contains so-called elements, written as `<... />` markups, that can be nested:

```
        <system-margins>
          <left-margin>15</left-margin>
          <right-margin>0</right-margin>
        </system-margins>
```

In the example above, the values of the two margins are `15` and `0`, respectively.

MusicXML elements can have attributes, such as `version` below:

```
<score-partwise version="3.1">
```

The values of the elements and attributes are strings.

There are two special elements at the beginning of MusicXML data:

- a `<?xml/>` element indicating the characters encoding used;

- a `<"!"DOCTYPE/>` element telling that the contents is in 'score-partwise' mode and containing the URL of the DTD.

An exemple is:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 2.0 Partwise//EN"
3                                  "http://www.musicxml.org/dtds/partwise.dtd">
```

24.2 The `xmlelement` and `xmlattribute` types

xmlelementxmlattribute

These two classes are defined in `libmusicxml/src/elements!xml.h/.cpp`:

```
1  class xmlelement;
2  class xmlattribute;
3
4  typedef SMARTP<xmlattribute>  Sxmlattribute;
5  typedef SMARTP<xmlelement>    Sxmlelement;
```

Class `xmlattribute` contains:

```
1  //_____
2  class EXP xmlattribute : public smartable {
3    //! the attribute name
4    std::string fName;
5    //! the attribute value
6    std::string  fValue;
7
8    // ... ... ...
9
10      //! returns the attribute value as a int
11      operator int () const;
12      //! returns the attribute value as a long
13      operator long () const;
14      //! returns the attribute value as a float
15      operator float () const;
```

Class `xmlelement` contains:

```
1  class EXP xmlelement : public ctree<xmlelement>, public visitable
2  {
3    private:
4      //! the element name
5      std::string fName;
6      //! the element value
7      std::string fValue;
8      //! list of the element attributes
9      std::vector<Sxmlattribute> fAttributes;
10
11    protected:
12      // the element type
13      int fType;
14      // the input line number for messages to the user
15      int fInputLineNumber;
16
17    // ... ... ...
18
19      //! returns the element value as a long
20      operator long () const;
21      //! returns the element value as a int
22      operator int () const;
23      //! returns the element value as a float
24      operator float () const;
25      //! elements comparison
```

```
26      Bool operator ==(const xmlelement& elt) const;
27      Bool operator !=(const xmlelement& elt) const { return !(*this == elt); }
28
29      //! adds an attribute to the element
30      long add (const Sxmlattribute& attr);
31
32   // ... ... ...
33 };
```

Type `Sxmlelement` is a smart pointer to an `xmlelement`, so it is an `xmlelement` tree, since `xmlelement` is a recursive type.

`fInputLineNumber` is used for example in warning and error messages, to help the user locate the problem.

`fType` typically contains a value of some enumeration type , more on this below.

24.3 Enumeration types for `xmlelement`'s `fType`

`xmlelement`

`libmusicxml2` uses `elements/templates/elements.bash`, a Bash script, to generate the enumeration type constants and classes source code from the MusicXML DTD. This is not done in the `Makefile`, since it is to be run by hand only once.

The DTD files we use as reference are in `libmusicxml/dtds/3.1/schema`;

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/libmusicxml/dtds/3.1/schema > ls -sal *.mod
2  40 -rwxr-xr-x@ 1 jacquesmenu  staff  20238 Apr 22 15:49 attributes.mod
3  16 -rwxr-xr-x  1 jacquesmenu  staff   4943 Apr 22 15:49 barLine.mod
4  80 -rwxr-xr-x@ 1 jacquesmenu  staff  37932 Apr 22 15:49 common.mod
5  88 -rwxr-xr-x@ 1 jacquesmenu  staff  41960 Apr 22 15:49 direction.mod
6  16 -rwxr-xr-x@ 1 jacquesmenu  staff   4097 Apr 22 15:49 identity.mod
7  24 -rwxr-xr-x@ 1 jacquesmenu  staff  10266 Apr 22 15:49 layout.mod
8   8 -rwxr-xr-x@ 1 jacquesmenu  staff   2833 Apr 22 15:49 link.mod
9 104 -rwxr-xr-x@ 1 jacquesmenu  staff  51384 Apr 22 15:49 note.mod
10  32 -rwxr-xr-x@ 1 jacquesmenu  staff  15476 Apr 22 15:49 score.mod
```

The first result of running `libmusicxml/src/elements/templates!elements.bash` is an anonymous enumeration type defined in `libmusicxml/src/elements!elements.h`:

```
1 enum {
2   kNoElement,
3   kComment,
4   kProcessingInstruction,
5   k_accent,
6   k_accidental,
7   k_accidental_mark,
8   k_accidental_text,
9
10   // ... ... ...
11
12   k_work,
13   k_work_number,
14   k_work_title,
15   kEndElement
16 };
```

The constants `kNoElement`, `kComment` and `kProcessingInstruction` are added by `elements.bash`.

24.4 Classes for the `xmlelements`

All the MusicXML classes are instantiated from the `musicxml` template class, defined in `libmusicxml/src/elements!`
This is where `fType` gets its value:

```
template <int elt> class musicxml : public xmlelement
{
  protected:
    musicxml (int inputLineNumber) : xmlelement (inputLineNumber) { fType = elt; }
};
```

The smart pointer s to the various elements are defined in `libmusicxml/src/elements!typedef.h`, using
an anonymous enumeration type :

```
typedef SMARTP<musicxml<kComment> >             S_comment;
typedef SMARTP<musicxml<kProcessingInstruction> > S_processing_instruction;

typedef SMARTP<musicxml<k_accent> >   S_accent;
typedef SMARTP<musicxml<k_accidental> >   S_accidental;
typedef SMARTP<musicxml<k_accidental_mark> >    S_accidental_mark;
typedef SMARTP<musicxml<k_accidental_text> >    S_accidental_text;

// ... ... ...

typedef SMARTP<musicxml<k_work> >    S_work;
typedef SMARTP<musicxml<k_work_number> >    S_work_number;
typedef SMARTP<musicxml<k_work_title> >   S_work_title;
```

The two-way correspondance of MusicXML elements names to type `Sxmlelement` is stored `fMap` and
`fType2Name`, defined in `libmusicxml/src/elements!factory.h`:

```
class EXP factory : public singleton<factory>{

  std::map<std::string, functor<Sxmlelement>*> fMap;
  std::map<int, const char*>  fType2Name;

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

Those two maps are initialized in `libmusicxml/samples/elements/factory.cpp`:

```
factory::factory()
{
  fMap["comment"]      = new newElementFunctor<kComment>;
  fMap["pi"]         = new newElementFunctor<kProcessingInstruction>;
  fType2Name[kComment]  = "comment";
  fType2Name[kProcessingInstruction]  = "pi";

  fMap["accent"]   = new newElementFunctor<k_accent>;
  fMap["accidental"]   = new newElementFunctor<k_accidental>;
  fMap["accidental-mark"]    = new newElementFunctor<k_accidental_mark>;
  fMap["accidental-text"]    = new newElementFunctor<k_accidental_text>;

  // ... ... ...

  fMap["work"]   = new newElementFunctor<k_work>;
  fMap["work-number"]    = new newElementFunctor<k_work_number>;
  fMap["work-title"]   = new newElementFunctor<k_work_title>;

  fType2Name[k_accent]  = "accent";
  fType2Name[k_accidental]  = "accidental";
  fType2Name[k_accidental_mark]    = "accidental-mark";
  fType2Name[k_accidental_text]    = "accidental-text";
```

```
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 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 (libmxmllineno); }
8 };
```

24.5 xmlelement trees

This section describes features supplied by `libmusicxml2`.

An `xmlelement` is the basic brick to represent a MusicXML element.

Smart pointer type `SXMLFile` is defined in `libmusicxml/src/files/xmlfile.h`:

```
1 //_____
2 class EXP TXMLFile : public smartable
3 {
4   private:
5     TXMLDecl*            fXMLDecl;
6     TDocType*            fDocType;
7     Sxmlelement          fXMLTree;
8
9   protected:
10        TXMLFile () : fXMLDecl(0), fDocType(0) {}
11    virtual ~TXMLFile () { delete fXMLDecl; delete fDocType; }
12
13   public:
14    static SMARTP<TXMLFile> create();
15
16   public:
17    TXMLDecl*    getXMLDecl ()     { return fXMLDecl; }
18    TDocType*    getDocType ()     { return fDocType; }
19    Sxmlelement  elements ()       { return fXMLTree; }
20
21    void      set (Sxmlelement root)  { fXMLTree = root; }
22    void      set (TXMLDecl * dec)  { fXMLDecl = dec; }
23    void      set (TDocType * dt)   { fDocType = dt; }
24
25    void      print (std::ostream& s);
26 };
27 typedef SMARTP<TXMLFile> SXMLFile;
```

24.5.1 Creating `xmlelement` trees from textual data

Reading MusicXML data creates instances of `xmlelement`. This is done by and instance of `xmlreader`, defined in `libmusicxml/src/files/xmlreader.h/.cpp`, which provides methods:

```
SXMLFile readbuff(const char* file);
SXMLFile read(const char* file);
SXMLFile read(FILE* file);
```

These three functions are defined this way:

```
//------------------------------------------------------------------------
SXMLFile xmlreader::readbuff(const char* buffer)
{
  fFile = TXMLFile::create();
  debug("read buffer", '-');
  return readbuffer (buffer, this) ? fFile : 0;
}

//------------------------------------------------------------------------
SXMLFile xmlreader::read(const char* file)
{
  fFile = TXMLFile::create();
  debug("read", file);
  return readfile (file, this) ? fFile : 0;
}

//------------------------------------------------------------------------
SXMLFile xmlreader::read(FILE* file)
{
  fFile = TXMLFile::create();
  return readstream (file, this) ? fFile : 0;
}
```

24.5.2 Printing `xmlelement` trees

An `xmlelement` can be printed by function `printMxsr ()`, defined in `src/formats/mxsr/mxsr.h/.cpp`:

```
void printMxsr (const Sxmlelement theMxsr, ostream& os)
{
  xmlvisitor v (os);
  tree_browser<xmlelement> browser (&v);
  browser.browse (*theMxsr);
}
```

This how MusicXML and Guido output are generated.

24.6 The SXMLFile type

`SXMLFile` is defined in `libmusicxml/src/factory!musicxmlfactory.h` as a smart pointer to class `TXMLFile`:

```
//------------------------------------------------------------------------
class EXP TXMLFile : public smartable
{
  private:
    TXMLDecl*              fXMLDecl;
    TDocType*             fDocType;
    Sxmlelement           fXMLTree;

  protected:
```

```
10          TXMLFile () : fXMLDecl(0), fDocType(0) {}
11      virtual ~TXMLFile () { delete fXMLDecl; delete fDocType; }
12
13    public:
14      static SMARTP<TXMLFile> create();
15
16    public:
17      TXMLDecl*     getXMLDecl ()      { return fXMLDecl; }
18      TDocType*     getDocType ()      { return fDocType; }
19      Sxmlelement   elements ()        { return fXMLTree; }
20
21      void      set (Sxmlelement root)  { fXMLTree = root; }
22      void      set (TXMLDecl * dec)  { fXMLDecl = dec; }
23      void      set (TDocType * dt)   { fDocType = dt; }
24
25      void      print (std::ostream& s);
26 };
27 typedef SMARTP<TXMLFile> SXMLFile;
```

`fXMLDecl` describes the `<?xml/>` element and `fDocType` contains the `<"!"DOCTYPE/>` element.

Part VII

Passes

Chapter 25

The passes

A pass performs a single translation from one music score description into another, such as from MusicXML to an MXSR, or from an MXSR to an MSR. The name 'pass' comes from the compiler writing field.

25.1   Translating MusicXML data to an MXSR format

This is supplied by the `libmusicxml2` library, a version of which is distributed as part of MusicFormats to avoid the need of two installs and the potential associated problems.

25.1.1   MusicXML coverage

`src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.h/.cpp` and `src/passes/mxsr2msr/mxsr2msrTranslator.h/.c` handle many of the MusicXML version 3.1 elements. Some of them are handled by both sub-passes, such as `S_voice`, `S_measure` and `S_harmony`.

Among the elements that MusicFormats does not handled are the ones for which there is no occurrence in the corpus in folder `files/musicxml`, such as `beat-unit-tied` and `metronome-tied`.

The elements that are new in MusicXML version 4.0 are not known nor handled yet.

25.2   Translating an MXSR to an MSR

This is done by class `mxsr2msrTranslator`.

25.3   Translating an MSR to an MXSR

25.4   Translating an MSR to another MSR

Such translation is meant to offer an opportunity to modify the score's description depending on options.

25.5   Translating an MSR to an LPSR

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

25.6 Translating an LPSR to LilyPond code

There are two visiting trace options for the generation of LilyPond code, one for its MSR component, and the other one for its LPSR own part:

```
// %%%JMI    Bool                    fGenerateMsrVisitingInformation;
   Bool                  fGenerateLpsrVisitingInformation;
```

25.7 Translating an MSR to an BSR

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

25.8 Translating a BSR to another BSR

25.9 Translating an MXSR to Guido

Chapter 26

LilyPond code generation

LilyPond code is produced on the standard output stream, unless options option `-output-file-name, -o` or
option `-auto-output-file-name, -aofn` are used.

26.1   Basic principle

Lilypond generation is done in `src/passes/lpsr2lilypond/lpsr2lilypondTranslator.h/.cpp`.

Class `lpsr2lilypondTranslator` provides:

```
public:

                        lpsr2lilypondTranslator (
                          S_msrOahGroup&  msrOpts,
                          S_lpsrOahGroup& lpsrOpts,
                          ostream&        lilypondCodeStream);

    virtual             ~lpsr2lilypondTranslator ();

    void                translateLpsrToLilypondCode (
                          S_lpsrScore lpsrScore);
```

It contains these fields among others:

```
private:

    // options
    // --------------------------------------------------------

    S_msrOahGroup       fMsrOahGroup;
    S_lpsrOahGroup      fLpsrOahGroup;

    // the LPSR score we're visiting
    // --------------------------------------------------------
    S_lpsrScore         fVisitedLpsrScore;

    // the output stream
    // --------------------------------------------------------

    ostream&            fLilypondCodeStream;
```

26.2   Generating Scheme functions in the LilyPond output

xml2ly can generate Scheme code that is used by the LilyPond code it generates. This is described in class lpsrScore by a number of *IsNeeded fields, such as:

```
1    // files includes
2    Bool                    fJianpuFileIncludeIsNeeded;
3
4    // Scheme modules
5    Bool                    fScmAndAccregSchemeModulesAreNeeded;
6
7    // Scheme functions
8    Bool                    fTongueSchemeFunctionIsNeeded;
```

Chapter 27

Braille generation

Braille is written to standard output or to files as binary data. Our reference is http://www.brailleauthority.org/music/Music_Braille_Code_2015.pdf.

27.1 Basic principle

Lilypond generation is done in src/passes/bsr2braille/bsr2brailleTranslator.h/.cpp.

Class bsr2brailleTranslator provides:

```
public:

                      bsr2brailleTranslator (
                        S_bsrScore      bsrScore,
                        const S_bsrOahGroup& bsrOpts,
                        ostream&        brailleOutputStream);

    virtual           ~bsr2brailleTranslator ();

    void              translateBsrToBraille ();
```

It contains these fields among others:

```
private:

    // options
    // ------------------------------------------------------

    S_bsrOahGroup        fBsrOahGroup;

    // the BSR score we're visiting
    // ------------------------------------------------------

    S_bsrScore           fVisitedBsrScore;

    // the braille generator used
    // ------------------------------------------------------

    S_bsrBrailleGenerator fBrailleGenerator;

    // the output stream
    // ------------------------------------------------------

    ostream&             fBrailleOutputStream;
```

27.2   Output files name and contents options

he contents options use the following enumeration types:

```
enum class bsrUTFKind {
  kUTF8, kUTF16
};

enum class bsrByteOrderingKind {
  kByteOrderingNone,
  kByteOrderingBigEndian, kByteOrderingSmallEndian
};
```

`xml2brl` supplies a option `-files` options subgroup:

```
jacquesmenu@macmini > xml2brl -query files
--- Help for subgroup "files" in group "Files group" ---
  Files group (-files-group):
  ------------------------
    Files (-files):
      -o, -output-file-name FILENAME
          Write Braille to file FILENAME instead of standard output.
      -aofn, -auto-output-file-name
          This option can only be used when reading from a file.
          Write MusicXML code to a file in the current working directory.
          The file name is derived from that of the input file,
          replacing any suffix after the '.' by 'xml'
          or adding '.xml' if none is present.
      -bok, -braille-output-kind OUTPUT_KIND
          Use OUTPUT_KIND to write the generated Braille to the output.
          The 4 output kinds available are:
          ascii, utf16, utf8 and utf8d.
          'utf8d' leads to every line in the braille score to be generated
          as a line of cells followed by a line of text showing the contents
          for debug purposes.
          The default is 'ascii'.
      -ueifn, -use-encoding-in-file-name
          Append a description of the encoding used
          and the presence of a BOM if any to the file name before the '.'.
      -bom, -byte-ordering-mark BOM_ENDIAN
          Generate an initial BOM_ENDIAN byte ordering mark (BOM)
          ahead of the Braille nusic code,
          which can be one of 'big' or 'small'.
          By default, a big endian BOM is generated.
```

27.3   Braille generators

The following classes are defined in `src/formatsgeneration/brailleGeneration/brailleGeneration.h/.cpp`

```
jacquesmenu@macmini: ~/musicformats-git-dev/src/formatsgeneration/brailleGeneration > grep
    class brailleGeneration.h
enum class bsrUTFKind {
enum class bsrByteOrderingKind {
class EXP bsrBrailleGenerator : public smartable
/* this class is purely virtual
class EXP bsrAsciiBrailleGenerator : public bsrBrailleGenerator
class EXP bsrUTF8BrailleGenerator : public bsrBrailleGenerator
class EXP bsrUTF8DebugBrailleGenerator : public bsrUTF8BrailleGenerator
class EXP bsrUTF16BigEndianBrailleGenerator : public bsrBrailleGenerator
class EXP bsrUTF16SmallEndianBrailleGenerator : public bsrBrailleGenerator
```

The base class `bsrBrailleGenerator` contains:

```
1    public:
2
3      // public services
4      // ------------------------------------------------------
5
6      virtual void          generateCodeForBrailleCell (
7                              bsrCellKind cellKind) = 0;
8
9      void                  generateCodeForCellsList (
10                             S_bsrCellsList cellsList);
11
12     virtual void          generateCodeForMusicHeading (
13                             S_bsrMusicHeading musicHeading);
14
15     virtual void          generateCodeForLineContents (
16                             S_bsrLineContents lineContents);
17
18     // ... ... ...
19
20   protected:
21
22     // protected fields
23     // ------------------------------------------------------
24
25     ostream&              fBrailleOutputStream;
```

27.4   Writing braille cells

Braille cells are output to an `ostream` as hexadecimal strings by virtual method `generateCodeForBrailleCell ()` methods in `src/passes/bsr2braille/brailleGeneration.h`, depending on the kind of output chosen.

For example, ASCII braille generation is done by:

```
1  void bsrAsciiBrailleGenerator::generateCodeForBrailleCell (
2    bsrCellKind cellKind)
3  {
4    string stringForCell;
5
6    switch (cellKind) {
7      case bsrCellKind::kCellUnknown:
8        {
9          stringstream s;
10
11         s <<
12           "cannot generate code for braille cell '" <<
13           bsrCellKindAsString (cellKind) <<
14           "'";
15         msrInternalError (
16           gGlobalServiceRunData->getInputSourceName (),
17           -999, // inputLineNumber, TICINO JMI
18           __FILE__, __LINE__,
19           s.str ());
20       }
21       break;
22
23     case bsrCellKind::kCellEOL:    stringForCell = "\x0a"; break;
24     case bsrCellKind::kCellEOP:    stringForCell = "\x0c"; break;
25
26     case bsrCellKind::kDotsNone:   stringForCell = "\x20"; break;
27
28     case bsrCellKind::kDots1:      stringForCell = "\x41"; break;
29     case bsrCellKind::kDots2:      stringForCell = "\x31"; break;
30
```

```
31      // ... ... ...
32
33      case bsrCellKind::kDots23456:  stringForCell = "\x29"; break;
34      case bsrCellKind::kDots123456: stringForCell = "\x3d"; break;
35    } // switch
36
37    fBrailleOutputStream <<
38      stringForCell;
39 }
```

Chapter 28

MusicXML generation

MusicXML text is produced on the standard output stream, unless options '`-output-file-name`' or '`-auto-output-f:`
are used.

28.1   Basic principle

MusicXML generation is done in two passes:

- first create and MXSR containing the data;

- then simply write this tree.

28.2   Creating an `xmlelement`

An simple example is:

```
// create a direction element
Sxmlelement directionElement = createMxmlelement (k_direction, "");

// set it's "placement" attribute if relevant
string
  placementString =
    msrPlacementKindAsMusicXMLString (placementKind);

if (placementString.size ()) {
  directionElement->add (createMxmlAttribute ("placement",  placementString));
}
```

This one supplies a value to the `xmlelement` it creates:

