LilyPond

The music typesetter

Inside LilyPond

The LilyPond development team

This document presents the details of the LilyPond version 2.19.35 source code.

The information herein is not dealing with the usage of Lilypond.

This manual is not intended to be read sequentially; interested people can read only the sections which are relevant to them.

For more information about how this manual fits with the other documentation, or to read this manual in other formats, see Section "Manuals" in *General Information*.

If you are missing any manuals, the complete documentation can be found at `http://www.lilypond.org/`.

For LilyPond version 2.19.35

Table of Contents

1 Foreword 2
- 1.1 Context 2
  - 1.1.1 Version numbers 2
  - 1.1.2 Operating system 2
  - 1.1.3 User environment 3

2 Folders hierarchy 4
- 2.1 flower 4
- 2.2 lily 4
- 2.3 ly 4
- 2.4 scm 4
- 2.5 make 4
- 2.6 input/regression 5
- 2.7 build 5

3 C++ 7
- 3.1 Usage 7
- 3.2 Trampolines 7
- 3.3 Friends 8

4 Scheme integration 9
- 4.1 Scheme data types 9
- 4.2 Useful values 10
- 4.3 Scheme code 10
- 4.4 Scheme macros 11
- 4.5 Scheme evaluation 11

5 Python for LilyPond 14
- 5.1 python-ly 14

6 Compiler architecture 15
- 6.1 Terminology 15
- 6.2 Analysis and synthesis 15
- 6.3 Automatic analyzer generation 15

7 Lexical analysis 16
- 7.1 BOM (Byte Order Mark) 16
- 7.2 Flex specification 16
- 7.3 Some scanning details 16

8 Syntactical analysis 21
- 8.1 Pure LilyPond grammar 21
- 8.2 Bison specification 21
- 8.3 Some parsing details 22

9 Control flow 26
9.1 Global variables 26
9.2 Arguments and options 26
9.3 Main program 26
9.4 `gdb` 29
9.4.1 LilyPond source code 29
9.4.2 Compilation log 29
9.4.3 Backtrace 30
9.4.4 Incriminated source file 33

10 Music interpretation 34
10.1 Root function 34

11 The libmusicxml2 library 37
11.1 A MusicXML example 37
11.2 Installation on LilyDev 4 39
11.3 Library version number 41
11.4 Fundamental types 41
11.5 MusicXML lexical analysis 41
11.6 MusicXML syntactical analysis 42
11.7 MusicXML data representation 42
11.8 The Visitors design pattern 44
11.9 Browsing MusicXML trees 46
11.10 A visitor example 48
11.11 LilyPond data representation 48
11.12 xml2lilypond 51
11.13 Visitors in practise 52

Appendix A Index 54

Note: PRELIMINARY, DRAFT VERSION!

1 Foreword

LilyPond is a remarkable piece of software in many a way:

- it is a compiler from mixed LilyPond/Scheme input files to PostScript, PDF or PNG scores and MIDI files.
- it's implementation involves many languages:
  - C++ Used as a foundation to build upon, including `main()` function, options and arguments decoding
  - Flex Generates the lexer (lexical analyzer) C++ source code from the specification found in `lily/lexer.ll`;
  - Bison Used to generate the parser C++ source code from the specification in `lily/parser.hh`;
  - Scheme Used to represent the data handled by the compiler and a provide a means for the user to adapts LilyPond to various needs. The GNU Guile implementation of Scheme is used;
  - Python Used to write tools for the developper and the user, as well as Frescobaldi, a nice IDE;
  - TexInfo Used to create the documentation in PDF, OMF and HTML from the TexInfo source files.

We hope this documentation will help both the curious and the developpers, seasoned or new to the field. In other words, it is meant for the rest of us.

1.1 Context

1.1.1 Version numbers

When work started on this documentation, the version numbers were the following:

LilyDev	4, running Debian 8 32 bits (jessie)
lilypond, stable	2.18.2
lilypond, development	2.19.3x
g++	4.9.x
guile	1.8.x (2.y available)
LilyDev	4
frescobaldi	2.18.x
MusicXML	3.0 (3.2 announced)

1.1.2 Operating system

This documentation relies on the LilyDev virtual machine, but other Linux flavors can be used as well. You'll be more on your own in that case.

When you install LilyDev from the ISO image, a minimal configuration for your account is setup by this `bash` script:

```
user@lilydev: ~ > ls -sal .lilydev-setup.sh
4 -rwxr-xr-x 1 user user 1121 Dec 30 18:16 .lilydev-setup.sh
user@lilydev: ~ >
```

1.1.3 User environment

This documentation uses a `user` user account. Since there are command lines and their output, the following `bash` prompt is used to indicate the username, hostane and current working directory:

```
PS1=$( echo "\[\033[01;34m\]$USER@\h\[\033[00m\]: \w"" > " )
```

For example, here are the informations about `lily/volta-engraver.cc`:

```
user@lilydev: ~/lilypond-git/lily > ls -sal volta-engraver.cc
8 -rw-r--r-- 1 user user 5409 Dec 30 18:25 volta-engraver.cc
user@lilydev: ~/lilypond-git/lily >
```

2 Folders hierarchy

This chapter presents the top-level folders and their use.

2.1 flower

This C++ library contains some general purpose routines which were not or are not standardised libraries yet. It may be replaced by STL in time.

```
user@lilydev: ~/lilypond/flower > ls
GNUmakefile  VERSION         getopt-long.cc    libc-extension.cc  real.cc            test-interval-set.cc
NEWS-1.0     cpu-timer.cc    include           memory-stream.cc   std-string.cc      test-std.cc
NEWS-1.1.46  file-cookie.cc  international.cc  offset.cc          string-convert.cc  test-string.cc
README       file-name.cc    interval-set.cc   polynomial.cc      test-file-name.cc  warn.cc
TODO         file-path.cc    interval.cc       rational.cc        test-file-path.cc
user@lilydev: ~/lilypond/flower > ls include
arithmetic-operator.hh  file-name.hh            interval.hh        polynomial.hh      tuple.hh
axis.hh                 file-path.hh            interval.tcc       pqueue.hh          virtual-methods.hh
compare.hh              flower-proto.hh         libc-extension.hh  rational.hh        warn.hh
cpu-timer.hh            getopt-long.hh          matrix.hh          real.hh            yaffut-parameters.h
direction.hh            guile-compatibility.hh  memory-stream.hh   std-string.hh      yaffut.hh
drul-array.hh           international.hh        offset.hh          std-vector.hh
file-cookie.hh          interval-set.hh         parray.hh          string-convert.hh
```

2.2 lily

This folder contains the C++ code, both headers in `./include` and implementation.

2.3 ly

This folder Scheme code, among which the language 'definitions' are legacy:

```
user@lilydev: ~/lilypond/ly > grep -i legacy *
catalan.ly:%%%% Legacy file. (see scm/define-note-names.scm)
deutsch.ly:%%%% Legacy file. (see scm/define-note-names.scm)
english.ly:%%%% Legacy file. (see scm/define-note-names.scm)
espanol.ly:%%%% Legacy file. (see scm/define-note-names.scm)
italiano.ly:%%%% Legacy file. (see scm/define-note-names.scm)
nederlands.ly:%%%% Legacy file. (see scm/define-note-names.scm)
norsk.ly:%%%% Legacy file. (see scm/define-note-names.scm)
portugues.ly:%%%% Legacy file. (see scm/define-note-names.scm)
suomi.ly:%%%% Legacy file. (see scm/define-note-names.scm)
svenska.ly:%%%% Legacy file. (see scm/define-note-names.scm)
vlaams.ly:%%%% Legacy file. (see scm/define-note-names.scm)
```

2.4 scm

The bulk of Scheme code.

2.5 make

This folder contains `make` specifications to be imported by actual `GNUmakefile`'s:

```
user@lilydev: ~/lilypond/make > ls
abc-rules.make              generic-vars.make           ly-targets.make      midi-vars.make           ste
abc-targets.make            lilypond-book-rules.make    ly-vars.make         musicxml-rules.make      sub
abc-vars.make               lilypond-book-targets.make  ly.make              musicxml-targets.make    top
doc-i18n-root-rules.make    lilypond-book-vars.make     lysdoc-rules.make    musicxml-vars.make       web
doc-i18n-root-targets.make  lilypond-rules.make         lysdoc-targets.make  mutopia-inclusions.make
doc-i18n-root-vars.make     lilypond-targets.make       lysdoc-vars.make     mutopia-rules.make
generic-rules.make          lilypond-vars.make          midi-rules.make      mutopia-targets.make
generic-targets.make        ly-rules.make               midi-targets.make    mutopia-vars.make
```

2.6 input/regression

The regression tests. The `texidoc` field of the `header` specifications are used by `texinfo`.