```
void msr2mxsrTranslator::visitStart (S_msrIdentification& elt)
{
  // composers
  const list<string>&
    composersList =
      elt->getComposersList ();

  for (
    list<string>::const_iterator i=composersList.begin ();
    i!=composersList.end ();
    ++i
  ) {
    string variableValue = (*i);
```

```
14
15      // create a creator element
16      Sxmlelement creatorElement = createMxmlelement (k_creator, variableValue);
17
18      // set its "type" attribute
19      creatorElement->add (createMxmlAttribute ("type", "composer"));
20
21      // append it to the composers elements list
22      fComposersElementsList.push_back (creatorElement);
23    } // for
24
25    // ... ... ...
26 }
```

28.3   Creating an `xmlelement` tree

In , this code:

```
1  void msr2mxsrTranslator::visitStart (S_msrClef& elt)
2  {
3      // ... ... ...
4
5      Sxmlelement clefElement = createMxmlelement (k_clef, "");
6
7      // set clefElement's "number" attribute if relevant
8      /*
9        0 by default in MSR,
10       1 by default in MusicXML:
11         The optional  number attribute refers to staff numbers within the part,
12         from top to bottom on the system.
13         A value of 1 is assumed if not present.
14      */
15
16      int clefStaffNumber =
17        elt->getClefStaffNumber ();
18
19      if (clefStaffNumber > 1) {
20        clefElement->add (
21          createMxmlIntegerAttribute ("number", clefStaffNumber));
22      }
23
24      // populate clefElement
25      switch (elt->getClefKind ()) {
26        // ... ... ...
27
28        case msrClefKind::kClefTrebleMinus8:
29          {
30            clefElement->push (
31              createMxmlelement (
32                k_sign,
33                "G"));
34            clefElement->push (
35              createMxmlIntegerElement (
36                k_line,
37                2));
38            clefElement->push (
39              createMxmlIntegerElement (
40                k_clef_octave_change,
41                -1));
42          }
43          break;
44
45        // ... ... ...
46 }
```

creates this MusicXML element depending on the value returned by method `msrClef::getClefStaffNumber ()`:

```
<clef number="2">
    <sign>G</sign>
    <line>2</line>
    <clef-octave-change>-1</clef-octave-change>
</clef>
```

28.4 Browsing the visited MSR score

The creation of the tree is done in `src/passes/msr2mxsr/msr2mxsrTranslator.h/.cpp`.

Class `msr2mxsrTranslator` is defined in those files, it contains:

```
public:

                      msr2mxsrTranslator (
                        S_msrScore visitedMsrScore);

    virtual           ~msr2mxsrTranslator ();

    Sxmlelement        translateMsrToMxsr ();

  // ... ... ...

  private:

    // the MSR score we're visiting
    // ------------------------------------------------------
    S_msrScore                fVisitedMsrScore;


    // the MXSR we're building
    // ------------------------------------------------------
    Sxmlelement               fResultingMusicxmlelement;
```

The method `msr2mxsrTranslator::translateMsrToMxsr ()` method does the following:

```
//_____
Sxmlelement msr2mxsrTranslator::translateMsrToMxsr ()
{
  // sanity check
  mfAssert (
    __FILE__, __LINE__,
    fVisitedMsrScore != nullptr,
    "fVisitedMsrScore is null");

  // create the current score part-wise element
  fResultingMusicxmlelement =
    createMxmlScorePartWiseElement ();

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

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

  return fResultingMusicxmlelement;
}
```

28.5 Ancillary functions to create MXSR data

The function `createMxmlScorePartWiseElement ()` is defined in `src/formats/mxsr/mxsr.h/.cpp`:

```
//-------------------------------------------------------------------
Sxmlelement createMxmlScorePartWiseElement ()
{
  Sxmlelement result = factory::instance ().create (k_score_partwise);

  Sxmlattribute versionAttribute = createMxmlAttribute("version", "3.1");
  result->add (versionAttribute);

  return result;
}
```

Chapter 29

Guido code generation

Guido code is produced on the standard output stream, unless options option `-output-file-name, -o` or option `-auto-output-file-name, -aofn` are used.

29.1   Basic principle

As is done for MusicXML generation, Guido generation is done in two passes:

- first create and `mxsr` containing the data;

- then simply write this tree.

The creation of the tree is done in `src/passes/msr2mxsr/msr2mxsrTranslator.h/.cpp`.   See subsection 28.1, [musicxmlGeneration], page 213, for more details.

Part VIII

Generators

Chapter 30

The generators

A generator creates a music score ex-nihilo, without any description of the music being input. It's behaviour can be adapted to the users needs with options if needed.

Generators are supplied in the `src/generators/` directory. They don't have any interface in at the time of this writing, even though they could.

30.1 MusicAndHarmonies

`MusicAndHarmonies.cpp`

30.2 Mikrokosmos3Wandering

This service produces the score for Zoltán Kodály's Mikrokosmos III Wandering score, taking inspiration from the same example in Abjad (http://abjad.mbrsi.org/literature_examples/bartok.html). Is was written in the first place to check the MSR API before writing the MSDL converter.

The score produced is shown at figure 30.1, [Zoltán Kodály's Mikrokosmos III Wandering], page 219.

Figure 30.1: Zoltán Kodály's Mikrokosmos III Wandering

30.3 `LilyPondIssue34`

This service produces the same score as that obtained by:

```
xml2ly -auto-output-file-name gracenotes/LilyPondIssue34.xml
```

The resulting score is shown at figure 30.2, [The LilyPondIssue34 score], page 220.

Figure 30.2: The LilyPondIssue34 score

The name `LilyPondIssue34` stems from the fact that translating this MusicXML file to LilyPond with `musicxml2ly` exhibits the famous LilyPond issue #34.

This example was written to design a LilyPond-oriented interface to LPSR, preparing the grounds for LilyPond export to other formats. This work in in progress at the time of this writing.

Part IX

Converters

Chapter 31

The converters

A multi-pass converter performs a sequence of passes, i.e. a sequence of steps. For example, `xml2ly` performs the following passes:

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

31.1   xml2ly

MusicXML (*Music eXtended Markup Language*) is a specification language meant to represent music scores by texts, readable both by humans and computers. It has been designed by the W3C Music Notation Community Group (https://www.w3.org/community/music-notation/) to help sharing music score files between applications, through export and import mechanisms.

The homepage to MusicXML is https://www.musicxml.com.

MusicXML data contains very detailed information about the music score, and it is quite verbose by nature. This makes creating such data by hand quite difficult, and this is done by applications actually.

The MusicXML data is not systematically checked for correctness. Checks are done, however, to ensure it won't crash due to missing values.

31.2   xml2brl

`xml2brl` is mentioned here, but not described in detail.

31.3   `xml2xml`

31.4   `xml2gmn`

31.5   `msdlconverter`

Part X

Interfaces

Chapter 32

Library interfaces

Chapter 33

Representations interfaces

These interfaces are a set of functions to create formats for various needs.

33.1 MSR interfaces

The MSR interfaces are in `interfaces/msrinterfaces/`:

```
1  jacquesmenu@macmini: ~/musicformats -git -dev/src > ll interfaces/msrinterfaces/
2  total 32
3   0 drwxr-xr-x  5 jacquesmenu  staff   160 May 26 08:20:55 2021 ./
4   0 drwxr-xr-x  8 jacquesmenu  staff   256 Jun 25 05:59:13 2021 ../
5   8 -rw-r--r--@ 1 jacquesmenu  staff    77 Apr 22 15:49:27 2021 README.md
6  16 -rw-r--r--@ 1 jacquesmenu  staff  5796 Jun 24 17:47:02 2021 msrInterface.cpp
7   8 -rw-r--r--@ 1 jacquesmenu  staff  1371 Jun 13 07:38:04 2021 msrInterface.h
```

33.2 LPSR interfaces

The LSPR interfaces are in `interfaces/lpsrinterfaces/`:

```
1  jacquesmenu@macmini: ~/musicformats -git -dev/src > ll interfaces/lpsrinterfaces/
2  total 24
3  0 drwxr-xr-x  5 jacquesmenu  staff   160 Jun 13 07:36:53 2021 ./
4  0 drwxr-xr-x  8 jacquesmenu  staff   256 Jun 25 05:59:13 2021 ../
5  8 -rw-r--r--@ 1 jacquesmenu  staff    78 Jun 13 07:37:13 2021 README.md
6  8 -rw-r--r--@ 1 jacquesmenu  staff   670 Jun 13 07:41:01 2021 lpsrInterface.cpp
7  8 -rw-r--r--@ 1 jacquesmenu  staff  1450 Jun 13 07:39:29 2021 lpsrInterface.h
```

33.3 MSDL interfaces

The MSDL interfaces are in `interfaces/msdlinterfaces/`:

```
1  jacquesmenu@macmini: ~/musicformats -git -dev/src > ll interfaces/msdlinterfaces/
2  total 8
3  0 drwxr-xr-x  3 jacquesmenu  staff    96 Jun 25 05:57:39 2021 ./
4  0 drwxr-xr-x  8 jacquesmenu  staff   256 Jun 25 05:59:13 2021 ../
5  8 -rw-r--r--@ 1 jacquesmenu  staff  1967 Jun  6 06:38:55 2021 libmsdl.h
```

Chapter 34

Passes interfaces

MusicFormats provides its functionality in two ways:

- a set of API functions providing its services to any application, including the ones hosted on the Web;

- a set of CLI tools, to be used in terminals and scripts.

The CLI tool actually use the API functions to do their job. For example, in , the `main ()` function does:

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

  catchSignals ();

  // ... ... ...

  switch (multiGenerationOutputKind) {
    case mfMultiGenerationOutputKind::k_NoGeneration:
      // should not occur, unless the run is a pure help one
      break;

    // ... ... ...

    case mfMultiGenerationOutputKind::kGenerationGuido:
      err =
        msrScore2guidoWithHandler (
          theMsrScore,
          "Pass 2",
          "Convert the MSR score into a second MSR",
          "Pass 3",
          "Convert the second MSR into an MXSR",
          "Pass 4",
          "Convert the MXSR into Guido text",
          cout,
          cerr,
          handler);
      break;

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

34.1 Translating MusicXML data to an MXSR

34.2 Translating an MXSR to an MSR

34.3 Translating an MSR to an MXSR

34.4 Translating an MSR to another MSR

Such translation is meant to offer an opportunity to modify the score's description depending on options.

34.5 Translating an MSR to an LPSR

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

34.6 Translating an LPSR to LilyPond code

34.7 Translating an MSR to an BSR

This converter embeds a specific converter of MSR to MSR, to circumvent the famous LilyPond issue #34.

34.8 Translating a BSR to another BSR

34.9 Translating an MXSR to Guido

Chapter 35

Converters interfaces

These interfaces are a set of functions to run the various converters. They are placed in the corresponding subdirectories of src/converters/, such as src/converters/musicxml2musicxml/musicxml2musicxmlInterface.

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/src/converters/musicxml2musicxml > cat
      musicxml2musicxmlInterface.h
2 /*
3   MusicFormats Library
4   Copyright (C) Jacques Menu 2016-2022
5
6   This Source Code Form is subject to the terms of the Mozilla Public
7   License, v. 2.0. If a copy of the MPL was not distributed with this
8   file, You can obtain one at http://mozilla.org/MPL/2.0/.
9
10   https://github.com/jacques-menu/musicformats
11 */
12
13 #ifndef ___musicxml2musicxmlInterface___
14 #define ___musicxml2musicxmlInterface___
15
16 #include "mfMusicformatsError.h" // for mfMusicformatsError
17
18
19 using namespace std;
20
21 namespace MusicFormats
22 {
23 /*
24   The API functions with an options and arguments and no handler
25   are declared in libmusicxml.h
26 */
27
28 //_____
29 EXP mfMusicformatsError convertMusicxmlFile2musicxmlWithHandler (
30   const char*   fileName,
31   ostream&      out,
32   ostream&      err,
33   S_oahHandler handler);
34
35 //_____
36 EXP mfMusicformatsError convertMusicxmlFd2musicxmlWithHandler (
37   FILE*         fd,
38   ostream&      out,
39   ostream&      err,
40   S_oahHandler handler);
41
42 //_____
43 EXP mfMusicformatsError convertMusicxmlString2musicxmlWithHandler (
44   const char*   buffer,
```

```
45    ostream&        out ,
46    ostream&        err ,
47    S_oahHandler  handler );
48
49
50 }
51
52
53 #endif
```

Part XI

Selected topics

Chapter 36

Initializations

Some initialization activities in MusicFormats use the OAH facility. OAH should thus be initialized first.

36.1   Options and help initializations

There is no initialization of the OAH architecture as such, but there are **create*OahGroup ()** functions to create the various OAH groups.

For example, global variable **gGlobalServiceRunData** is supplied by src/mflibrary/mfServiceRunData.h/.cpp:

```
EXP extern S_generalOahGroup gGlobalServiceRunData;

//_____
EXP S_generalOahGroup createGlobalGeneralOahGroup ();
```

```
S_generalOahGroup createGlobalGeneralOahGroup ()
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
    gLogStream <<
      "Creating global general OAH group" <<
      endl;
  }
#endif

  // protect library against multiple initializations
  if (! gGlobalServiceRunData) {
    // create the global general options group
    gGlobalServiceRunData =
      generalOahGroup::create ();
    assert (gGlobalServiceRunData != 0);
  }

  // return the global OAH group
  return gGlobalServiceRunData;
}
```

36.2 Representations initializations

There are `initialize*` `()` functions such as `initializeLPSR` `()` and `initializeBSR` `()`. They essentially build global data structures, such as the tables of supported languages and their correspondance with an internal enumeration type   both ways.

For example, `initializeMSR` `()` is defined in `src/formats/msr/msr.h/.cpp`:

```
void EXP initializeMSR ();
```

```
static S_mfcVersions pMsrRepresentationComponent;

static void initializeMsrRepresentationComponent ()
{
  // create the component
  pMsrRepresentationComponent =
    mfcVersions::create ("MSR");

  // populate it
  pMsrRepresentationComponent->
    appendVersionDescrToComponent (
      mfcVersionDescr::create (
        mfcVersionDescr::create (
          mfcVersionNumber::createFromString ("0.9.50"),
          "October 6, 2021",
          list<string> {
            "Start of sequential versions numbering"
          }
      )));
}

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

  if (! pPrivateThisMethodHasBeenRun) {
    // initialize the history
    initializeMsrRepresentationComponent ();

    // initialize
    initializeMsrBasicTypes ();

    pPrivateThisMethodHasBeenRun = true;
  }
}
```

36.2.1 MSR initialization

`src/formats/msr/msrBasicTypes.h/.cpp` defines function `initializeMSRBasicTypes` `()` for this initialization:

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

  if (! pPrivateThisMethodHasBeenRun) {
#ifdef TRACING_IS_ENABLED
    if (gGlobalOahEarlyOptions.getEarlyTracingOah () && ! gGlobalOahEarlyOptions.
    getEarlyQuietOption ()) {
      gLogStream <<
```

```
10          "Initializing MSR basic types handling" <<
11          endl;
12    }
13 #endif
14
15     // languages handling
16     // ------------------------------------------------------
17
18     initializeQuarterTonesPitchesLanguageKinds ();
19
20     // clefs handling
21     // ------------------------------------------------------
22
23     initializeClefKinds ();
24
25     // harmonies handling
26     // ------------------------------------------------------
27
28     initializeHarmonyKinds ();
29
30     // harmony structures handling
31     // ------------------------------------------------------
32
33     initializeHarmonyStructuresMap ();
34
35     // MSR lengths handling
36     // ------------------------------------------------------
37
38     initializeMsrLengthUnitKindsMap ();
39
40     // MSR margins types handling
41     // ------------------------------------------------------
42
43     initializeMsrMarginTypeKindsMap ();
44
45     pPrivateThisMethodHasBeenRun = true;
46    }
47 }
```

36.2.2   LPSR initialization

36.2.3   BSR initialization

36.3   Passes initializations

36.4   Converters initializations

The converters create only the global OAH groups they need. Since the order of initializations is critical, initialization of the formats is done when the latter's insider handler is created.

This is how class `xml2lyInsiderHandler` initializes the MSR and LSPR formats in method `xml2lyInsiderHandler::createTheXml2lyOptionGroups ()` in `src/converters/musicxml2lilypond/musicxml2lilypondInsiderHandler.cpp`:

```
1 void xml2lyInsiderHandler::createTheXml2lyOptionGroups (
2   string serviceName)
3 {
4   // ... ... ...
5
6   // initialize options handling, phase 1
```

```
 7    // ------------------------------------------------------
 8
 9    // create the OAH OAH group first
10    appendGroupToHandler (
11      createGlobalOahOahGroup (
12        serviceName));
13
14    // create the WAE OAH group
15    appendGroupToHandler (
16      createGlobalWaeOahGroup ());
17
18  #ifdef TRACING_IS_ENABLED
19    // create the tracing OAH group
20    appendGroupToHandler (
21      createGlobalTracingOahGroup (
22        this));
23  #endif
24
25    // create the output file OAH group
26    appendGroupToHandler (
27      createGlobalOutputFileOahGroup ());
28
29    // initialize the library
30    // ------------------------------------------------------
31
32    initializeMSR ();
33    initializeLPSR ();
34
35    // initialize options handling, phase 2
36    // ------------------------------------------------------
37
38    // create the MXSR OAH group
39    appendGroupToHandler (
40      createGlobalMxsrOahGroup ());
41
42    // create the mxsr2msr OAH group
43    appendGroupToHandler (
44      createGlobalMxsr2msrOahGroup (
45        this));
46
47    // create the MSR OAH group
48    appendGroupToHandler (
49      createGlobalMsrOahGroup ());
50
51    // create the msr2msr OAH group
52    appendGroupToHandler (
53      createGlobalMsr2msrOahGroup ());
54
55    // create the msr2lpsr OAH group
56    appendGroupToHandler (
57      createGlobalMsr2lpsrOahGroup ());
58
59    // create the LPSR OAH group
60    appendGroupToHandler (
61      createGlobalLpsrOahGroup ());
62
63    // create the LilyPond generation OAH group
64    appendGroupToHandler (
65      createGlobalLpsr2lilypondOahGroup ());
66
67  #ifdef EXTRA_OAH_IS_ENABLED
68    // create the extra OAH group
69    appendGroupToHandler (
70      createGlobalHarmoniesExtraOahGroup ());
71  #endif
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 37

The OAH atoms collection

These handly general-purpose OAH atoms are used in MusicFormats itself. They are defined in `src/oah/oahAtomsCol`

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

37.1   OAH macro atoms

A OAH macro atom is a combination, a list of several options under a single name. The `oahMacroAtom` class is defined in `src/oah/oahAtomsCollection.h/.cpp`:

```
1  class EXP oahMacroAtom : public oahAtom
2  {
3  /*
4    a list of atoms
5  */
6
7    // ... ... ...
8
9    public:
10
11     // public services
12     // ------------------------------------------------------------
13
14     void                    appendAtomToMacro (S_oahAtom atom);
15
16     void                    applyElement (ostream& os) override;
17
18
19
20    private:
21
22     // private fields
23     // ------------------------------------------------------------
24
25      list<S_oahAtom>       fMacroAtomsList;
26 };
```

Populating field `oahMacroAtom::fMacroAtomsList` is straightfoward:

```
1  void oahMacroAtom::appendAtomToMacro (S_oahAtom atom)
2  {
3    // sanity check
4    mfAssert (
5      __FILE__, __LINE__,
6      atom != nullptr,
7      "atom is null");
8
9    fMacroAtomsList.push_back (atom);
10 }
```

Applying the macro atom is done in method `oahMacroAtom::applyElement ()`:

```
1  void oahMacroAtom::applyElement (ostream& os)
2  {
3  #ifdef TRACING_IS_ENABLED
4    if (gGlobalOahEarlyOptions.getEarlyTracingOah ()) {
5      gLogStream <<
6        "==> option '" << fetchNames () << "' is a oahMacroAtom" <<
7        endl;
8    }
9  #endif
10
11   for (
12     list<S_oahAtom>::const_iterator i =
13       fMacroAtomsList.begin ();
14     i != fMacroAtomsList.end ();
15     ++i
16   ) {
17     S_oahAtom atom = (*i);
18
19     if (
20       // oahAtomStoringAValue?
21       S_oahAtomStoringAValue
22         atomWithVariable =
23           dynamic_cast<oahAtomStoringAValue*>(&(*atom))
```

```
24        ) {
25 //        atomWithVariable -> JMI ???
26 //          applyAtomWithValue (theString, os);
27     }
28     else {
29       // valueless atom
30       atom->
31         applyElement (os);
32     }
33   } // for
34 }
```

37.2 A OAH macro atom example

xml2brl has the -auto-utf8, -au8d option:

```
1 jacquesmenu@macmini > xml2brl -query auto-utf8d
2 --- Help for atom "auto-utf8d" in subgroup "Files"
3     -auto-utf8d, -au8d
4           Combines -auto-output-file-name, -utf8d and -use-encoding-in-file-name
```

This macro options is defined in src/formatsgeneration/brailleGeneration/brailleGenerationOah.cpp the following way:

```
1 void brailleGenerationOahGroup::initializeMacroOptions ()
2 {
3   S_oahSubGroup
4     subGroup =
5       oahSubGroup::create (
6         "Macros",
7         "help-braille-generation-macros", "hbgm",
8 R"()",
9         oahElementVisibilityKind::kElementVisibilityWhole,
10        this);
11
12   appendSubGroupToGroup (subGroup);
13
14   // create the auto utfd8 macro
15
16   S_oahMacroAtom
17     autoUTFd8MacroAtom =
18       oahMacroAtom::create (
19         "auto-utf8d", "au8d",
20         "Combines -auto-output-file-name, -utf8d and -use-encoding-in-file-name");
21
22   subGroup->
23     appendAtomToSubGroup (
24       autoUTFd8MacroAtom);
25
26   // populate it
27   autoUTFd8MacroAtom->
28     appendAtomToMacro (
29       gGlobalOutputFileOahGroup->getAutoOutputFileNameAtom ());
30
31   fBrailleOutputKindAtom->
32     applyAtomWithValue (
33       "utf8d",
34       gLogStream);
35   autoUTFd8MacroAtom->
36     appendAtomToMacro (
37       fBrailleOutputKindAtom);
38
39   autoUTFd8MacroAtom->
```

```
40      appendAtomToMacro (
41        fUseEncodingInFileNameAtom);
42 }
```

37.3   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    string                fVariableName;
11    msrOctaveEntryKind    fOctaveEntryKind;
12
13    Bool                  fSetByUser;
14 };
```

The three classes:

- `lilypondAbsoluteOctaveEntryAtom`

- `lilypondRelativeOctaveEntryAtom`

- `lilypondFixedOctaveEntryAtom`

all contain an alias for an class `msrOctaveEntryVariable` variable:

```
1    // private fields
2    // ---------------------------------------------------------
3
4    msrOctaveEntryVariable&
5                      fOctaveEntryKindVariable;
```

The `fOctaveEntryVariable` filed of class `lpsr2lilypondOahGroup` shared be all three options atoms is:

```
1    // notes
2    // -------------------------------------
3
4    msrOctaveEntryVariable
5                      fOctaveEntryVariable;
```

Chapter 38

Measures handling

Measures are presented at section 19.30, [Measures], page 173.

38.1 Voices contents

Class `msrVoice` contain a list of the first elements and a last segment:

```
1    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 'list<S_msrElement>'
9    // because it is not a mere S_msrElement, but a S_msrSegment
10   S_msrSegment        fVoiceLastSegment;
```

38.2 Voice elements

The class `msrVoiceElement` subclasses 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/msrFullMeasureRests.h:class EXP msrFullMeasureRests : 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    list<S_msrMeasure>   fSegmentMeasuresList;
```

Class contains a list of mesure elements:

```
1    // elements
2
3    list<S_msrMeasureElement>
4                       fMeasureElementsList;
```

38.3   Measure elements

The class `msrMeasureElements` subclasses 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 msrFiguredBassElements.h:class EXP msrFiguredBassElement : 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 msrSegnos.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 msrTranspose : public msrMeasureElement
31 msrTupletElements.h:class EXP msrTupletElement : public msrMeasureElement
32 msrVoiceStaffChanges.h:class EXP msrVoiceStaffChange : public msrMeasureElement
```

38.4   Appending measure elements to a measure

Appending music elements to a measure is done by method `msrMeasure::appendElementToMeasure ()`,
defined in `src/formats/msr/msrMeasures.h/.cpp`:

```
1  void msrMeasure::appendElementToMeasure (S_msrMeasureElement elem)
2  {
3    int inputLineNumber =
4      elem->getInputLineNumber ();
5
6  #ifdef TRACING_IS_ENABLED
7    if (gGlobalTracingOahGroup->getTraceMeasures ()) {
8      gLogStream <<
9        "Appending element " <<
10       elem->asShortString () <<
11       " to measure " <<
12       asShortString () <<
13       " in voice \"" <<
14       fetchMeasureVoiceUpLink ()->
15         getVoiceName () <<
16       "\", currentMeasureWholeNotesDuration = " <<
17       fCurrentMeasureWholeNotesDuration <<
18       ", line " << inputLineNumber <<
19       endl;
```

```
20    }
21  #endif
22
23    // set elem's measure number
24    elem->
25      setMeasureElementMeasureNumber (
26        fMeasureElementMeasureNumber);
27
28    // set elem's position in measure
29    elem->
30      setMeasureElementPositionInMeasure (
31        fCurrentMeasureWholeNotesDuration,
32        "appendElementToMeasure()");
33
34    fMeasureElementsList.push_back (elem);
35
36    // take elem's sounding whole notes into account JMI ???
37  if (false) // JMI CAFE
38    incrementCurrentMeasureWholeNotesDuration (
39      inputLineNumber,
40      elem->
41        getMeasureElementSoundingWholeNotes ());
42  }
```

Here is how a harmony instance is appended to a measure:

```
1  void msrMeasure::appendHarmonyToMeasure (S_msrHarmony harmony)
2  {
3    int inputLineNumber =
4      harmony->getInputLineNumber ();
5
6  #ifdef TRACING_IS_ENABLED
7    if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
8      gLogStream <<
9        "Appending harmony " << harmony->asString () <<
10       " to measure " <<
11       this->asShortString () <<
12       " in segment '" <<
13       fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
14       "' in voice \"" <<
15       fMeasureSegmentUpLink->
16         getSegmentVoiceUpLink ()->
17           getVoiceName () <<
18       "\", currentMeasureWholeNotesDuration = " <<
19       fCurrentMeasureWholeNotesDuration <<
20       ", line " << inputLineNumber <<
21       endl;
22    }
23  #endif
24
25    // set harmony's measure number
26    harmony->
27      setMeasureElementMeasureNumber (
28        fMeasureElementMeasureNumber);
29
30    // append the harmony to the measure elements list
31    // DON'T call 'appendElementToMeasure (harmony)':
32    // that would override harmony's position in measure,
33    // which already has the correct value, thus:
34    fMeasureElementsList.push_back (harmony);
35
36    // get harmony sounding whole notes
37    rational
38      harmonySoundingWholeNotes =
39        harmony->
40          getHarmonySoundingWholeNotes ();
```

```
41
42    // account for harmony duration in measure whole notes
43    incrementCurrentMeasureWholeNotesDuration (
44      inputLineNumber,
45      harmonySoundingWholeNotes);
46
47    // this measure contains music
48    fMeasureContainsMusic = true;
49 }
```

The task is simpler when appending a harmony to a measure clone, because the clone's harmony's measure number comes from the clone's original:

```
1 void msrMeasure::appendHarmonyToMeasureClone (S_msrHarmony harmony)
2 {
3   int inputLineNumber =
4     harmony->getInputLineNumber ();
5
6 #ifdef TRACING_IS_ENABLED
7   if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
8     gLogStream <<
9       "Appending harmony " << harmony->asString () <<
10      " to measure clone " <<
11      this->asShortString () <<
12      " in segment clone '" <<
13      fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
14      "' in voice clone \"" <<
15      fMeasureSegmentUpLink->
16        getSegmentVoiceUpLink ()->
17          getVoiceName () <<
18      "\", currentMeasureWholeNotesDuration = " <<
19      fCurrentMeasureWholeNotesDuration <<
20      ", line " << inputLineNumber <<
21      endl;
22  }
23 #endif
24
25   ++gIndenter;
26
27   // append the harmony to the measure elements list
28   appendElementToMeasure (harmony);
29
30   // fetch harmony sounding whole notes
31   rational
32     harmonySoundingWholeNotes =
33       harmony->
34         getHarmonySoundingWholeNotes ();
35
36   // account for harmony duration in measure whole notes
37   incrementCurrentMeasureWholeNotesDuration (
38     inputLineNumber,
39     harmonySoundingWholeNotes);
40
41   // this measure contains music
42   fMeasureContainsMusic = true;
43
44   --gIndenter;
45 }
```

38.5   Appending measures to a segment

Measures are appended to a segment by method `msrSegment::appendMeasureToSegment ()` in
`src/formats/msr/msrSegments.h/.cpp`:

```cpp
void msrSegment::appendMeasureToSegment (S_msrMeasure measure)
{
  int inputLineNumber =
    measure->getInputLineNumber ();

  string measureNumber =
    measure->getMeasureElementMeasureNumber ();

  unsigned int segmentMeasuresListSize =
    fSegmentMeasuresList.size ();

  string currentMeasureNumber =
    segmentMeasuresListSize == 0
      ? ""
      : fSegmentMeasuresList.back ()->getMeasureElementMeasureNumber ();

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceMeasures ()) {
    gLogStream <<
      "Appending measure '" << measureNumber <<
      "' to segment " << asString ();

    if (fSegmentMeasuresList.size () == 0)
      gLogStream <<
        ", as first measure";
    else
      gLogStream <<
      ", after measure number '" << currentMeasureNumber << "'";

    gLogStream <<
      "' in voice \"" <<
      fSegmentVoiceUpLink->getVoiceName () <<
      "\"" <<
      ", line " << measure->getInputLineNumber () <<
      endl;
  }
#endif

  if (measureNumber == currentMeasureNumber) {
    stringstream s;

    s <<
      "appending measure number '" << measureNumber <<
      "' occurs twice in a row in voice \"" <<
      fSegmentVoiceUpLink->getVoiceName () <<
      "\"";

//    msrInternalWarning ( // JMI
    msrInternalError (
      gGlobalServiceRunData->getInputSourceName (),
      inputLineNumber,
      __FILE__, __LINE__,
      s.str ());
  }

  // is measure the first one this segment?
  if (segmentMeasuresListSize == 0) {
    measure->
      setMeasureFirstInSegmentKind (
        msrMeasure::kMeasureFirstInSegmentKindYes);
  }
  else {
    measure->
      setMeasureFirstInSegmentKind (
        msrMeasure::kMeasureFirstInSegmentKindNo);
  }
```

```
67
68    // is measure the first one it the voice?
69    // this is necessary for voice clones ,
70    // which don't go down the part - staff -voice - segment hierarchy
71    if (! fSegmentVoiceUpLink ->getVoiceFirstMeasure ()) {
72      // yes, register it as such
73      fSegmentVoiceUpLink ->
74        setVoiceFirstMeasure (measure);
75
76      measure ->
77        setMeasureFirstInVoice ();
78    }
79
80    // append measure to the segment
81    fSegmentMeasuresList.push_back (measure);
82 }
```

Calls to method `msrSegment::appendMeasureToSegment ()` occur in:

- method `msrSegment::createAMeasureAndAppendItToSegment ()` called from:

  method `msrVoice::createAMeasureAndAppendItToVoice ()`

- method `msrVoice::appendMeasureCloneToVoiceClone ()` called from:

  visitor method `msr2msrTranslator::visitStart (S_msrMeasure& elt)`

- method `msrFullMeasureRests::appendMeasureCloneToFullMeasureRests ()` called from:

  visitor method `msr2msrTranslator::visitStart (S_msrMeasure& elt)`

- method `msrVoice::appendMeasureCloneToVoiceClone ()` called from:

  visitor method `msr2lpsrTranslator::visitStart (S_msrMeasure& elt)`

- method `msrVoice::createNewLastSegmentFromItsFirstMeasureForVoice ()` called from:

  method `msrVoice::handleVoiceLevelRepeatStart ()`,
  method `msrVoice::handleVoiceLevelRepeatEndingStartWithoutExplicitStart ()`,
  method `msrVoice::handleVoiceLevelRepeatEndingStartWithExplicitStart ()`,
  method `msrVoice::createMeasureRepeatFromItsFirstMeasures ()`,
  method `msrVoice::appendPendingMeasureRepeatToVoice ()`,
  method `msrVoice::createFullMeasureRestsInVoice ()`

38.6   Appending measures to a voice

Method `msrVoice::appendMeasureCloneToVoiceClone ()` does the job in `src/formats/msr/msrVoices.h/.cpp`.

```
1  S_msrMeasure msrVoice::createAMeasureAndAppendItToVoice (
2    int     inputLineNumber ,
3    string measureNumber ,
4    msrMeasureImplicitKind
5          measureImplicitKind)
6  {
7    fVoiceCurrentMeasureNumber = measureNumber;
8
9  #ifdef TRACING_IS_ENABLED
10   if (gGlobalTracingOahGroup ->getTraceMeasures ()) {
11     gLogStream <<
12       "Creating measure '" <<
```

```
13        measureNumber <<
14        "' and appending it to voice \"" << getVoiceName () << "\"" <<
15        "', line " << inputLineNumber <<
16        endl;
17    }
18 #endif
19
20    fCallsCounter++;
21
22    if (
23 //      true
24 //        ||
25      false
26        &&
27      (
28        fCallsCounter == 2 && getVoiceName ()
29          ==
30        "Part_POne_HARMONIES_Staff_Voice_Eleven_HARMONIES"
31      )
32    ) { // POUSSE JMI
33      gLogStream <<
34        endl <<
35        "++++ createAMeasureAndAppendItToVoice() POUSSE, fCallsCounter: " << fCallsCounter
      << " ++++" <<
36        endl;
37      this->print (gLogStream);
38      gLogStream <<
39        endl;
40    }
41
42 #ifdef TRACING_IS_ENABLED
43    if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
44      displayVoice (
45        inputLineNumber,
46        "createAMeasureAndAppendItToVoice() 1");
47    }
48 #endif
49
50    ++gIndenter;
51
52    // create the voice last segment if needed
53    if (! fVoiceLastSegment) {
54      createNewLastSegmentForVoice (
55        inputLineNumber,
56        "createAMeasureAndAppendItToVoice() 2");
57    }
58
59    // append a new measure with given number to voice last segment
60    S_msrMeasure
61      result =
62        fVoiceLastSegment->
63          createAMeasureAndAppendItToSegment (
64            inputLineNumber,
65            measureNumber,
66            measureImplicitKind);
67
68    // result is the new voice last appended measure
69    fVoiceLastAppendedMeasure = result;
70
71 #ifdef TRACING_IS_ENABLED
72    if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
73      displayVoice (
74        inputLineNumber,
75        "createAMeasureAndAppendItToVoice() 3");
76    }
77 #endif
78
```

```
79    --gIndenter;
80
81    return result;
82 }
```

38.7   Translating from MXSR to MSR

This is done in src/passes/mxsr2msr/.

Upon the first visit of S_measure, as class msrMeasure instance is created and append to the current part:

```cpp
1  void mxsr2msrTranslator::visitStart (S_measure& elt)
2  {
3    // ... ... ...
4
5    // take this measure into account
6    ++fPartMeasuresCounter;
7
8    // ... ... ...
9
10   // implicit
11
12 /*
13   Measures with an implicit attribute set to "yes"
14   never display a measure number,
15   regardless of the measure-numbering setting.
16   *
17   The implicit attribute is set to "yes" for measures where
18   the measure number should never appear, such as pickup
19   measures and the last half of mid-measure repeats. The
20   value is "no" if not specified.
21 */
22
23   string
24     implicit =
25       elt->getAttributeValue ("implicit");
26
27   msrMeasureImplicitKind
28     measureImplicitKind =
29       msrMeasureImplicitKind::kMeasureImplicitKindNo; // default value
30
31   if       (implicit == "yes") {
32     measureImplicitKind =
33       msrMeasureImplicitKind::kMeasureImplicitKindYes;
34   }
35   else  if (implicit == "no") {
36     measureImplicitKind =
37       msrMeasureImplicitKind::kMeasureImplicitKindNo;
38   }
39   else {
40     if (implicit.size ()) {
41       stringstream s;
42
43       s <<
44         "implicit \"" << implicit <<
45         "\" is unknown";
46
47       musicxmlError (
48         gGlobalServiceRunData->getInputSourceName (),
49         inputLineNumber,
50         __FILE__, __LINE__,
51         s.str ());
52     }
```

```
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 }
```

38.8   Translating from MXSR to MSR

A new class `msrMeasure` instance is created in `src/passes/mxsr2msr/mxsr2msrTranslator.cpp` upon the first visit of `S_measure`:

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

38.9 Translating from MSR to MSR

This is done in `src/passes/msr2msr/`.

38.10 Translating from MSR to LPSR

This is done in `src/passes/msr2lpsr/`.

38.11 Translating from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

Chapter 39

Positions in measures

Positions in measures is stored in field `msrMeasureElement::fMeasureElementPositionInMeasure` in class `msrMeasureElement`, defined in `src/formats/msr/msrMeasureElement.h/.cpp`:

```cpp
class EXP msrMeasureElement : public msrElement
{
  // ... ... ...

  protected:

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

    /*
      The measure uplink is declared in the sub-classes,
      to allow for separate *.h files, C++ constraint
    */

    rational              fMeasureElementSoundingWholeNotes;

    string                fMeasureElementMeasureNumber;

    rational              fMeasureElementPositionInMeasure;
    rational              fMeasureElementPositionInVoice;

    msrMoment             fMeasureElementMomentInMeasure;
    msrMoment             fMeasureElementMomentInVoice;
};
```

39.1   Determining positions in measures

Chapter 40

Finalizations

40.1 Clones vs non-clones finalization

Finalizing clones may be simpler that finalizing a just-created and populated non-clone, due to the information available in the clone's original.

For example, method `msrMeasure::finalizeMeasure ()` delegates part of the job to methods handling the three kinds of voices, respectively:

```cpp
void msrMeasure::finalizeMeasure (
  int                          inputLineNumber,
  msrMeasureRepeatContextKind measureRepeatContextKind,
  string                       context)
{
  // ... ... ...

    S_msrVoice
      voice =
        fMeasureSegmentUpLink->
          getSegmentVoiceUpLink ();

  // ... ... ...

    // delegate to voice kind specific methods
    switch (voice->getVoiceKind ()) {
      case msrVoiceKind::kVoiceKindRegular:
        finalizeRegularMeasure (
          inputLineNumber,
          measureRepeatContextKind,
          context);
        break;

      case msrVoiceKind::kVoiceKindDynamics:
        break;

      case msrVoiceKind::kVoiceKindHarmonies:
        finalizeHarmoniesMeasure (
          inputLineNumber,
          measureRepeatContextKind,
          context);
        break;

      case msrVoiceKind::kVoiceKindFiguredBass:
        finalizeFiguredBassMeasure (
          inputLineNumber,
          measureRepeatContextKind,
          context);
        break;
```

```
40      } // switch
41
42    // ... ... ...
43  }
```

In the case of harmony and figured bass voices, padding may have to be added to obtain a complete measure. This does not happen for clones of such voices: the padding skips are in the original voice and will be visited and handled without anything special to be done.

40.2 The finalization methods

There is a set of virtual method `finalize* ()` methods in MusicFormats. There basic ones are:

- method `msrPart::finalizePart ()` and method `msrPart::finalizePartClone ()`,
  defined in src/formats/msr/msrParts.h/.cpp

- method `msrStaff::finalizeStaff ()`,
  defined in src/formats/msr/msrStaves.h/.cpp

- method `msrVoice::finalizeVoice ()`,
  defined in src/formats/msr/msrVoices.h/.cpp

- method `msrSegment::finalizeAllTheMeasuresOfSegment ()`,
  defined in src/formats/msr/msrSegments.h/.cpp

- method `msrMeasure::finalizeMeasure ()`, method `msrMeasure::finalizeMeasureClone ()` and
  method `msrMeasure::finalizeRegularMeasure ()`,
  defined in src/formats/msr/msrMeasures.h/.cpp

- method `msrChord::finalizeChord ()`,
  defined in src/formats/msr/msrChords.h/.cpp

- method `msrTuplets::finalizeTuplet ()`,
  defined in src/formats/msr/msrTuplets.h/.cpp

- method `mxsr2msrTranslator::finalizeTupletAndPopItFromTupletsStack ()`,
  defined in src/passes/mxsr2msr/mxsr2msrTranslator.h.h/.cpp

- method `msrMeasure::finalizeFiguredBassMeasure ()`,
  defined in src/formats/msr/msrMeasures.h/.cpp

- method `msrMeasure::finalizeHarmoniesMeasure ()`,
  defined in src/formats/msr/msrMeasures.h/.cpp

- method `msr2bsrTranslator::finalizeCurrentMeasureClone ()`,
  defined in src/passes/msr2bsr/(.h/.cppmsr2bsrTranslator)

- method `mxsr2msrTranslator::finalizeCurrentChord ()`,
  defined in src/passes/mxsr2msr/mxsr2msrTranslator.h/.cpp

Handling repeats is rather complex in MusicFormats. Repeat ends are finalized with these methods:

- method `msrPart::finalizeRepeatEndInPart ()`,
  defined in src/formats/msr/msrParts.h/.cpp

- method `msrStaff::finalizeRepeatEndInStaff ()`,
  defined in src/formats/msr/msrStaves.h/.cpp

- method `msrVoice::finalizeRepeatEndInVoice ()`,
  defined in src/formats/msr/msrVoices.h/.cpp

There are also 'cascading' finalization methods: they propagate finalization going from class `msrPart` towards class `msrVoice`:

- method `msrPart::finalizeLastAppendedMeasureInPart ()`,
  defined in src/formats/msr/msrParts.h/.cpp

- method `msrStaff::finalizeLastAppendedMeasureInStaff ()`,
  defined in src/formats/msr/msrStaves.h/.cpp

- method `msrVoice::finalizeLastAppendedMeasureInVoice ()`,
  defined in src/formats/msr/msrVoices.h/.cpp

- method `msrPart::finalizePartAndAllItsMeasures ()`,
  defined in src/formats/msr/msrParts.h/.cpp

- method `msrVoice::finalizeVoiceAndAllItsMeasures ()`,
  defined in src/formats/msr/msrVoices.h/.cpp

A typical example of cascadingfinalization is method `msrPart::finalizePartAndAllItsMeasures ()`:

```cpp
void msrPart::finalizePartAndAllItsMeasures (
  int inputLineNumber)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceParts ()) {
    gLogStream <<
      "Finalizing part clone " <<
      getPartCombinedName () <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceVoices ()) {
    gLogStream <<
      "Finalizing all the measures of part \"" <<
      getPartCombinedName () <<
      "\", line " << inputLineNumber <<
      endl;
  }
#endif

  for (
    list<S_msrVoice>::const_iterator i = fPartAllVoicesList.begin ();
    i != fPartAllVoicesList.end ();
    ++i
  ) {
    S_msrVoice voice = (*i);

    voice->
      finalizeVoiceAndAllItsMeasures (
        inputLineNumber);
  } // for

  // collect the part measures slices from the staves
  collectPartMeasuresSlices (
    inputLineNumber);
}
```

40.3 Finalizing parts

Method `msrPart::finalizePart ()` warns if there are no staves in the part, and otherwise handles them, including cascadingto method `msrStaff::finalizeStaff ()`:

```
void msrPart::finalizePart (
  int inputLineNumber)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceParts ()) {
    gLogStream <<
      "Finalizing part " <<
      getPartCombinedName () <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  ++gIndenter;

  if (! getPartStaveNumbersToStavesMap.size ()) {
    stringstream s;

    s <<
      "Part " <<
      getPartCombinedName () <<
      " appears in the part list, but doesn't contain any stave";

    musicxmlWarning (
      gGlobalServiceRunData->getInputSourceName (),
      inputLineNumber,
      s.str ());
  }

  else {
    // sort the staves to have harmonies above and figured bass below the part
    fPartAllStavesList.sort (
      compareStavesToHaveFiguredBassElementsBelowCorrespondingPart);

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

      staff->
        finalizeStaff (
          inputLineNumber);
    } // for
  }

  // set score instrument names max lengthes if relevant
  setPartInstrumentNamesMaxLengthes ();

  // collect the part measures slices from the staves
  collectPartMeasuresSlices (
    inputLineNumber);

  --gIndenter;
}
```

40.4   Finalizing staves

Method `msrStaff::finalizeStaff ()` cascade s to method `msrVoice::finalizeVoice ()` and collects the staff measusres slices:

```cpp
void msrStaff::finalizeStaff (int inputLineNumber)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceStaves ()) {
    gLogStream <<
      "Finalizing staff \"" <<
      getStaffName () << "\"" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  ++gIndenter;

  // finalize the voices
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceVoices ()) {
    gLogStream <<
      "Finalizing the voices in staff \"" <<
      getStaffName () << "\"" <<
      ", line " << inputLineNumber <<
      endl;
  }
#endif

  for (
    map<int, S_msrVoice>::const_iterator i =
      fStaffVoiceNumbersToAllVoicesMap.begin ();
    i != fStaffVoiceNumbersToAllVoicesMap.end ();
    ++i
  ) {
    S_msrVoice
      voice = (*i).second;

    voice->
      finalizeVoice (
        inputLineNumber);
  } // for

  // collect the staff measures slices from the voices
  collectStaffMeasuresSlices (
    inputLineNumber);

  --gIndenter;
}
```

40.5   Finalizing voices

Method `msrVoice::finalizeVoice ()` handles pending repeats if any and collects the voice measures into a flat list. It does not, however, cascade to finalizing the voice repeats and measures.

```cpp
void msrVoice::finalizeVoice (
  int inputLineNumber)
{
  // ... ... ...

  if (fVoiceHasBeenFinalized) {
```

```
 7      stringstream s;
 8
 9      s <<
10        "Attempting to finalize  voice \"" <<
11        asShortString () <<
12        "\" more than once";
13
14      msrInternalError (
15        gGlobalServiceRunData->getInputSourceName (),
16        fInputLineNumber,
17        __FILE__, __LINE__,
18        s.str ());
19    }
20
21    // set part shortest note duration if relevant
22    S_msrPart
23      voicePart =
24        fetchVoicePartUpLink ();
25
26    rational
27      partShortestNoteDuration =
28        voicePart->
29          getPartShortestNoteDuration ();
30
31    // ... ... ...
32
33    if (fVoiceShortestNoteDuration < partShortestNoteDuration) {
34      // set the voice part shortest note duration
35      voicePart->
36        setPartShortestNoteDuration (
37          fVoiceShortestNoteDuration);
38
39      // set the voice part shortest note tuplet factor // JMI
40      voicePart->
41        setPartShortestNoteTupletFactor (
42          fVoiceShortestNoteTupletFactor);
43    }
44
45    // is this voice totally empty? this should be rare...
46    if (
47      fVoiceInitialElementsList.size () == 0
48        &&
49      fVoiceLastSegment->getSegmentMeasuresList ().size () == 0
50    ) {
51      stringstream s;
52
53      s <<
54        "Voice \"" <<
55        getVoiceName () <<
56        "\" is totally empty, no contents ever specified for it" <<
57        endl;
58
59      musicxmlWarning (
60        gGlobalServiceRunData->getInputSourceName (),
61        inputLineNumber,
62        s.str ());
63    }
64
65    // are there pending repeats in the voice repeats stack???
66    unsigned int voicePendingRepeatDescrsStackSize =
67      fVoicePendingRepeatDescrsStack.size ();
68
69    // ... ... ...
70
71    // collect the voice measures into the flat list
72    collectVoiceMeasuresIntoFlatList (
73      inputLineNumber);
```

```
74
75   fVoiceHasBeenFinalized = true;
76
77   // ... ... ...
78 }
```

40.6   Finalizing repeats

40.7   Finalizing measures

Method `msrMeasure::finalizeMeasure ()` is not cascaded. It delegates finalization to voice kind specific methods presented in the subsections below, handles pending repeats if any, and assigns positions in the measure to the measure's elements:

```cpp
void msrMeasure::finalizeMeasure (
  int                         inputLineNumber,
  msrMeasureRepeatContextKind measureRepeatContextKind,
  string                      context)
{
  if (fMeasureHasBeenFinalized) {
    stringstream s;

    s <<
      "Attempting to finalize measure " <<
      this->asShortString () <<
      " more than once in segment '" <<
      fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
      "', context: " << context <<
      "', measureFinalizationContext: " << fMeasureFinalizationContext <<
      " in voice \"" <<
      fMeasureSegmentUpLink->
        getSegmentVoiceUpLink ()->
          getVoiceName () <<
      "\" (" << context << ")" <<
      ", line " << inputLineNumber;

    // ... ... ...

    msrInternalWarning (
      gGlobalServiceRunData->getInputSourceName (),
      fInputLineNumber,
      s.str ());
  }

  else {
    S_msrVoice
      voice =
        fMeasureSegmentUpLink->
          getSegmentVoiceUpLink ();

    // ... ... ...

    // delegate to voice kind specific methods
    switch (voice->getVoiceKind ()) {
      case msrVoiceKind::kVoiceKindRegular:
        finalizeRegularMeasure (
          inputLineNumber,
          measureRepeatContextKind,
          context);
        break;

      case msrVoiceKind::kVoiceKindDynamics:
```

```
49          break;
50
51        case msrVoiceKind::kVoiceKindHarmonies:
52          finalizeHarmoniesMeasure (
53            inputLineNumber,
54            measureRepeatContextKind,
55            context);
56          break;
57        case msrVoiceKind::kVoiceKindFiguredBass:
58          finalizeFiguredBassMeasure (
59            inputLineNumber,
60            measureRepeatContextKind,
61            context);
62          break;
63     } // switch
64
65     // position in voice
66     rational
67       positionInVoice =
68         fetchMeasureVoiceUpLink ()->
69           getCurrentPositionInVoice ();
70
71     // assign measure' elements position in measure
72     for (
73       list<S_msrMeasureElement>::const_iterator i = fMeasureElementsList.begin ();
74       i != fMeasureElementsList.end ();
75       ++i
76     ) {
77       S_msrMeasureElement measureElement = (*i);
78
79       measureElement->
80         assignMeasureElementPositionInVoice (
81           positionInVoice,
82           "finalizeMeasure()");
83     } // for
84
85     // register finalization
86     fMeasureHasBeenFinalized = true;
87     fMeasureFinalizationContext = context;
88   }
89 }
```

40.7.1   Finalizing regular measures

```
1  void msrMeasure::finalizeRegularMeasure (
2    int                         inputLineNumber,
3    msrMeasureRepeatContextKind measureRepeatContextKind,
4    string                      context)
5  {
6    // fetch the regular voice
7    S_msrVoice
8      voice =
9        fMeasureSegmentUpLink->
10         getSegmentVoiceUpLink ();
11
12   // fetch the regular voice's part
13   S_msrPart
14     regularPart =
15       voice->
16         fetchVoicePartUpLink ();
17
18   mfAssert (
19     __FILE__, __LINE__,
20     regularPart != nullptr,
```

```
21     "regularPart is null");
22
23   if (false) { // JMI
24     gLogStream <<
25       "---> regularPart: " <<
26       endl;
27
28     ++gIndenter;
29     gLogStream <<
30       regularPart <<
31       endl;
32     --gIndenter;
33     gLogStream << endl;
34   }
35
36   rational
37     measureWholeNotesDurationFromPartMeasuresVector =
38       regularPart->
39         getPartMeasuresWholeNotesDurationsVector () [
40           fMeasureOrdinalNumberInVoice - 1 ];
41
42 #ifdef TRACING_IS_ENABLED
43   if (gGlobalTracingOahGroup->getTraceMeasures ()) {
44     gLogStream <<
45       "Finalizing regular measure " <<
46       this->asShortString () <<
47       " in segment '" <<
48       fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
49       "' in regular voice \"" <<
50       voice->getVoiceName () <<
51       "\" (" << context << ")" <<
52       ", measureWholeNotesDurationFromPartMeasuresVector: " <<
53       measureWholeNotesDurationFromPartMeasuresVector <<
54       ", line " << inputLineNumber <<
55       endl;
56   }
57 #endif
58
59   ++gIndenter;
60
61 #ifdef TRACING_IS_ENABLED
62   if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
63     displayMeasure (
64       inputLineNumber,
65       "finalizeRegularMeasure() 1");
66   }
67 #endif
68
69   padUpToPositionAtTheEndOfTheMeasure (
70     inputLineNumber,
71     measureWholeNotesDurationFromPartMeasuresVector);
72
73   // register this measures's length in the part
74   S_msrPart
75     part =
76       this->fetchMeasurePartUpLink ();
77
78   part->
79     registerOrdinalMeasureNumberWholeNotesDuration (
80       inputLineNumber,
81       fMeasureOrdinalNumberInVoice,
82       fCurrentMeasureWholeNotesDuration);
83
84   // determine the measure kind and purist number
85   determineMeasureKindAndPuristNumber (
86     inputLineNumber,
87     measureRepeatContextKind);
```

```
88
89    // pad measure up to whole measure whole notes high tide JMI ???
90    switch (fMeasureKind) {
91      case msrMeasureKind::kMeasureKindCadenza:
92        break;
93
94      case msrMeasureKind::kMeasureKindOvercomplete:
95      case msrMeasureKind::kMeasureKindAnacrusis:
96      case msrMeasureKind::kMeasureKindRegular:
97      case msrMeasureKind::kMeasureKindIncompleteStandalone: // JMI
98      case msrMeasureKind::kMeasureKindIncompleteLastInRepeatCommonPart: // JMI
99      case msrMeasureKind::kMeasureKindIncompleteLastInRepeatHookedEnding: // JMI
100     case msrMeasureKind::kMeasureKindIncompleteLastInRepeatHooklessEnding: // JMI
101     case msrMeasureKind::kMeasureKindIncompleteNextMeasureAfterCommonPart: // JMI
102     case msrMeasureKind::kMeasureKindIncompleteNextMeasureAfterHookedEnding: // JMI
103     case msrMeasureKind::kMeasureKindIncompleteNextMeasureAfterHooklessEnding: // JMI
104       break;
105
106     case msrMeasureKind::kMeasureKindUnknown:
107       // JMI ???
108       break;
109
110     case msrMeasureKind::kMeasureKindMusicallyEmpty:
111       {
112       /* JMI
113           */
114       }
115       break;
116   } // switch
117
118   // is there a single note or rest occupying the full measure?
119   if (fMeasureLongestNote) {
120     if (
121       fMeasureLongestNote-> getNoteSoundingWholeNotes ()
122         ==
123       fFullMeasureWholeNotesDuration
124     ) {
125 #ifdef TRACING_IS_ENABLED
126       if (gGlobalTracingOahGroup->getTraceMeasures ()) {
127         gLogStream <<
128           "Note '" <<
129           fMeasureLongestNote->asShortString () <<
130           "' occupies measure " <<
131           this->asShortString () <<
132           " fully in segment '" <<
133           fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
134           "' in voice \"" <<
135           voice->getVoiceName () <<
136           "\", line " << inputLineNumber <<
137           endl;
138       }
139 #endif
140
141       fMeasureLongestNote->
142         setNoteOccupiesAFullMeasure ();
143     }
144   }
145
146 #ifdef TRACING_IS_ENABLED
147   if (gGlobalTracingOahGroup->getTraceMeasuresDetails ()) {
148     displayMeasure (
149       inputLineNumber,
150       "finalizeRegularMeasure() 2");
151   }
152 #endif
153
154   --gIndenter;
```

```
155 }
```

40.7.2   Finalizing harmonies measures

```
1  void msrMeasure::finalizeHarmoniesMeasure (
2    int                              inputLineNumber,
3    msrMeasureRepeatContextKind measureRepeatContextKind,
4    string                           context)
5  {
6    // fetch the harmonies voice
7    S_msrVoice
8      harmoniesVoice =
9        fMeasureSegmentUpLink->
10         getSegmentVoiceUpLink ();
11
12   // fetch the harmonies part
13   S_msrPart
14     harmoniesPart =
15       harmoniesVoice->
16         fetchVoicePartUpLink ();
17
18   mfAssert (
19     __FILE__, __LINE__,
20     harmoniesPart != nullptr,
21     "harmoniesPart is null");
22
23 #ifdef TRACING_IS_ENABLED
24   if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
25     gLogStream <<
26       "Finalizing harmonies measure " <<
27       this->asShortString () <<
28       " in segment '" <<
29       fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
30       "' in harmonies voice \"" <<
31       harmoniesVoice->getVoiceName () <<
32       "\" (" << context << ")" <<
33       ", line " << inputLineNumber <<
34       endl;
35   }
36 #endif
37
38   ++gIndenter;
39
40 #ifdef TRACING_IS_ENABLED
41   if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
42     displayMeasure (
43       inputLineNumber,
44       "finalizeHarmoniesMeasure() 1");
45   }
46 #endif
47
48 #ifdef TRACING_IS_ENABLED
49   // get the harmoniesPart number of measures
50   int
51     harmoniesPartNumberOfMeasures =
52       harmoniesPart->
53         getPartNumberOfMeasures ();
54
55   if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
56     gLogStream <<
57       "fMeasureOrdinalNumberInVoice = " <<
58       fMeasureOrdinalNumberInVoice <<
59       ", harmoniesPartNumberOfMeasures = " <<
60       harmoniesPartNumberOfMeasures <<
```

```
61       endl;
62   }
63 #endif
64
65   // the measureWholeNotesDuration has to be computed
66   rational
67     measureWholeNotesDurationFromPartMeasuresVector =
68       harmoniesPart->
69         getPartMeasuresWholeNotesDurationsVector () [
70           fMeasureOrdinalNumberInVoice - 1 ];
71
72   // handle the harmonies in this measure
73   finalizeHarmoniesInHarmoniesMeasure (
74     inputLineNumber,
75     context);
76
77   // pad the measure up to measureWholeNotesDurationFromPartMeasuresVector
78   padUpToPositionAtTheEndOfTheMeasure (
79     inputLineNumber,
80     measureWholeNotesDurationFromPartMeasuresVector);
81
82   // determine the measure kind and purist number
83   determineMeasureKindAndPuristNumber (
84     inputLineNumber,
85     measureRepeatContextKind);
86
87 #ifdef TRACING_IS_ENABLED
88   if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
89     displayMeasure (
90       inputLineNumber,
91       "finalizeHarmoniesMeasure () 2");
92   }
93 #endif
94
95   --gIndenter;
96 }
```

40.7.3   Finalizing figured bass measures

```
1 void msrMeasure::finalizeFiguredBassMeasure (
2   int                        inputLineNumber,
3   msrMeasureRepeatContextKind measureRepeatContextKind,
4   string                     context)
5 {
6   // fetch the figured bass voice
7   S_msrVoice
8     figuredBassVoice =
9       fMeasureSegmentUpLink->
10        getSegmentVoiceUpLink ();
11
12   // fetch the figured bass part
13   S_msrPart
14     figuredBassPart =
15      figuredBassVoice->
16        fetchVoicePartUpLink ();
17
18   mfAssert (
19     __FILE__, __LINE__,
20     figuredBassPart != nullptr,
21     "figuredBassPart is null");
22
23 #ifdef TRACING_IS_ENABLED
24   if (gGlobalTracingOahGroup->getTraceFiguredBass ()) {
25     gLogStream <<
```

```
26        "Finalizing figured bass measure " <<
27        this->asShortString () <<
28        " in segment '" <<
29        fMeasureSegmentUpLink->getSegmentAbsoluteNumber () <<
30        "' in figured bass voice \"" <<
31        figuredBassVoice->getVoiceName () <<
32        "\" (" << context << ")" <<
33        ", line " << inputLineNumber <<
34        endl;
35    }
36 #endif
37
38    ++gIndenter;
39
40 #ifdef TRACING_IS_ENABLED
41    if (gGlobalTracingOahGroup->getTraceFiguredBassDetails ()) {
42      displayMeasure (
43        inputLineNumber,
44        "finalizeFiguredBassMeasure() 1");
45    }
46 #endif
47
48 #ifdef TRACING_IS_ENABLED
49    // get the figuredBassPart number of measures
50    int
51      figuredBassPartNumberOfMeasures =
52        figuredBassPart->
53          getPartNumberOfMeasures ();
54
55    if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
56      gLogStream <<
57        "fMeasureOrdinalNumberInVoice = " <<
58        fMeasureOrdinalNumberInVoice <<
59        ", figuredBassPartNumberOfMeasures = " <<
60        figuredBassPartNumberOfMeasures <<
61        endl;
62    }
63 #endif
64
65    // the measureWholeNotesDuration has to be computed
66    rational
67      measureWholeNotesDuration =
68        figuredBassPart->
69          getPartMeasuresWholeNotesDurationsVector () [
70            fMeasureOrdinalNumberInVoice - 1 ];
71
72    // handle the figured bass elements in this measure
73    finalizeFiguredBassElementsInFiguredBassMeasure (
74      inputLineNumber,
75      context);
76
77    // pad the measure up to fFullMeasureWholeNotesDuration
78    padUpToPositionAtTheEndOfTheMeasure (
79      inputLineNumber,
80      measureWholeNotesDuration);
81
82    // determine the measure kind and purist number
83    determineMeasureKindAndPuristNumber (
84      inputLineNumber,
85      measureRepeatContextKind);
86
87 #ifdef TRACING_IS_ENABLED
88    if (gGlobalTracingOahGroup->getTraceFiguredBassDetails ()) {
89      displayMeasure (
90        inputLineNumber,
91        "finalizeFiguredBassMeasure() 2");
92    }
```

```
93  #endif
94
95    --gIndenter;
96  }
```

40.8   Determining positions in measures

Chapter 41

Tempos handling

Tempos are presented at section 19.18, [Tempos], page 161.

41.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

41.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

41.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

41.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

41.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 42

Positions in measures

Positions in measures is stored in field `msrMeasureElement::fMeasureElementPositionInMeasure` in class `msrMeasureElement`, defined in `src/formats/msr/msrMeasureElement.h/.cpp`:

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

  protected:

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

    /*
      The measure uplink is declared in the sub-classes,
      to allow for separate *.h files, C++ constraint
    */

    rational              fMeasureElementSoundingWholeNotes;

    string                fMeasureElementMeasureNumber;

    rational              fMeasureElementPositionInMeasure;
    rational              fMeasureElementPositionInVoice;

    msrMoment             fMeasureElementMomentInMeasure;
    msrMoment             fMeasureElementMomentInVoice;
};
```

42.1 Determining positions in measures

Chapter 43

Notes handling

Notes are presented at section **??**, [Notes], page **??**.

43.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

43.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

43.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

43.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

43.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 44

Segments handling

Segments are presented at section 19.37, [Segments], page 179.

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 (
```

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 (
```

Method `msrSegment::createSegmentDeepClone ()` is not used at the time of this writing.

Explicit segments creation is thus entirely done in methods inside `src/formats/msr/msrVoices.cpp`: the passes are not aware of this happening.

The first occurrence of method `msrSegment::create ()` is in method `msrVoice::initializeVoice ()`: when a voice is created, a segment is created and stored in its `fVoiceLastSegment` if requested :

```
1  void msrVoice::initializeVoice (
2    msrVoiceCreateInitialLastSegmentKind
3      voiceCreateInitialLastSegmentKind)
4  {
5    // ... ... ...
6
7      // create the initial last segment if requested
8    switch (voiceCreateInitialLastSegmentKind) {
9      case msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentYes:
10        // sanity check // JMI LAST
11        mfAssert (
12          __FILE__, __LINE__,
13          fVoiceLastSegment == nullptr,
14          "fVoiceLastSegment is null");
15
16        // create the last segment
17        fVoiceLastSegment =
18          msrSegment::create (
19            fInputLineNumber,
20            this);
21
22        if (! fVoiceFirstSegment) {
23          fVoiceFirstSegment = fVoiceLastSegment;
24        }
25        break;
26      case msrVoiceCreateInitialLastSegmentKind::kCreateInitialLastSegmentNo:
27        break;
28    } // switch
29
30    // ... ... ...
31  };
```

Method `msrVoice::createMeasureRepeatFromItsFirstMeasures ()` is presented in section 46, [Measure repeats handling], page 277, 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 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::handleFullMeasureRestsStartInVoiceClone ()`

- method `msrVoice::handleHooklessRepeatEndingEndInVoice ()`

- method `msrVoice::appendBarLineToVoice ()`

- method `msrVoice::appendSegnoToVoice ()`

- method `msrVoice::appendCodaToVoice ()`

- method `msrVoice::appendEyeGlassesToVoice ()`

- method `msrVoice::appendPedalToVoice ()`

- method `msrVoice::appendDampToVoice ()`

- method `msrVoice::appendDampAllToVoice ()`

44.1.2 Creating a new last segment for a voice from its first measure

Method `msrVoice::createNewLastSegmentFromItsFirstMeasureForVoice ()` is used at several places in
`src/formats/msr/msrVoices.cpp`:

```
void msrVoice::createNewLastSegmentFromItsFirstMeasureForVoice (
  int          inputLineNumber,
  S_msrMeasure firstMeasure,
  string       context)
{
  // create the last segment
  fVoiceLastSegment =
    msrSegment::create (
      inputLineNumber,
      this);

  if (! fVoiceFirstSegment) {
    fVoiceFirstSegment = fVoiceLastSegment;
  }

  // ... ... ...

  // append firstMeasure to fVoiceLastSegment
  fVoiceLastSegment->
    appendMeasureToSegment (firstMeasure);
```

```
21
22    // firstMeasure is the new voice last appended measure
23    fVoiceLastAppendedMeasure = firstMeasure;
24
25    // is firstMeasure the first one it the voice?
26    if (! fVoiceFirstMeasure) {
27      // yes, register it as such
28      setVoiceFirstMeasure (
29        firstMeasure);
30
31      firstMeasure->
32        setMeasureFirstInVoice ();
33    }
34
35    // ... ... ...
36 }
```

All the uses of this method concern repeats (section 48, [Repeats handling], page 279), measure repeats (section 46, [Measure repeats handling], page 277) and full measure rests(section 65, [Full measure rests handling], page 330).

44.2 Appending measures to a segment

Method `msrSegment::assertSegmentMeasuresListIsNotEmpty ()` is called as a sanity check by many methods in `src/formats/msr/msrSegments.cpp`:

```
1  void msrSegment::assertSegmentMeasuresListIsNotEmpty (
2    int inputLineNumber) const
3  {
4    if (! fSegmentMeasuresList.size ()) {
5  #ifdef TRACING_IS_ENABLED
6    if (
7      gGlobalTracingOahGroup->getTraceMeasuresDetails ()
8        ||
9      gGlobalTracingOahGroup->getTraceSegmentsDetails ()
10        ||
11     gGlobalTracingOahGroup->getTraceRepeatsDetails ()
12   ) {
13     fSegmentVoiceUpLink->
14       displayVoiceRepeatsStackFullMeasureRestsMeasureRepeatAndVoice (
15         inputLineNumber,
16         "assertSegmentMeasuresListIsNotEmpty()");
17   }
18 #endif
19
20     gLogStream <<
21       "assertSegmentMeasuresListIsNotEmpty()" <<
22       ", fSegmentMeasuresList is empty" <<
23       ", segment: " <<
24       this->asString () <<
25       ", in voice \"" <<
26       fSegmentVoiceUpLink->getVoiceName () <<
27       "\"" <<
28       "', line " << inputLineNumber <<
29       endl;
30
31     mfAssert (
32       __FILE__, __LINE__,
33       false,
34       ", fSegmentMeasuresList is empty");
35   }
36 }
```

One such call is:

```
void msrSegment::appendKeyToSegment (S_msrKey key)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalTracingOahGroup->getTraceKeys ()) {
    gLogStream <<
      "Appending key " << key->asString () <<
      " to segment " << asString () <<
    ", in voice \"" <<
    fSegmentVoiceUpLink->getVoiceName () <<
    "\"" <<
      endl;
  }
#endif

  // sanity check
  assertSegmentMeasuresListIsNotEmpty (
    key->getInputLineNumber ());

  ++gIndenter;

  // register key in segments's current measure
  fSegmentMeasuresList.back ()->
    appendKeyToMeasure (key);

  --gIndenter;
}
```

44.3   Translating from MXSR to MSR

44.4   Translating from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

44.5   Translating from MSR to MSR

This is done in `src/passes/msr2msr/`.

44.6   Translating from MSR to LPSR

This is done in `src/passes/msr2lpsr/`.

44.7   Translating from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

Chapter 45

Beat repeats handling

Beat repeats are presented at section 19.32, [Beat repeats], page 175.

45.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

45.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

45.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

45.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

45.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 46

Measure repeats handling

Measure repeats are presented at section 19.33, [Measure repeats], page 176.

46.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

method `msrVoice::createMeasureRepeatFromItsFirstMeasures ()`:

46.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

46.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

46.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

46.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 47

Full measure rests handling

Full measure rests are presented at section 19.34, [Full measure rests], page 177.

47.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

47.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

47.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

47.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

47.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 48

Repeats handling

Repeats are presented at section 19.36, [Repeats], page 178.

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 type s are defined in `src/formats/msr/msrBarLines.h` local to class `msrBarLine`:

```
1    // location
2    enum msrBarLineLocationKind {
3      kBarLineLocationNone,
4
5      kBarLineLocationLeft,
6      kBarLineLocationMiddle,
7      kBarLineLocationRight // by default
8    };
```

```
1    // style
2    enum 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 msrBarLineRepeatDirectionKind {
3      kBarLineRepeatDirectionNone,
4      kBarLineRepeatDirectionForward, kBarLineRepeatDirectionBackward
5    };
```

```
1     // ending type
2     enum msrBarLineEndingTypeKind {
3       kBarLineEndingNone ,
4
5       kBarLineEndingTypeStart ,
6       kBarLineEndingTypeStop ,
7       kBarLineEndingTypeDiscontinue
8     };
```

```
1     // category
2     enum msrBarLineCategoryKind {
3       k_NoBarLineCategory ,
4
5       kBarLineCategoryStandalone ,
6
7       kBarLineCategoryRepeatStart , kBarLineCategoryRepeatEnd ,
8
9       kBarLineCategoryHookedEndingStart , kBarLineCategoryHookedEndingEnd ,
10      kBarLineCategoryHooklessEndingStart , kBarLineCategoryHooklessEndingEnd
11    };
```

```
1     // segno
2     enum msrBarLineHasSegnoKind {
3       kBarLineHasSegnoYes , kBarLineHasSegnoNo
4     };
```

```
1     // coda
2     enum msrBarLineHasCodaKind {
3       kBarLineHasCodaYes , kBarLineHasCodaNo
4     };
```

```
1     // repeat winged
2     enum 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      string
9        location =
10         elt->getAttributeValue ("location");
11
12     fCurrentBarLineLocationKind =
13       msrBarLine::kBarLineLocationRight; // by default
14
15     if       (location == "left") {
16       fCurrentBarLineLocationKind = msrBarLine::kBarLineLocationLeft;
17     }
18     else  if (location == "middle") {
19       fCurrentBarLineLocationKind = msrBarLine::kBarLineLocationMiddle;
20     }
21     else if  (location == "right") {
22       fCurrentBarLineLocationKind = msrBarLine::kBarLineLocationRight;
```

```
23        }
24      else {
25        stringstream s;
26
27        s <<
28          "barLine location \"" << location <<
29          "\" is unknown, using 'right' by default";
30
31   // JMI    musicxmlError (
32        musicxmlWarning (
33          gGlobalServiceRunData->getInputSourceName (),
34          inputLineNumber,
35   //     __FILE__, __LINE__,
36          s.str ());
37      }
38    }
39
40    fOnGoingBarLine = true;
41  }
```

Then the class `msrBarLine` instance is created upon the second visit of `S_barline`:

```
1  void mxsr2msrTranslator::visitEnd ( S_barline& elt )
2  {
3    // ... ... ...
4
5    // create the barLine
6    S_msrBarLine
7      barLine =
8        msrBarLine::create (
9          inputLineNumber,
10         fCurrentBarLineLocationKind,
11         fCurrentBarLineStyleKind,
12         fCurrentBarLineRepeatDirectionKind,
13         fCurrentBarLineEndingTypeKind,
14         fCurrentBarLineEndingNumber,
15         fCurrentBarLineTimes,
16         msrBarLine::k_NoBarLineCategory, // 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 msrBarLine::kBarLineLocationNone:
33        // should not occur
34        break;
35
36      case msrBarLine::kBarLineLocationLeft:
37        if (
38          fCurrentBarLineEndingTypeKind
39            ==
40          msrBarLine::kBarLineEndingTypeStart
41        ) {
42          // ending start, don't know yet whether it's hooked or hookless
43          // -------------------------------------------------------
44          if (! fCurrentBarLineEndingNumber.size ()) {
```

```
45        musicxmlWarning (
46          gGlobalServiceRunData->getInputSourceName (),
47          inputLineNumber,
48          "mandatory ending number is missing, assuming \"1\"");
49
50        fCurrentBarLineEndingNumber = "1";
51      }
52
53      // don't know yet whether repeat ending start barLine is hooked or hookless
54      // remember it in fCurrentRepeatEndingStartBarLine,
55      fCurrentRepeatEndingStartBarLine = barLine;
56
57      // handle the repeat ending start
58      handleRepeatEndingStart (barLine);
59
60      barLineHasBeenHandled = true;
61    }
62
63    else if (
64      fCurrentBarLineRepeatDirectionKind
65        ==
66      msrBarLine::kBarLineRepeatDirectionForward
67    ) {
68      // repeat start
69      // -------------------------------------------------------
70      // set the barLine category
71      barLine->
72        setBarLineCategory (
73          msrBarLine::kBarLineCategoryRepeatStart);
74
75      // handle the repeat start
76      handleRepeatStart (barLine);
77
78      barLineHasBeenHandled = true;
79    }
80    break;
81
82  case msrBarLine::kBarLineLocationMiddle:
83    // JMI ???
84    break;
85
86  case msrBarLine::kBarLineLocationRight:
87    {
88      if (
89        fCurrentBarLineEndingTypeKind == msrBarLine::kBarLineEndingTypeStop
90          &&
91        fCurrentBarLineEndingNumber.size () != 0
92      ) {
93        // hooked ending end
94        // -------------------------------------------------------
95        // set current barLine ending start category
96        fCurrentRepeatEndingStartBarLine->
97          setBarLineCategory (
98            msrBarLine::kBarLineCategoryHookedEndingStart);
99
100       // set this barLine's category
101       barLine->
102         setBarLineCategory (
103           msrBarLine::kBarLineCategoryHookedEndingEnd);
104
105       // handle the repeat hooked ending end
106       handleRepeatHookedEndingEnd (barLine);
107
108       barLineHasBeenHandled = true;
109     }
110
111     else if (
```

```cpp
            fCurrentBarLineRepeatDirectionKind
              ==
            msrBarLine::kBarLineRepeatDirectionBackward
          ) {
            // repeat end
            // -------------------------------------------------------

            // set this barLine's category
            barLine->
              setBarLineCategory (
                msrBarLine::kBarLineCategoryRepeatEnd);

            // handle the repeat end
            handleRepeatEnd (barLine);

            barLineHasBeenHandled = true;
          }

          else if (
            fCurrentBarLineEndingTypeKind == msrBarLine::kBarLineEndingTypeDiscontinue
              &&
            fCurrentBarLineEndingNumber.size () != 0
          ) {
            // hookless ending end
            // -------------------------------------------------------
            // set current barLine ending start category
            fCurrentRepeatEndingStartBarLine->
              setBarLineCategory (
                msrBarLine::kBarLineCategoryHooklessEndingStart);

            // set this barLine's category
            barLine->
              setBarLineCategory (
                msrBarLine::kBarLineCategoryHooklessEndingEnd);

            // handle the repeat hookless ending end
            handleRepeatHooklessEndingEnd (barLine);

            barLineHasBeenHandled = true;
          }

          // forget about current repeat ending start barLine
          fCurrentRepeatEndingStartBarLine = nullptr;
      }
      break;
  } // switch

  // set the barLine category to stand alone if not yet handled
  if (! barLineHasBeenHandled) {
    switch (fCurrentBarLineStyleKind) {
      case msrBarLine::kBarLineStyleRegular:
      case msrBarLine::kBarLineStyleDotted:
      case msrBarLine::kBarLineStyleDashed:
      case msrBarLine::kBarLineStyleHeavy:
      case msrBarLine::kBarLineStyleLightLight:
      case msrBarLine::kBarLineStyleLightHeavy:
      case msrBarLine::kBarLineStyleHeavyLight:
      case msrBarLine::kBarLineStyleHeavyHeavy:
      case msrBarLine::kBarLineStyleTick:
      case msrBarLine::kBarLineStyleShort:
        barLine->
          setBarLineCategory (
            msrBarLine::kBarLineCategoryStandalone);

        // append the bar line to the current part
        // ... ... ...
```

```
179          fCurrentPart ->
180            appendBarLineToPart (barLine);
181
182          barLineHasBeenHandled = true;
183          break;
184
185        case msrBarLine::kBarLineStyleNone:
186          stringstream s;
187
188          s <<
189            "barLine " <<
190            barLine ->asString () <<
191            " has no barLine style";
192
193          musicxmlWarning (
194            gGlobalServiceRunData ->getInputSourceName (),
195            inputLineNumber,
196    //        __FILE__, __LINE__,
197            s.str ());
198          break;
199      } // switch
200    }
201
202    // has this barLine been handled?
203    if (! barLineHasBeenHandled) {
204      stringstream s;
205
206      s << left <<
207        "cannot handle a barLine containing: " <<
208        barLine ->asString ();
209
210      msrInternalWarning (
211        gGlobalServiceRunData ->getInputSourceName (),
212        inputLineNumber,
213        s.str ());
214    }
215
216    fOnGoingBarLine = false;
217  }
```

48.2   Translating repeats from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

48.3   Translating repeats from MSR to MSR

This is done in `src/passes/msr2msr/`.

48.4   Translating repeats from MSR to LPSR

This is done in `src/passes/msr2lpsr/`.

48.5   Translating repeats from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

Chapter 49

Voices handling

Voices are presented at section 19.29, [Voices], page 172.

49.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

49.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

49.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

49.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

49.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 50

Staves handling

Staves are presented at section 19.27, [Staves], page 170.

50.1   Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

50.2   Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

50.3   Translating from MSR to MSR (`src/passes/msr2msr/`)

50.4   Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

50.5   Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 51

Parts handling

Parts are presented at section 19.26, [Parts], page 170.

51.1 Parts browsing

Method `msrPart::browseData ()` defined in `src/formats/msr/msrParts.h/.cpp` is pecular in that it imposes a *partial order* on the part staves browsing:

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

#ifdef TRACING_IS_ENABLED // JMI
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) { // JMI TEMP
    gLogStream <<
      "++++++++ fPartAllStavesList.size(): " <<
      fPartAllStavesList.size () <<
      endl;

    if (fPartAllStavesList.size ()) {
      for (S_msrStaff staff : fPartAllStavesList) {
        gLogStream <<
          endl <<
          "++++++++ staff: +++++++" <<
          " \"" << staff->getStaffName () << "\"" <<
          endl;
      } // for
    }

    gLogStream <<
      "++++++++ fPartNonHarmoniesNorFiguredBassStavesList.size(): " <<
      fPartNonHarmoniesNorFiguredBassStavesList.size () <<
      endl;

    if (fPartNonHarmoniesNorFiguredBassStavesList.size ()) {
      for (S_msrStaff staff : fPartNonHarmoniesNorFiguredBassStavesList) {
        gLogStream <<
          endl <<
          "++++++++ staff: +++++++" <<
          " \"" << staff->getStaffName () << "\"" <<
          endl;
      } // for
    }
```

```
40    }
41 #endif
42
43    /* don't enforce any order here, leave it to the client thru sorting JMI */
44
45    // browse the part harmonies staff if any right now, JMI
46    // to place it before the corresponding part
47    if (fPartHarmoniesStaff) {
48      msrBrowser<msrStaff> browser (v);
49      browser.browse (*fPartHarmoniesStaff);
50    }
51
52    // browse all non harmonies and non figured bass staves
53    for (S_msrStaff staff : fPartNonHarmoniesNorFiguredBassStavesList) {
54      // browse the staff
55      msrBrowser<msrStaff> browser (v);
56      browser.browse (*staff);
57    } // for
58
59    // browse the part figured bass staff if any only now, JMI
60    // to place it after the corresponding part
61    if (fPartFiguredBassStaff) {
62      msrBrowser<msrStaff> browser (v);
63      browser.browse (*fPartFiguredBassStaff);
64    }
65
66 //    // browse all the part staves JMI
67 //    for (S_msrStaff staff : fPartAllStavesList) {
68 //      if (staff != fPartHarmoniesStaff && staff != fPartFiguredBassStaff) {
69 //        // browse the staff
70 //        msrBrowser<msrStaff> browser (v);
71 //        browser.browse (*staff);
72 //      }
73 //    } // for
74 }
```

51.2   Translating from MXSR to MSR (src/passes/mxsr2msr/)

51.3   Translating from MXSR to MSR (src/passes/mxsr2msr/)

51.4   Translating from MSR to MSR (src/passes/msr2msr/)

51.5   Translating from MSR to LPSR (src/passes/msr2lpsr/)

51.6   Translating from LPSR to LilyPond (src/passes/lpsr2lilypond/)

Chapter 52

Scores handling

Scores are presented at section 19.24, [Scores], page 169.

52.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

52.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

52.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

52.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

52.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 53

Books handling

Books are presented at section 19.23, [Books], page 169.

53.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

53.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

53.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

53.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

53.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 54

Ornaments handling

Ornaments are presented at section 19.40, [Ornaments], page 181.

54.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

54.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

54.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

54.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

54.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 55

Ties handling

Ties are presented at section 19.41, [Ties], page 181.

55.1   Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

55.2   Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

55.3   Translating from MSR to MSR (`src/passes/msr2msr/`)

55.4   Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

55.5   Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 56

Dynamics handling

Dynamics are presented at section 19.42, [Dynamics], page 181.

56.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

56.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

56.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

56.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

56.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 57

Beams handling

Beams are presented at section 19.43, [Beams], page 181.

57.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

57.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

57.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

57.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

57.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 58

Slurs handling

Slurs are presented at section 19.44, [Slurs], page 181.

58.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

58.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

58.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

58.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

58.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 59

Grace notes groups handling

Grace notes groups are presented at section 19.45, [Grace notes groups], page 181.

59.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

59.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

59.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

59.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

59.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 60

Chords handling

Chords are presented at section 19.46, [Chords], page 181.

60.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

60.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

60.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

60.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

60.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 61

Tuplets handling

Tuplets are presented at section 19.47, [Tuplets], page 181.

61.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

61.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

61.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

61.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

61.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 62

Harmonies handling

Harmonies are presented at section 19.49, [Harmonies], page 184.

The useful options here are:

- option -trace-harmonies, -tharms

- option -display-msr-skeleton, -dmsrskel

- option -display-msr-1, -dmsr1

- option -display-msr-1-short, -dmsr1s and option -display-msr-1-details, -dmsr1d

- option -display-msr-2msr2, -dmsr2

- option -display-msr-2-short, -msr2s and option -display-msr-2-details, -dmsr2d

Harmonies need special treatment since we need to determine their position in a harmony voice. This is different than MusicXML, where they are simply *drawn* at the current music position, so to say.

They are handled this way:

- harmonies are stored in class `msrNote`:

- they are also stored in class `msrPart`, class `msrChord` and class `msrTuplet` (denormalization);

In class `msrNote`, there is:

```
// harmonies
void                    appendHarmonyToNoteHarmoniesList (
                          S_msrHarmony harmony);

const list<S_msrHarmony>&
                        getNoteHarmoniesList () const
                            { return fNoteHarmoniesList; }

// ... ... ...

// harmonies
// ----------------------------------------------------

list<S_msrHarmony>    fNoteHarmoniesList;

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

62.1 Harmonies staves and voices

Every class `msrVoice` instance in MusicFormats belongs to an class `msrStaff` instance. Staves are created specifically to hold harmonies voices, using specific numbers defined in `src/formats/msr/msrParts.h`:

```
public:

  // constants
  // --------------------------------------------------------

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

In class `msrStaff`, there is:

```
    void                  registerHarmoniesVoiceByItsNumber (
                            int          inputLineNumber,
                            S_msrVoice voice);
```

Class `msrPart` also contains:

```
    // harmonies

    S_msrVoice            createPartHarmoniesVoice (
                            int    inputLineNumber,
                            string currentMeasureNumber);

    void                  appendHarmonyToPart (
                            S_msrVoice   harmonySupplierVoice,
                            S_msrHarmony harmony);

    void                  appendHarmonyToPartClone (
                            S_msrVoice    harmonySupplierVoice,
                            S_msrHarmony  harmony);
```

```
    // harmonies

    S_msrStaff            fPartHarmoniesStaff;
    S_msrVoice            fPartHarmoniesVoice;
```

62.2 Harmonies staves creation

This is done in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.cpp.h/.cpp`:

```
S_msrVoice mxsr2msrSkeletonBuilder::createPartHarmoniesVoiceIfNotYetDone (
  int        inputLineNumber,
  S_msrPart  part)
{
  // is the harmonies voice already present in part?
  S_msrVoice
    partHarmoniesVoice =
      part->
        getPartHarmoniesVoice ();

  if (! partHarmoniesVoice) {
    // create the harmonies voice and append it to the part
    partHarmoniesVoice =
      part->
        createPartHarmoniesVoice (
          inputLineNumber,
          fCurrentMeasureNumber);
```

```
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    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 }
```

62.3   Harmonies creation

There are several methods for harmonies creation:

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/src > grep create formats/msr/msrHarmonies.h
2      static SMARTP<msrHarmonyDegree> create (
3      static SMARTP<msrHarmony> createWithoutVoiceUplink (
4      SMARTP<msrHarmony> createHarmonyNewbornClone (
5  //      SMARTP<msrHarmony> createHarmonyDeepClone ( // JMI ???
6      static SMARTP<msrHarmony> createWithVoiceUplink (
```

62.4 Translating harmonies from MXSR to MSR

This is done in `src/passes/mxsr2msr/`.

The MSR score skeleton created in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.h/.cpp` contains the part groups, parts, staves and voices, as well as the number of measures. The voices do not contain any music elements yet.

A harmoniy belongs to a `<part/>` in MusicXML, but we sometimes need to have it attached to a note. When visiting an `S_harmony` element, field `mxsr2msrSkeletonBuilder::fThereAreHarmoniesToBeAttachedToCurrentNote` it used to account for that:

```cpp
void mxsr2msrSkeletonBuilder::visitStart ( S_harmony& elt )
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
    gLogStream <<
      "--> Start visiting S_harmony" <<
      ", harmoniesVoicesCounter = " << fHarmoniesVoicesCounter <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  /* JMI ???
    several harmonies can be attached to a given note,
    leading to as many harmonies voices in the current part
  */

  // take harmonies voice into account
  ++fHarmoniesVoicesCounter; // UNUSED JMI

  fThereAreHarmoniesToBeAttachedToCurrentNote = true;
}
```

Upon the second visit of class `msrNote`, the part harmonies voice is created if harmonies are not to be ignored due to option option `-ignore-harmonies, -oharms` and it has not been created yet:

```cpp
void mxsr2msrSkeletonBuilder::visitEnd ( S_note& elt )
{
  // ... ... ...

  // are there harmonies attached to the current note?
  if (fThereAreHarmoniesToBeAttachedToCurrentNote) {
    if (gGlobalMxsr2msrOahGroup->getIgnoreHarmonies ()) {
#ifdef TRACING_IS_ENABLED
      if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
        gLogStream <<
          "Ignoring the harmonies" <<
          ", line " <<
          inputLineNumber <<
          endl;
      }
#endif
    }
    else {
      // create the part harmonies voice if not yet done
      S_msrVoice
        partHarmoniesVoice =
          createPartHarmoniesVoiceIfNotYetDone (
            inputLineNumber,
            fCurrentPart);
    }

    fThereAreHarmoniesToBeAttachedToCurrentNote = false;
```

```
28    }
29
30    // ... ... ...
31 }
```

Creating the part harmonies voice is delegated to the part:

```
1  S_msrVoice mxsr2msrSkeletonBuilder::createPartHarmoniesVoiceIfNotYetDone (
2    int        inputLineNumber,
3    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 }
```

62.5   Translating harmonies from MXSR to MSR

This is done in src/passes/mxsr2msr/.

This is where the class msrHarmony instances are created.

62.5.1   First S_harmony visit

The first visit of S_harmony initializes the fields storing values to be gathered visiting subelements:

```
1  void mxsr2msrTranslator::visitStart ( S_harmony& elt )
2  {
3    int inputLineNumber =
4      elt->getInputLineNumber ();
5
6  #ifdef TRACING_IS_ENABLED
7    if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
8      gLogStream <<
9        "--> Start visiting S_harmony" <<
10       ", line " << inputLineNumber <<
11       endl;
12   }
13 #endif
14
15   ++fHarmoniesVoicesCounter;
16
17   fCurrentHarmonyInputLineNumber        = inputLineNumber;
18
19   fCurrentHarmonyRootDiatonicPitchKind = msrDiatonicPitchKind::k_NoDiatonicPitch;
20   fCurrentHarmonyRootAlterationKind    = msrAlterationKind::kAlterationNatural;
21   fCurrentHarmonyKind                  = msrHarmonyKind::k_NoHarmony;
22   fCurrentHarmonyKindText              = "";
```

```
23    fCurrentHarmonyInversion               = K_HARMONY_NO_INVERSION;
24    fCurrentHarmonyBassDiatonicPitchKind = msrDiatonicPitchKind::k_NoDiatonicPitch;
25    fCurrentHarmonyBassAlterationKind      = msrAlterationKind::kAlterationNatural;
26    fCurrentHarmonyDegreeValue            = -1;
27    fCurrentHarmonyDegreeAlterationKind  = msrAlterationKind::kAlterationNatural;
28
29    fCurrentHarmonyWholeNotesOffset = rational (0, 1);
30
31    fOnGoingHarmony = true;
32 }
```

62.5.2   Second `S_harmony` visit

Upon the second visit of `S_harmony` , a class `msrHarmony` instance is created, populated and appended to `mxsr2msrTranslatorfPendingHarmoniesList`.

The voice uplink will be set later, hence the use of method `msrHarmony::createWithoutVoiceUplink ():`

```
1  void mxsr2msrTranslator::visitEnd ( S_harmony& elt )
2  {
3    // ... ... ...
4
5    if (gGlobalMxsr2msrOahGroup->getIgnoreHarmonies ()) {
6  #ifdef TRACING_IS_ENABLED
7      if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
8        gLogStream <<
9          "Ignoring harmony" <<
10         ", line " <<
11         inputLineNumber <<
12         endl;
13     }
14 #endif
15   }
16   else {
17     // create the harmony
18 #ifdef TRACING_IS_ENABLED
19     if (gGlobalTracingOahGroup->getTraceHarmoniesDetails ()) {
20       gLogStream <<
21         "Creating a harmony" <<
22         ", line " << inputLineNumber << ":" <<
23         endl;
24
25       // ... ... ...
26     }
27 #endif
28
29     S_msrHarmony
30       harmony =
31         msrHarmony::createWithoutVoiceUplink (
32           fCurrentHarmonyInputLineNumber,
33           // no harmoniesVoiceUpLink yet
34
35           fCurrentHarmonyRootQuarterTonesPitchKind,
36
37           fCurrentHarmonyKind,
38           fCurrentHarmonyKindText,
39
40           fCurrentHarmonyInversion,
41
42           fCurrentHarmonyBassQuarterTonesPitchKind,
43
44           rational (1, 1),            // harmonySoundingWholeNotes,
45                                       // will be set upon next note handling
46           rational (1, 1),            // harmonyDisplayWholeNotes,
47                                       // will be set upon next note handling
```

```
48            fCurrentHarmoniesStaffNumber ,
49            msrTupletFactor (1, 1),        // will be set upon next note handling
50            fCurrentHarmonyWholeNotesOffset);
51
52      // append pending harmony degrees if any to the harmony
53      if (! fCurrentHarmonyDegreesList.size ()) {
54 #ifdef TRACING_IS_ENABLED
55        if (gGlobalTracingOahGroup ->getTraceHarmoniesDetails ()) {
56          musicxmlWarning (
57            gGlobalServiceRunData ->getInputSourceName (),
58            inputLineNumber ,
59            "harmony has no degrees contents");
60        }
61 #endif
62      }
63
64      else {
65        // handle harmony degrees if any
66        while (fCurrentHarmonyDegreesList.size ()) {
67          S_msrHarmonyDegree
68            harmonyDegree =
69              fCurrentHarmonyDegreesList.front ();
70
71          // ... ... ...
72
73          // append it to harmony's degrees list
74          harmony ->
75            appendHarmonyDegreeToHarmony (
76              harmonyDegree);
77
78          // remove it from the list
79          fCurrentHarmonyDegreesList.pop_front ();
80        } // while
81      }
82
83      // attach the current frame if any to the harmony
84      if (fCurrentFrame) {
85        harmony ->setHarmonyFrame (fCurrentFrame);
86      }
87
88      // append the harmony to the pending harmonies list
89      fPendingHarmoniesList.push_back (harmony);
90    }
91
92   fOnGoingHarmony = false;
93 }
```

62.5.3   Attaching `msrHarmony` instances to notes

msrHarmony

The contents of `mxsr2msrTranslatorfPendingHarmoniesList` is attached to the class `msrNote` instance in method `mxsr2msrTranslator::populateNote ()`:

```
1 void mxsr2msrTranslator::populateNote (
2   int        inputLineNumber ,
3   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 }
```

62.5.4   Populating `msrHarmony` instances

`msrHarmony`

The class `msrHarmony` instances are populated further in `src/formats/msr/mxsr2msrTranslator.cpp` and attached to the note by method `msrNote::appendHarmonyToNoteHarmoniesList ()`:

```
1 void mxsr2msrTranslator :: handlePendingHarmonies (
2   S_msrNote   newNote ,
3   S_msrVoice voiceToInsertInto)
4 {
5   // ... ... ...
6
7   rational
8     newNoteSoundingWholeNotes =
9       newNote ->
10         getNoteSoundingWholeNotes (),
11     newNoteDisplayWholeNotes =
12       newNote ->
13         getNoteDisplayWholeNotes ();
14
15   while (fPendingHarmoniesList.size ()) { // recompute at each iteration
16     S_msrHarmony
17       harmony =
18         fPendingHarmoniesList.front ();
19
20     /*
21        MusicXML harmonies don't have a duration ,
22        and MSR could follow this line, but LilyPond needs one...
23        So:
24          - we register all harmonies with the duration of the next note
25          - they will be sorted by position in the measure in finalizeMeasure(),
26            at which time their duration may be shortened
27            so that the offsets values are enforced
28            and they don't overflow the measure
29        It is VITAL that harmonies measures be finalized
30        AFTER the corresponding measure in the regular voice ,
31        since the current sounding whole notes of the latter is needed for that
32     */
33
34     // set the harmony's sounding whole notes
35     harmony ->
36       setHarmonySoundingWholeNotes (
37         newNoteSoundingWholeNotes);
38
39     // set the harmony's display whole notes JMI useless???
40     harmony ->
41       setHarmonyDisplayWholeNotes (
42         newNoteDisplayWholeNotes);
```

```
43
44      // set the harmony's tuplet factor
45      harmony->
46        setHarmonyTupletFactor (
47          msrTupletFactor (
48            fCurrentNoteActualNotes,
49            fCurrentNoteNormalNotes));
50
51      // attach the harmony to newNote
52      newNote->
53        appendHarmonyToNoteHarmoniesList (
54          harmony);
55
56      // get the part harmonies voice
57      S_msrVoice
58        partHarmoniesVoice =
59          fCurrentPart->
60            getPartHarmoniesVoice ();
61
62      // sanity check
63      mfAssert (
64        __FILE__, __LINE__,
65        partHarmoniesVoice != nullptr,
66        "partHarmoniesVoice is null");
67
68      // set the harmony's voice upLink
69      // only now that we know which harmonies voice will contain it
70      harmony->
71        setHarmoniesVoiceUpLink (
72          partHarmoniesVoice);
73
74 /* JMI CAFE
75      // append the harmony to the part harmonies voice
76      partHarmoniesVoice->
77        appendHarmonyToVoice (
78          harmony);
79 */
80      // don't append the harmony to the part harmonies voice // BLARK
81      // before the note itself has been appended to the voice
82
83      // remove the harmony from the list
84      fPendingHarmoniesList.pop_front ();
85    } // while
86 }
```

62.5.5   First `S_harmony` visit

`msrHarmony`

Method **msrNote::appendHarmonyToNoteHarmoniesList ()** is where the harmony's note uplink is set:

```
1  void msrNote::appendHarmonyToNoteHarmoniesList (S_msrHarmony harmony)
2  {
3  #ifdef TRACING_IS_ENABLED
4    if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
5      gLogStream <<
6        "Appending harmony " <<
7        harmony->asString () <<
8        " to the harmonies list of " <<
9        asString () <<
10       ", line " << fInputLineNumber <<
11       endl;
12   }
13 #endif
```

```
14
15    // update the harmony whole notes if it belongs to a tuplet ??? utf8.xml JMI
16
17    fNoteHarmoniesList.push_back (harmony);
18
19    // register this note as the harmony note upLink
20    harmony->
21      setHarmonyNoteUpLink (this);
22  }
```

When a harmony is attached to a note that is a chord member, we have to attach it to the chord too, to facilitate setting its position in measure when setting the chord's one.

```
1  void mxsr2msrTranslator::copyNoteHarmoniesToChord (
2    S_msrNote note, S_msrChord chord)
3  {
4    // copy note's harmony if any from the first note to chord
5
6    const list<S_msrHarmony>&
7      noteHarmoniesList =
8        note->getNoteHarmoniesList ();
9
10   if (noteHarmoniesList.size ()) {
11     list<S_msrHarmony>::const_iterator i;
12     for (i=noteHarmoniesList.begin (); i!=noteHarmoniesList.end (); ++i) {
13       S_msrHarmony harmony = (*i);
14
15 #ifdef TRACING_IS_ENABLED
16       if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
17         gLogStream <<
18           "Copying harmony '" <<
19           harmony->asString () <<
20           "' from note " << note->asString () <<
21           " to chord '" << chord->asString () <<
22           "'" <<
23           endl;
24       }
25 #endif
26
27       chord->
28         appendHarmonyToChord (harmony);
29
30     } // for
31   }
32 }
```

62.5.6   Inserting `msrHarmony` instances in the part harmonies voice

`msrHarmony`

Inserting the harmonies in the part harmonies voice is done in method `msrVoice::appendNoteToVoice ()` in src/formats/msr/msrNotes.cpp:

```
1  void msrVoice::appendNoteToVoice (S_msrNote note)
2  {
3    // ... ... ...
4
5    // are there harmonies attached to this note? // BLARK
6    const list<S_msrHarmony>&
7      noteHarmoniesList =
8        note->
9          getNoteHarmoniesList ();
10
```

```
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  };
```

62.6   Translating harmonies from MSR to MSR

This is done in src/passes/msr2msr/.

In src/passes/msr2msr/msr2msrTranslator.cpp, a newborn clone of the harmony is created upon the first visit, stored in msr2msrTranslatorfCurrentHarmonyClone, and appended to the current non grace note clone, the current chord clone or to the current voice clone, if the latter is a harmonies voice:

```
1  void msr2msrTranslator::visitStart (S_msrHarmony& elt)
2  {
3  #ifdef TRACING_IS_ENABLED
4    if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5      gLogStream <<
6        "--> Start visiting msrHarmony '" <<
7        elt->asString () <<
8        ", fOnGoingNonGraceNote: " << fOnGoingNonGraceNote <<
9        ", fOnGoingChord: " << fOnGoingChord <<
10        ", fOnGoingHarmoniesVoice: " << fOnGoingHarmoniesVoice <<
11        ", fOnGoingHarmony: " << fOnGoingHarmony <<
12        "', line " << elt->getInputLineNumber () <<
13        endl;
14    }
15  #endif
16
17    // create a harmony new born clone
18    fCurrentHarmonyClone =
19      elt->
20        createHarmonyNewbornClone (
21          fCurrentVoiceClone);
22
23    if (fOnGoingNonGraceNote) {
24      // register the harmony in the current non-grace note clone
25      fCurrentNonGraceNoteClone->
26        appendHarmonyToNoteHarmoniesList (
27          fCurrentHarmonyClone);
28
29      // don't append the harmony to the part harmony,
30      // this has been done in pass2b // JMI ???
31    }
32
33    else if (fOnGoingChord) {
34      // register the harmony in the current chord clone
35      fCurrentChordClone->
36        appendHarmonyToChord (fCurrentHarmonyClone); // JMI
37    }
```

```
38
39    else if (fOnGoingHarmoniesVoice) {
40    /* JMI
41      // get the harmony whole notes offset
42      rational
43        harmonyWholeNotesOffset =
44          elt->getHarmonyWholeNotesOffset ();
45
46      // is harmonyWholeNotesOffset not equal to 0?
47      if (harmonyWholeNotesOffset.getNumerator () != 0) {
48        // create skip with duration harmonyWholeNotesOffset
49        S_msrNote
50          skip =
51            msrNote::createSkipNote (
52              elt->                    getInputLineNumber (),
53              "666", // JMI elt->                 getHarmoniesMeasureNumber (),
54              elt->                    getHarmonyDisplayWholeNotes (), // would be 0/1 otherwise
      JMI
55              elt->                    getHarmonyDisplayWholeNotes (),
56              0, // JMI elt->                  getHarmonyDotsNumber (),
57              fCurrentVoiceClone-> getRegularVoiceStaffSequentialNumber (), // JMI
58              fCurrentVoiceClone-> getVoiceNumber ());
59
60        // append it to the current voice clone
61        // to 'push' the harmony aside
62        fCurrentVoiceClone->
63          appendNoteToVoice (skip);
64      }
65 */
66
67      // append the harmony to the current voice clone
68      fCurrentVoiceClone->
69        appendHarmonyToVoiceClone (
70          fCurrentHarmonyClone);
71    }
72
73    else {
74      stringstream s;
75
76      s <<
77        "harmony is out of context, cannot be handled: '" <<
78        elt->asShortString () <<
79        "'";
80
81      msrInternalError (
82        gGlobalServiceRunData->getInputSourceName (),
83        elt->getInputLineNumber (),
84        __FILE__, __LINE__,
85        s.str ());
86    }
87
88    fOnGoingHarmony = true;
89 }
```

There are only fields updates upon the second visit:

```
1  void msr2msrTranslator::visitEnd (S_msrHarmony& elt)
2  {
3  #ifdef TRACING_IS_ENABLED
4    if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5      gLogStream <<
6        "--> End visiting msrHarmony '" <<
7        elt->asString () <<
8        "'" <<
9        ", line " << elt->getInputLineNumber () <<
10       endl;
```

```
11    }
12 #endif
13
14   fCurrentHarmonyClone = nullptr;
15   fOnGoingHarmony = false;
16 }
```

62.7   Translating harmonies from MSR to LPSR

This is done in `src/passes/msr2lpsr/`.

The same occurs in `src/passes/msr2lpsr/msr2lpsrTranslator.cpp`: a newborn clone of the harmony is created and appended to the current non grace note clone, the current chord clone or to the current voice clone, if the latter is a harmonies voice: :

```
1 void msr2lpsrTranslator::visitStart (S_msrHarmony& elt)
2 {
3 #ifdef TRACING_IS_ENABLED
4   if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5     gLogStream <<
6       "--> Start visiting msrHarmony '" <<
7       elt->asString () <<
8       ", onGoingNonGraceNote: " << fOnGoingNonGraceNote <<
9       ", onGoingChord: " << fOnGoingChord <<
10       ", onGoingHarmoniesVoice: " << fOnGoingHarmoniesVoice <<
11       ", onGoingHarmony: " << fOnGoingHarmony <<
12       "', line " << elt->getInputLineNumber () <<
13       endl;
14   }
15 #endif
16
17   // create a harmony new born clone
18   fCurrentHarmonyClone =
19     elt->
20       createHarmonyNewbornClone (
21         fCurrentVoiceClone);
22
23   if (fOnGoingNonGraceNote) {
24     // register the harmony in the current non-grace note clone
25     fCurrentNonGraceNoteClone->
26       appendHarmonyToNoteHarmoniesList (
27         fCurrentHarmonyClone);
28
29     // don't append the harmony to the part harmony,
30     // this has been done in pass2b // JMI ???
31   }
32
33   else if (fOnGoingChord) {
34     // register the harmony in the current chord clone
35     fCurrentChordClone->
36       appendHarmonyToChord (fCurrentHarmonyClone); // JMI
37   }
38
39   else if (fOnGoingHarmoniesVoice) {
40   /* JMI
41     // get the harmony whole notes offset
42     rational
43       harmonyWholeNotesOffset =
44         elt->getHarmonyWholeNotesOffset ();
45
46     // is harmonyWholeNotesOffset not equal to 0?
47     if (harmonyWholeNotesOffset.getNumerator () != 0) {
48       // create skip with duration harmonyWholeNotesOffset
```

```
49        S_msrNote
50          skip =
51            msrNote::createSkipNote (
52              elt->                    getInputLineNumber (),
53              "666", // JMI elt->                    getHarmoniesMeasureNumber (),
54              elt->                    getHarmonyDisplayWholeNotes (), // would be 0/1 otherwise
      JMI
55              elt->                    getHarmonyDisplayWholeNotes (),
56              0, // JMI elt->                    getHarmonyDotsNumber (),
57              fCurrentVoiceClone-> getRegularVoiceStaffSequentialNumber (), // JMI
58              fCurrentVoiceClone-> getVoiceNumber ());
59
60        // append it to the current voice clone
61        // to 'push' the harmony aside
62        fCurrentVoiceClone->
63          appendNoteToVoice (skip);
64      }
65 */
66
67      // append the harmony to the current voice clone
68      fCurrentVoiceClone->
69        appendHarmonyToVoiceClone (
70          fCurrentHarmonyClone);
71    }
72
73    else {
74      stringstream s;
75
76      s <<
77        "harmony is out of context, cannot be handled: '" <<
78        elt->asShortString () <<
79        "'";
80
81      msrInternalError (
82        gGlobalServiceRunData->getInputSourceName (),
83        elt->getInputLineNumber (),
84        __FILE__, __LINE__,
85        s.str ());
86    }
87
88    fOnGoingHarmony = true;
89 }
```

Here too, there are only fields updates upon the second visit of `S_msrHarmony` instances:

```cpp
void msr2lpsrTranslator::visitEnd (S_msrHarmony& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> End visiting msrHarmony '" <<
      elt->asString () <<
      "'" <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  fCurrentHarmonyClone = nullptr;
  fOnGoingHarmony = false;
}
```

62.8   Translating harmonies from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

There is only one visit of class `msrHarmony` instances in `src/passes/lpsr2lilypond/lpsr2lilypondTranslator.cpp`. The LilyPond code is generated only if the harmony belongs to a voice: this is where denormalization ends in the workflow:

```cpp
void lpsr2lilypondTranslator::visitStart (S_msrHarmony& elt)
{
#ifdef TRACING_IS_ENABLED
  {
    Bool
      traceMsrVisitors =
        gGlobalMsrOahGroup->
          getTraceMsrVisitors (),
      generateMsrVisitingInformation =
        gGlobalLpsr2lilypondOahGroup->
          getGenerateMsrVisitingInformation ();

    if (traceMsrVisitors || generateMsrVisitingInformation) {
      stringstream s;

      s <<
        "% --> Start visiting msrHarmony '" <<
        elt->asString () <<
        "'" <<
        ", fOnGoingNotesStack.size () = " <<
        fOnGoingNotesStack.size () <<
        ", fOnGoingChord = " <<
        fOnGoingChord <<
        ", fOnGoingHarmoniesVoice = " <<
        fOnGoingHarmoniesVoice <<
        ", line " << elt->getInputLineNumber () <<
        endl;

      if (traceMsrVisitors) {
        gLogStream << s.str ();
      }

      if (generateMsrVisitingInformation) {
        fLilypondCodeStream << s.str ();
      }
    }
```

```
37    }
38 #endif
39
40    if (fOnGoingNotesStack.size () > 0) {
41    /* JMI
42 #ifdef TRACING_IS_ENABLED
43      if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
44        fLilypondCodeStream <<
45          "%{ fOnGoingNotesStack.size () S_msrHarmony JMI " <<
46          elt->asString () <<
47          " %}" <<
48          endl;
49      }
50 #endif
51 */
52    }
53
54    else if (fOnGoingChord) { // JMI
55    }
56
57    else if (fOnGoingHarmoniesVoice) {
58      // actual LilyPond code generation
59      fLilypondCodeStream <<
60        harmonyAsLilypondString (elt) <<
61        ' ';
62
63      // generate the input line number as comment if relevant
64      if (
65        gGlobalLpsr2lilypondOahGroup->getInputLineNumbers ()
66          ||
67        gGlobalLpsr2lilypondOahGroup->getGeneratePositionsInMeasures ()
68      ) {
69        generateInputLineNumberAndOrPositionInMeasureAsAComment (
70          elt);
71      }
72    }
73 }
```

Chapter 63

Figured bass elements handling

Figured bass elements are presented at section 19.50, [Figured bass elements], page 184.

The useful options here are:

- option **-trace-figured-bass, -tfigbass**

- option **-display-msr-skeleton, -dmsrskel**

- option **-display-msr-1, -dmsr1**

- option **-display-msr-1-short, -dmsr1s** and option **-display-msr-1-details, -dmsr1d**

- option **-display-msr-2, -dmsr2**

- option **-display-msr-2-short, -dmsr2s** and option **-display-msr-2-details, -dmsr2d**

Figured bass elements need special treatment since we need to determine their position in a figured bass voice. This is different than MusicXML, where they are simply *drawn* at the current music position, so to say.

They are handled this way:

- figured bass elements are stored in class `msrNote`:

- they are also stored in class `msrPart` and class `msrChord` and class `msrTuplet` (denormalization);

In class `msrNote`, there is:

```
// figured bass
void                    appendFiguredBassToNoteFiguredBassElementsList (
                          S_msrFiguredBassElement figuredBass);

const list<S_msrFiguredBassElement>&
                        getNoteFiguredBassElementsList () const
                            { return fNoteFiguredBassElementsList; }

// ... ... ...

// figured bass
// ------------------------------------------------------

list<S_msrFiguredBassElement>
                        fNoteFiguredBassElementsList;
```

63.1   Figured bass staves and voices

Every class `msrVoice` instance in MusicFormats belongs to an class `msrStaff` instance. Staves are created specifically to hold figured bass voices, using specific numbers defined in `src/formats/msr/msrParts.h`:

```
public:

  // constants
  // --------------------------------------------------------

  // ... ... ...

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

In class `msrStaff`, there is:

```
    void                     registerFiguredBassVoiceByItsNumber (
                               int         inputLineNumber,
                               S_msrVoice voice);
```

Class `msrPart` also contains:

```
    // figured bass

    S_msrVoice               createPartFiguredBassVoice (
                               int    inputLineNumber,
                               string currentMeasureNumber);

    void                     appendFiguredBassToPart (
                               S_msrVoice               figuredBassSupplierVoice,
                               S_msrFiguredBassElement figuredBass);

    void                     appendFiguredBassToPartClone (
                               S_msrVoice               figuredBassSupplierVoice,
                               S_msrFiguredBassElement figuredBass);
```

```
    // figured bass

    S_msrStaff               fPartFiguredBassStaff;
    S_msrVoice               fPartFiguredBassVoice;
```

63.2   Figured bass staves creation

This is done in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.cpp.h/.cpp`:

```
S_msrVoice mxsr2msrSkeletonBuilder::createPartFiguredBassVoiceIfNotYetDone (
  int        inputLineNumber,
  S_msrPart  part)
{
  // is the figured bass voice already present in part?
  S_msrVoice
    partFiguredBassVoice =
      part->
        getPartFiguredBassVoice ();

  if (! partFiguredBassVoice) {
    // create the figured bass voice and append it to the part
    partFiguredBassVoice =
      part->
        createPartFiguredBassVoice (
```

```
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    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 }
```

63.3   Figured bass elements creation

There several methods for Figured bass elements creation:

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/src > grep create formats/msr/
     msrFiguredBassElements.h
2      static SMARTP<msrBassFigure> create (
3      SMARTP<msrBassFigure> createFigureNewbornClone (
4  //     SMARTP<msrBassFigure> createFigureDeepClone ( // JMI ???
5      static SMARTP<msrFiguredBassElement> create (
```

```
6      static SMARTP < msrFiguredBassElement > create (
7      SMARTP < msrFiguredBassElement > createFiguredBassElementNewbornClone (
8 //      SMARTP < msrFiguredBassElement > createFiguredBassElementDeepClone ();
```

63.4 Translating figured bass elements from MXSR to MSR

This is done in `src/passes/mxsr2msr/`, and this is where the class `msrFiguredBassElement` instances are created.

The MSR score skeleton created in `src/passes/mxsr2msr/mxsr2msrSkeletonBuilder.h/.cpp` contains the part groups, parts, staves and voices, as well as the number of measures. The voices do not contain any music elements yet.

A figured bass element belongs to `<part/>` in MusicXML, but we sometimes need to have it attached to a note.
Field`mxsr2msrSkeletonBuilderfThereAreFiguredBassToBeAttachedToCurrentNote` it used when visiting an `S_FiguredBassElement` element to account for that:

```
1  void mxsr2msrSkeletonBuilder::visitStart ( S_figured_bass& elt )
2  {
3  #ifdef TRACING_IS_ENABLED
4    if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
5      gLogStream <<
6        "--> Start visiting S_figured_bass" <<
7        ", figuredBassVoicesCounter = " << fFiguredBassVoicesCounter <<
8        ", line " << elt->getInputLineNumber () <<
9        endl;
10   }
11 #endif
12
13   /* JMI
14     several figured bass elements can be attached to a given note,
15     leading to as many figured bass voices in the current part JMI TRUE???
16   */
17
18   // take figured bass voice into account
19   ++fFiguredBassVoicesCounter;
20
21   fThereAreFiguredBassToBeAttachedToCurrentNote = true;
22 }
```

Upon the second visit of class `msrNote`, the part figured bass voice is created if figured bass elements are not to be ignored due to option option `-ignore-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->getIgnoreFiguredBassElements ()) {
8  #ifdef TRACING_IS_ENABLED
9        if (gGlobalTracingOahGroup->getTraceFiguredBass ()) {
10         gLogStream <<
11           "Ignoring the figured bass elements" <<
12           ", line " <<
13           inputLineNumber <<
14           endl;
15       }
16 #endif
17     }
```

```
18      else {
19        // create the part figured bass voice if not yet done
20        S_msrVoice
21          partFiguredBassVoice =
22            createPartFiguredBassVoiceIfNotYetDone (
23              inputLineNumber,
24              fCurrentPart);
25      }
26
27      fThereAreFiguredBassToBeAttachedToCurrentNote = false;
28
29    // ... ... ...
30 }
```

Creating the part figured bass voice is delegated to the part:

```
1 S_msrVoice mxsr2msrSkeletonBuilder::createPartFiguredBassVoiceIfNotYetDone (
2   int        inputLineNumber,
3   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 }
```

63.4.1   First `S_figured_bass` visit

The first visit of `S_figured_bass` initializes the fields storing values to be gathered visiting subelements:

```
1 void mxsr2msrTranslator::visitStart ( S_figured_bass& elt )
2 {
3   int inputLineNumber =
4     elt->getInputLineNumber ();
5
6 #ifdef TRACING_IS_ENABLED
7   if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
8     gLogStream <<
9       "--> Start visiting S_figured_bass" <<
10      ", line " << inputLineNumber <<
11      endl;
12  }
13 #endif
14
15   ++fFiguredBassVoicesCounter;
16
17   string parentheses = elt->getAttributeValue ("parentheses");
18
19   fCurrentFiguredBassParenthesesKind =
20     msrFiguredBassElement::kFiguredBassElementParenthesesNo; // default value
21
22   if (parentheses.size ()) {
```

```
23      if (parentheses == "yes")
24        fCurrentFiguredBassParenthesesKind =
25          msrFiguredBassElement::kFiguredBassElementParenthesesYes;
26
27      else if (parentheses == "no")
28       fCurrentFiguredBassParenthesesKind =
29          msrFiguredBassElement::kFiguredBassElementParenthesesNo;
30
31      else {
32        stringstream s;
33
34        s <<
35          "parentheses value " << parentheses <<
36          " should be 'yes' or 'no'";
37
38        musicxmlError (
39          gGlobalServiceRunData->getInputSourceName (),
40          inputLineNumber,
41          __FILE__, __LINE__,
42          s.str ());
43      }
44    }
45
46    fCurrentFiguredBassInputLineNumber   = -1;
47
48    fCurrentFigureNumber = -1;
49
50    fCurrentFigurePrefixKind = msrBassFigure::k_NoFigurePrefix;
51    fCurrentFigureSuffixKind = msrBassFigure::k_NoFigureSuffix;
52
53    fCurrentFiguredBassSoundingWholeNotes = rational (0, 1);
54    fCurrentFiguredBassDisplayWholeNotes  = rational (0, 1);
55
56    fOnGoingFiguredBass = true;
57 }
```

63.4.2 Second `S_figured_bass` visit

Upon the second visit of `S_figured_bass`, the class `msrFiguredBassElement` instance is created, populated and appended to `mxsr2msrTranslatorfPendingFiguredBassElementsList`:

```
1 void mxsr2msrTranslator::visitEnd ( S_figured_bass& elt )
2 {
3   int inputLineNumber =
4     elt->getInputLineNumber ();
5
6 #ifdef TRACING_IS_ENABLED
7   if (gGlobalMxsrOahGroup->getTraceMxsrVisitors ()) {
8     gLogStream <<
9       "--> End visiting S_figured_bass" <<
10      ", line " << inputLineNumber <<
11      endl;
12  }
13 #endif
14
15   // create the figured bass element
16 #ifdef TRACING_IS_ENABLED
17   if (gGlobalTracingOahGroup->getTraceFiguredBass ()) {
18     gLogStream <<
19       "Creating a figured bass" <<
20       ", line " << inputLineNumber << ":" <<
21       endl;
22   }
23 #endif
```

```
24
25    // create the figured bass element
26    // if the sounding whole notes is 0/1 (no <duration /> was found), JMI ???
27    // it will be set to the next note's sounding whole notes later
28    S_msrFiguredBassElement
29      figuredBassElement =
30        msrFiguredBassElement::create (
31          inputLineNumber,
32    // JMI        fCurrentPart,
33          fCurrentFiguredBassSoundingWholeNotes,
34          fCurrentFiguredBassDisplayWholeNotes,
35          fCurrentFiguredBassParenthesesKind,
36          msrTupletFactor (1, 1));    // will be set upon next note handling
37
38    // attach pending figures to the figured bass element
39    if (! fPendingFiguredBassFiguresList.size ()) {
40      musicxmlWarning (
41        gGlobalServiceRunData->getInputSourceName (),
42        inputLineNumber,
43        "figured-bass has no figures contents, ignoring it");
44    }
45    else {
46      // append the pending figures to the figured bass element
47      for (S_msrBassFigure bassFigure : fPendingFiguredBassFiguresList) {
48        figuredBassElement->
49          appendFigureToFiguredBass (bassFigure);
50      } // for
51
52      // forget about those pending figures
53      fPendingFiguredBassFiguresList.clear ();
54
55      // append the figured bass element to the pending figured bass elements list
56      fPendingFiguredBassElementsList.push_back (figuredBassElement);
57    }
58
59    fOnGoingFiguredBass = false;
60  }
```

63.4.3 Attaching `msrFiguredBassElement` instances to notes

`msrFiguredBassElement`

The contents of `mxsr2msrTranslatorfPendingFiguredBassElementsList` is attached to the class `msrNote` instance in method
method `mxsr2msrTranslator::populateNote ()`:

```
1  void mxsr2msrTranslator::populateNote (
2    int        inputLineNumber,
3    S_msrNote newNote)
4  {
5    // ... ... ...
6
7    // handle the pending figured bass elements if any
8    if (fPendingFiguredBassElementsList.size ()) {
9      // get voice to insert figured bass elements into
10     S_msrVoice
11       voiceToInsertFiguredBassElementsInto =
12         fCurrentPart->
13           getPartFiguredBassVoice ();
14
15     // ... ... ...
16
17     handlePendingFiguredBassElements (
18       newNote,
```

```
19        voiceToInsertFiguredBassElementsInto);
20
21    // reset figured bass counter
22    fFiguredBassVoicesCounter = 0;
23  }
24 }
```

63.4.4 Populating `msrFiguredBassElement` instances

`msrFiguredBassElement`

In `src/formats/msr/mxsr2msrTranslator.cpp`, the class `msrFiguredBassElement` instances are populated further and attached to the note by method `mxsr2msrTranslator::handlePendingFiguredBassElements ():`

```
1 void mxsr2msrTranslator::handlePendingFiguredBassElements (
2   S_msrNote   newNote,
3   S_msrVoice voiceToInsertInto)
4 {
5   // ... ... ...
6
7   rational
8     newNoteSoundingWholeNotes =
9       newNote->
10        getNoteSoundingWholeNotes (),
11    newNoteDisplayWholeNotes =
12      newNote->
13        getNoteDisplayWholeNotes ();
14
15  while (fPendingFiguredBassElementsList.size ()) { // recompute at each iteration
16    S_msrFiguredBassElement
17      figuredBassElement =
18        fPendingFiguredBassElementsList.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    figuredBassElement->
29      setMeasureElementSoundingWholeNotes (
30        newNoteSoundingWholeNotes,
31        "handlePendingFiguredBassElements()");
32
33    // set the figured bass element's display whole notes JMI useless???
34    figuredBassElement->
35      setFiguredBassDisplayWholeNotes (
36        newNoteDisplayWholeNotes);
37
38    // set the figured bass element's tuplet factor
39    figuredBassElement->
40      setFiguredBassTupletFactor (
41        msrTupletFactor (
42          fCurrentNoteActualNotes,
43          fCurrentNoteNormalNotes));
44
45    // append the figured bass to newNote
46    newNote->
47      appendFiguredBassElementToNoteFiguredBassElementsList (
48        figuredBassElement);
49
50 /* JMI
```

```
51      // get the figured bass voice for the current voice
52      S_msrVoice
53        voiceFiguredBassVoice =
54          voiceToInsertInto->
55            getRegularVoiceFiguredBassVoiceForwardLink ();
56
57      // sanity check
58      mfAssert (
59        __FILE__, __LINE__,
60        voiceFiguredBassVoice != nullptr,
61        "voiceFiguredBassVoice is null");
62
63      // set the figuredBassElement's voice upLink
64      // only now that we know which figured bass voice will contain it
65      figuredBassElement->
66        setFiguredBassVoiceUpLink (
67          voiceFiguredBassVoice);
68
69      // append the figured bass to the figured bass voice for the current voice
70      voiceFiguredBassVoice->
71        appendFiguredBassElementToVoice (
72          figuredBassElement);
73 */
74
75      // don't append the figured bass to the part figured bass voice
76      // before the note itself has been appended to the voice
77
78      // remove the figured bass from the list
79      fPendingFiguredBassElementsList.pop_front ();
80    } // while
81 }
```

```
1 %void mxsr2msrTranslator::copyNoteHarmoniesToChord (
2 %   S_msrNote note, S_msrChord chord)
3 %{
4 %   // copy note's harmony if any from the first note to chord
5 %
6 %   const list<S_msrHarmony>&
7 %     noteHarmoniesList =
8 %       note->getNoteHarmoniesList ();
9 %
10 %   if (noteHarmoniesList.size ()) {
11 %     list<S_msrHarmony>::const_iterator i;
12 %     for (i=noteHarmoniesList.begin (); i!=noteHarmoniesList.end (); ++i) {
13 %       S_msrHarmony harmony = (*i);
14 %
15 %#ifdef TRACING_IS_ENABLED
16 %       if (gGlobalTracingOahGroup->getTraceHarmonies ()) {
17 %         gLogStream <<
18 %           "Copying harmony '" <<
19 %           harmony->asString () <<
20 %           "' from note " << note->asString () <<
21 %           " to chord '" << chord->asString () <<
22 %           "'" <<
23 %           endl;
24 %       }
25 %#endif
26 %
27 %       chord->
28 %         appendHarmonyToChord (harmony);
29 %
30 %     } // for
31 %   }
32 %}
33 %
```

63.4.5 Inserting `S_msrFiguredBassElement` instances in the part figured bass voice

Method `msrVoice::appendNoteToVoice ()` in `src/formats/msr/msrNotes.cpp` inserts the figured bass elements in the part figured bass voice:

```
void msrVoice::appendNoteToVoice (S_msrNote note)
{
  // ... ... ...

  // are there figured bass elements attached to this note?
  const list<S_msrFiguredBassElement>&
    noteFiguredBassElementsList =
      note->
        getNoteFiguredBassElementsList ();

  if (noteFiguredBassElementsList.size ()) {
    // get the current part's figured bass voice
    S_msrVoice
      partFiguredBassVoice =
        part->
          getPartFiguredBassVoice ();

    for (S_msrFiguredBassElement figuredBassElement : noteFiguredBassElementsList) {
      // append the figured bass element to the part figured bass voice
      partFiguredBassVoice->
        appendFiguredBassElementToVoice (
          figuredBassElement);
    } // for
  }
};
```

63.5 Translating figured bass elements from MSR to MSR

This is done in `src/passes/msr2msr/`.

In `src/passes/msr2msr/msr2msrTranslator.cpp`, a newborn clone of the figured bass element is created upon the first visit, stored in `msr2msrTranslatorfCurrentFiguredBassElementClone`, and appended to the current non grace note clone, the current chord clone or to the current voice clone, if the latter is a figured bass voice:

```
void msr2msrTranslator::visitStart (S_msrFiguredBassElement& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrFiguredBassElement '" <<
      elt->asString () <<
      "'" <<
      ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  // create a figured bass element new born clone
  fCurrentFiguredBassElementClone =
    elt->
      createFiguredBassElementNewbornClone (
        fCurrentVoiceClone);

  if (fOnGoingNonGraceNote) {
    // append the figured bass to the current non-grace note clone
```

```
23      fCurrentNonGraceNoteClone ->
24        appendFiguredBassElementToNoteFiguredBassElementsList (
25          fCurrentFiguredBassElementClone);
26
27      // don't append the figured bass to the part figured bass,  JMI ???
28      // this will be done below
29    }
30
31    /* JMI
32    else if (fOnGoingChord) {
33      // register the figured bass in the current chord clone
34      fCurrentChordClone ->
35        setChordFiguredBass (fCurrentFiguredBassElementClone); // JMI
36    }
37    */
38
39    else if (fOnGoingFiguredBassVoice) { // JMI
40      /*
41      // register the figured bass in the part clone figured bass
42      fCurrentPartClone ->
43        appendFiguredBassElementToPartClone (
44          fCurrentVoiceClone ,
45          fCurrentFiguredBassElementClone);
46          */
47      // append the figured bass to the current voice clone
48      fCurrentVoiceClone ->
49        appendFiguredBassElementToVoiceClone (
50          fCurrentFiguredBassElementClone);
51    }
52
53    else {
54      stringstream s;
55
56      s <<
57        "figured bass is out of context, cannot be handled:'" <<
58        elt->asShortString () <<
59        "'";
60
61      msrInternalError (
62        gGlobalServiceRunData ->getInputSourceName (),
63        elt->getInputLineNumber (),
64        __FILE__, __LINE__,
65        s.str ());
66    }
67 }
```

There are only fields updates upon the second visit:

```
1  void msr2msrTranslator::visitEnd (S_msrFiguredBassElement& elt)
2  {
3  #ifdef TRACING_IS_ENABLED
4    if (gGlobalMsrOahGroup ->getTraceMsrVisitors ()) {
5      gLogStream <<
6        "--> End visiting msrFiguredBassElement '" <<
7        elt->asString () <<
8        "'" <<
9        ", line " << elt->getInputLineNumber () <<
10       endl;
11   }
12 #endif
13
14   fCurrentFiguredBassElementClone = nullptr;
15 }
```

63.6 Translating figured bass elements from MSR to LPSR

This is done in `src/passes/msr2lpsr/`.

The same occurs in `src/passes/msr2lpsr/msr2lpsrTranslator.cpp`: a newborn clone of the figured bass element is created and appended to the current non grace note clone, the current chord clone or to the current voice clone, if the latter is a figured bass voice:

```
void msr2lpsrTranslator::visitStart (S_msrFiguredBassElement& elt)
{
#ifdef TRACING_IS_ENABLED
  if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
    gLogStream <<
      "--> Start visiting msrFiguredBassElement '" <<
      elt->asString () <<
      "'" <<
      ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
      ", line " << elt->getInputLineNumber () <<
      endl;
  }
#endif

  // create a figured bass new born clone
  fCurrentFiguredBassElementClone =
    elt->
      createFiguredBassElementNewbornClone (
        fCurrentVoiceClone);

  if (fOnGoingNonGraceNote) {
    // append the figured bass to the current non-grace note clone
    fCurrentNonGraceNoteClone->
      appendFiguredBassElementToNoteFiguredBassElementsList (
        fCurrentFiguredBassElementClone);

    // don't append the figured bass to the part figured bass,  JMI ???
    // this will be done below
  }

  /* JMI
  else if (fOnGoingChord) {
    // register the figured bass in the current chord clone
    fCurrentChordClone->
      setChordFiguredBass (fCurrentFiguredBassElementClone); // JMI
  }
  */

  else if (fOnGoingFiguredBassVoice) { // JMI
    /*
    // register the figured bass in the part clone figured bass
    fCurrentPartClone->
      appendFiguredBassElementToPartClone (
        fCurrentVoiceClone,
        fCurrentFiguredBassElementClone);
        */
    // append the figured bass to the current voice clone
    fCurrentVoiceClone->
      appendFiguredBassElementToVoiceClone (
        fCurrentFiguredBassElementClone);
  }

  else {
    stringstream s;

    s <<
      "figured bass is out of context, cannot be handled:'" <<
      elt->asShortString () <<
```

```
59        "’";
60
61      msrInternalError (
62        gGlobalServiceRunData->getInputSourceName (),
63        elt->getInputLineNumber (),
64        __FILE__, __LINE__,
65        s.str ());
66    }
67  }
```

Here too, there are only fields updates upon the second visit of `S_msrFiguredBassElement` instances:

```
1  void msr2lpsrTranslator::visitEnd (S_msrFiguredBassElement& elt)
2  {
3  #ifdef TRACING_IS_ENABLED
4    if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5      gLogStream <<
6        "--> End visiting msrFiguredBassElement ’" <<
7        elt->asString () <<
8        "’" <<
9        ", line " << elt->getInputLineNumber () <<
10        endl;
11    }
12  #endif
13
14    fCurrentFiguredBassElementClone = nullptr;
15  }
```

63.7   Translating figured bass elements from LPSR to LilyPond

This is done in `src/passes/lpsr2lilypond/`.

There is only one visit of class `msrFiguredBassElement` instances in
`src/passes/lpsr2lilypond/lpsr2lilypondTranslator.cpp`.

The LilyPond code is generated only if the figured bass element belongs to a figured bass voice: this is where denormalization ends in the workflow:

```
1  void msr2lpsrTranslator::visitStart (S_msrFiguredBassElement& elt)
2  {
3  #ifdef TRACING_IS_ENABLED
4    if (gGlobalMsrOahGroup->getTraceMsrVisitors ()) {
5      gLogStream <<
6        "--> Start visiting msrFiguredBassElement ’" <<
7        elt->asString () <<
8        "’" <<
9        ", fOnGoingFiguredBassVoice = " << fOnGoingFiguredBassVoice <<
10        ", line " << elt->getInputLineNumber () <<
11        endl;
12    }
13  #endif
14
15    // create a figured bass new born clone
16    fCurrentFiguredBassElementClone =
17      elt->
18        createFiguredBassElementNewbornClone (
19          fCurrentVoiceClone);
20
21    if (fOnGoingNonGraceNote) {
22      // append the figured bass to the current non-grace note clone
23      fCurrentNonGraceNoteClone->
24        appendFiguredBassElementToNoteFiguredBassElementsList (
25          fCurrentFiguredBassElementClone);
```

```cpp
26
27      // don't append the figured bass to the part figured bass,  JMI ???
28      // this will be done below
29    }
30
31    /* JMI
32    else if (fOnGoingChord) {
33      // register the figured bass in the current chord clone
34      fCurrentChordClone ->
35        setChordFiguredBass (fCurrentFiguredBassElementClone); // JMI
36    }
37    */
38
39    else if (fOnGoingFiguredBassVoice) { // JMI
40      /*
41      // register the figured bass in the part clone figured bass
42      fCurrentPartClone ->
43        appendFiguredBassElementToPartClone (
44          fCurrentVoiceClone ,
45          fCurrentFiguredBassElementClone);
46          */
47      // append the figured bass to the current voice clone
48      fCurrentVoiceClone ->
49        appendFiguredBassElementToVoiceClone (
50          fCurrentFiguredBassElementClone);
51    }
52
53    else {
54      stringstream s;
55
56      s <<
57        "figured bass is out of context, cannot be handled:'" <<
58        elt->asShortString () <<
59        "'";
60
61      msrInternalError (
62        gGlobalServiceRunData ->getInputSourceName (),
63        elt->getInputLineNumber (),
64        __FILE__, __LINE__,
65        s.str ());
66    }
67 }
```

Chapter 64

Lyrics handling

Lyrics are presented at section 19.51, [Lyrics], page 184.

64.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

64.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

64.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

64.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

64.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 65

Full measure rests handling

Full measure rests are presented at section 19.34, [Full measure rests], page 177.

65.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

65.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

65.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

65.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

65.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Chapter 66

MIDI handling

MIDI is presented at section 19.52, [MIDI], page 185.

At the day of this writing, MIDI handling is partial, i.e. not all MIDI elements present in MusicXML are incorporated in MSR and no MIDI data can generated generated by MusicFormats.

66.1 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

66.2 Translating from MXSR to MSR (`src/passes/mxsr2msr/`)

66.3 Translating from MSR to MSR (`src/passes/msr2msr/`)

66.4 Translating from MSR to LPSR (`src/passes/msr2lpsr/`)

66.5 Translating from LPSR to LilyPond (`src/passes/lpsr2lilypond/`)

Part XII

Music Scores Description Language (MSDL)

Chapter 67

MSDL

MSDL is an attempt at a description of music score in a non-linear way, much like a painter puts touches of paint on his work. This is also what users do with GUI music scoring applications, but scores textual descriptions such as LilyPond and Guido impose a linear, left to right, writing of the scores contents.

Contrary to LilyPond, the | token in MSDL is not the end of a measure. Writing |2 means that the music that follows will be placed in a new layer in measure 2.

67.1   Main features of MSDL

They are:

- note are written much like in LilyPond such as `b2...`;

- the keywords such as `pitches` and `music`, are reserved;

- they are available in a number of languages such as english, french, german and italian. It is easy to add other languages;

A first, limited converter is provided by MusicFormats with service `msdl`. It also performs reserved keywords translation from one language to another:

67.2   MSDL basic types

Some types used thoughout MSR are defined in `src/formats/msdl/msdlBasicTypes.h/.cpp`:

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/src/formats/msdl >   egrep -rIn  'ˆ// '
       msdlBasicTypes.h
2  msdlBasicTypes.h:28:// user languages
3  msdlBasicTypes.h:52:// comments types
4  msdlBasicTypes.h:74:// initialization
```

67.3 What the MSDL converter does

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

67.3.1 LilyPond generation

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl > msdl -lilypond -about
2  What msdlConverter does:
3
4      This multi-pass converter basically performs 3 passes when generating LilyPond output
       output:
5
6          Pass 1:  converts the MSDL input into a first MSR
7          Pass 2:  converts the first MSR into a second MSR;
8          Pass 3:  converts the second MSR into a
9                   LilyPond Score Representation (LPSR);
10         Pass 4:  converts the LPSR to LilyPond 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.
```

67.3.2 Braille generation

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl > msdl -braille -about
2  What msdlConverter does:
3
4      This multi-pass converter basically performs 4 passes when generating braille output
       output:
5
6          Pass 1:  converts the MSDL input into a first MSR
7          Pass 2:  converts the first MSR into a second MSR;
8          Pass 3a: converts the second MSR into a
9                   Braille Score Representation (BSR)
10                  containing one Braille page per MusicXML page;
11         Pass 3b: converts the BSRinto another BSR
12                  with as many Braille pages as needed
13                  to fit the line and page lengthes;
14         Pass 4:  converts the BSR to Braille text
15                  and writes it to standard output.)
16
17     In this preliminary version, pass 2b merely clones the BSR it receives.
18
19     Other passes are performed according to the options, such as
20     displaying views of the internal data or printing a summary of the score.
21
22     The activity log and warning/error messages go to standard error.
```

67.3.3 MusicXML generation

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl > msdl -musicxml -about
2  What msdlConverter does:
3
4      This multi-pass converter basically performs 4 passes when generating MusicXML output
       output:
5
6          Pass 1:  converts the MSDL input into a first MSR
7          Pass 2:  converts the first MSR into a second MSR;
8          Pass 3:  converts the second MSR into an MusicXML tree;
9          Pass 4:  converts the MusicXML tree to MusicXML code
10                  and writes it to standard output.
11
12     Other passes are performed according to the options, such as
13     displaying views of the internal data or printing a summary of the score.
14
15     The activity log and warning/error messages go to standard error.
```

67.3.4 Guido generation

```
1  jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl > msdl -guido -about
2  What msdlConverter does:
3
4      This multi-pass converter basically performs 4 passes when generating Guido output
       output:
5
6          Pass 1:  converts the MSDL input into a first MSR
7          Pass 2:  converts the first MSR into a second MSR;
8          Pass 3:  converts the second MSR into an MusicXML tree;
9          Pass 4:  converts the MusicXML tree to Guido code
10                  and writes it to standard output.
11
12     Other passes are performed according to the options, such as
13     displaying views of the internal data or printing a summary of the score.
14
15     The activity log and warning/error messages go to standard error.
16 jacquesmenu@macmini: ~/musicformats-git-dev/files/msdl >
```

67.4 A first example

`HelloWorld.msdl` 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
```

67.5 First example output from the MSDL converter

Compiling `HelloWorld.msdl` to LilyPond, we get the output below.

67.5.1 LilyPond output

```
1   \version "2.22.0"
2
3   % Comment or adapt next line as needed (default is 20)
4   #(set-global-staff-size 20 )
5
6   % Pick your choice from the next two lines as needed
7   %myBreak = { \break }
8   myBreak = {}
9
10  % Pick your choice from the next two lines as needed
11  %myPageBreak = { \pageBreak }
12  myPageBreak = {}
13
14  \header {
15    title                 = ""
16    encodingDate          = "Sunday 2021-05-30 @ 12:11:50 CEST"
17    software              = "MSDL converter 1.0"
18  }
19
20  \paper {
21  }
22
23  \layout {
24    \context {
25      \Score
26      autoBeaming = ##f % to display tuplets brackets
27    }
28    \context {
29      \Voice
30    }
31  }
32
33  Part_Part_One_Staff_One_Voice_One = \absolute {
34    \language "nederlands"
35  c2.. d''8 }
36
37  \book {
38    \score {
39      <<
40
41        \new Staff = "Part_Part_One_Staff_One"
42        \with {
43        }
44        <<
```

```
45        \context Voice = "Part_Part_One_Staff_One_Voice_One" <<
46          \Part_Part_One_Staff_One_Voice_One
47        >>
48      >>
49
50    >>
51
52    \layout {
53      \context {
54        \Score
55        autoBeaming = ##f % to display tuplets brackets
56      }
57      \context {
58        \Voice
59      }
60    }
61
62    \midi {
63      \tempo 4 = 90
64    }
65  }
66
67 }
```

67.5.2 Braille output

With:

```
1 msdl -braille HelloWorld.msdl -use-encoding-in-file-name -braille-output-kind utf8d
```

we get in file `HelloWorld.msdl_UTF8Debug.brf` Braille 6-dots cells, which can be displayed in a suitable editor as:
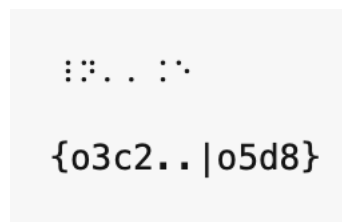
Figure 67.1: Braille for HelloWorld.xml with interpretation

The interpretation shows a textual view of the contents of the previous line. `o*` indicates the octave number.

67.5.3 MusicXML output

Compiling `HelloWorld.msdl` to MusicXML, we get:

```
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.
   msdl ===== -->
6    <work>
7        <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-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-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>
```

67.5.4   Guido output

Compiling `HelloWorld.msdl` 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   }
```

67.6 A more realistic example

Thanks to Jean Abou-Samra for providing `UnPetitAir.msdl`:

```
%{
  An explicit and implicit voices piano score
%}


% l'identification
% ------------------------------------------------

titre       "Un petit air"
compositeur "Jean Abou Samra"


% la langue pour les hauteurs de notes
% ------------------------------------------------

hauteurs francais         % par défaut: english


% la partition
% ------------------------------------------------

musique unPetitAir =
{
  |1  clef treble
      key c
      time 9/8
      r4. a,4-> <e g bf>8~ <e g bf>4.~

  |2  <e g bf>4. r2.

  %  Maintenant, je reviens en arrière pour la voix supérieure.
  |2  fs''16 gs'' fs''8 cs'' ds'' e'' b' d'' a' e'

  %  La voix inférieure s'éteint.
  |3  c''8 gs' d' c' fs' a' b' gs' b
  |4  a'8 e' a g as gs' d'( a ds)
  |5  e8( b g d' a' e'' b'' c''' b''
  |6  e'''4.) % Rien à la fin.

  %  Je décide d'ajouter une tenue de la basse.
  |5  e2.~ e4.

  %  J'ajoute encore une voix. Au passage, je change la métrique.
  |6  time 6/8
  |6  r8 e'( f') e' c'' d''

  %  Et encore un changement de métrique.
  |7  time 4/4
  |7  e''1~

  %  Je finis la phrase.
  |7  e''4 e' d''8 c'' b' a'
  |8  b'1

  %  Je retourne sur mes pas pour introduire l'ostinato.
  |7  r8 e8 f e f e c' a
  |8  r8 e8 f e c' a e f
  |9  r8 ds e ds e ds b fs

  % etc.
}
```

Jean also provided the output created by hand with LilyPond, see figure 67.2, [Un Petit Air, par Jean Abou-Samra], page 340:
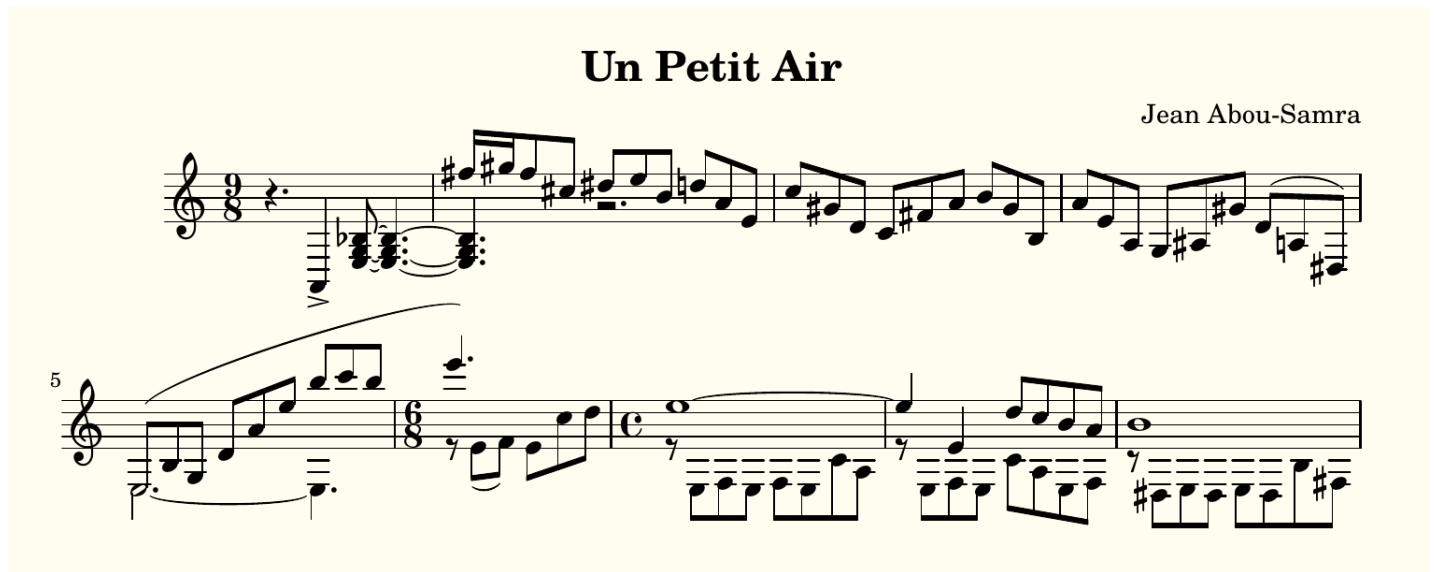
Figure 67.2: Un Petit Air, par Jean Abou-Samra

67.7 Multi-language support

67.7.1 Multi-language messages handling

67.7.2 Multi-language keywords handling

67.8 Lexical analysis

67.9 Music Scores Descriptions Representation (MSDR)

67.10 Syntax and semantic analysis

The language-dependent keywords leads to a recursive descent parser, since `flex`-generated scanners need 'fixed' keyword in the language description.

67.10.1 Error recovery

The MSDL converter uses a variant of the *stopper sets* method that was present in the early Pascal and Pascal-S converters. The latter passed a set of tokens not to be overtaken to the procedures in charge of accepting the various statements in the language. Strangely enough, this was not done for declarations.

We use a stack of tokens sets that grows and shrinks in parallel with the accepting functions, to know more contextual informations when deciding wether to consume a token or not. The corresponding term is it shift when building the analysis tables in LR technology.

Part XIII

Debugging

Chapter 68

Debugging

Debugging MusicFormats can be quite time-consuming. The trace options available have designed to provide fine-grained tracing information to help locate issues.

function `catchSignals ()` in `main ()` functions

File `src/wae/enableAbortToDebugErrors.h` contains:

```
/*
  MusicFormats Library
  Copyright (C) Jacques Menu 2016-2022

  This Source Code Form is subject to the terms of the Mozilla Public
  License, v. 2.0. If a copy of the MPL was not distributed with this
  file, You can obtain one at http://mozilla.org/MPL/2.0/.

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

#ifndef ___enableAbortToDebugErrorsIfDesired___
#define ___enableAbortToDebugErrorsIfDesired___


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

#ifndef ABORT_TO_DEBUG_ERRORS
  #define ABORT_TO_DEBUG_ERRORS
#endif


#endif
```

68.1   Useful options

Here are the most basing options used when debugging:

- option `-trace-passes, -tpasses` this is the first option to use, to locate in which pass the problem arises

- option `-input-line-numbers, -iln` this option produces the music elements input-line numbers in the output files

- the `-display*` options

Part XIV

Indexes

Files index

Types index

Symbols

S

Methods and fields index

Constants, functions and variables index

Options

Symbols

MusicXML index

Main index

Main index