2.7 build

This folder is where LilyPond is built by convention. To start again building from the beginning, remove it and run something like this script:

```
user@lilydev: ~/lilypond-git > cat BuildLilypond.bash
#!/bin/bash

cd $LILYPOND_GIT

sh autogen.sh --noconfigure

mkdir -p build/
cd build/

../configure

cd $LILYPOND_GIT/build/
make
```

Folder `build` also contains the documentation, in particular:

- The `lilypond` executable:

```
user@lilydev: ~/lilypond-git > ll build/lily/out/lilypond
56160 -rwxr-xr-x 1 user 57507120 Jan  8 09:27 build/lily/out/lilypond*
```

- The various Python utilities and their `man` page file:

```
user@lilydev: ~/lilypond-git > ll build/scripts/out
total 348
  4 drwxr-xr-x 2 user   4096 Jan  8 09:04 ./
  4 drwxr-xr-x 5 user   4096 Jan  8 09:33 ../
  4 -rw-r--r-- 1 user      2 Jan  8 09:04 .gitignore
 44 -rwxr-xr-x 1 user  41358 Jan  8 09:27 abc2ly*
  4 -rw-r--r-- 1 user   1073 Jan  8 09:27 abc2ly.1
 16 -rwxr-xr-x 1 user  12863 Jan  8 09:27 convert-ly*
  4 -rw-r--r-- 1 user   1920 Jan  8 09:27 convert-ly.1
  0 -rw-r--r-- 1 user      0 Jan  8 09:04 dummy.dep
 36 -rwxr-xr-x 1 user  34358 Jan  8 09:27 etf2ly*
  4 -rw-r--r-- 1 user   1018 Jan  8 09:27 etf2ly.1
 12 -rwxr-xr-x 1 user   8923 Jan  8 09:27 lilymidi*
  4 -rw-r--r-- 1 user    987 Jan  8 09:27 lilymidi.1
 28 -rwxr-xr-x 1 user  27621 Jan  8 09:27 lilypond-book*
  4 -rw-r--r-- 1 user   3042 Jan  8 09:27 lilypond-book.1
  8 -rwxr-xr-x 1 user   5316 Jan  8 09:27 lilypond-invoke-editor*
  4 -rw-r--r-- 1 user    833 Jan  8 09:27 lilypond-invoke-editor.1
  8 -rwxr-xr-x 1 user   7827 Jan  8 09:27 lilysong*
  4 -rw-r--r-- 1 user   1038 Jan  8 09:27 lilysong.1
 36 -rwxr-xr-x 1 user  36091 Jan  8 09:27 midi2ly*
  4 -rw-r--r-- 1 user   1929 Jan  8 09:27 midi2ly.1
112 -rwxr-xr-x 1 user 112199 Jan  8 09:27 musicxml2ly*
  4 -rw-r--r-- 1 user   2441 Jan  8 09:27 musicxml2ly.1
```

- The `build/Documentation/lilypond/*.pdf` PDF files

3 C++

3.1 Usage

3.2 Trampolines

Trampolines are a programming technique used to avoid tail function calls. Instead of calling a function, the caller returns a way to call it, and the call itself is done in a loop outside the caller, see:

`http://stackoverflow.com/questions/189725/what-is-a-trampoline-function`
this introduction to trampolines (`http://www.artificialworlds.net/blog/2012/04/30/tail-call-optimisation-in-cpp/`)

In `lily/include/smobs.hh`:

```
// Well, function template argument packs are a C++11 feature.  So
// we just define a bunch of trampolines manually.  It turns out
// that GUILE 1.8.8 cannot actually make callable structures with
// more than 3 arguments anyway.  That's surprising, to say the
// least, but in emergency situations one can always use a "rest"
// argument and take it apart manually.
```

In `lily/grob-array.cc`:

```
void
Grob_array::filter (bool (*predicate) (const Grob *))
{
 vsize new_size = 0;
 for (vsize i = 0; i < grobs_.size (); ++i)
   if (predicate (grobs_[i]))
     grobs_[new_size++] = grobs_[i];
 grobs_.resize (new_size);
 // could call grobs_.shrink_to_fit () with C++11
}
```

```
void
Grob_array::filter_map (Grob * (*map_fun) (Grob *))
{
 vsize new_size = 0;
 for (vsize i = 0; i < grobs_.size (); ++i)
   if (Grob *grob = map_fun (grobs_[i]))
     grobs_[new_size++] = grob;
 grobs_.resize (new_size);
 // could call grobs_.shrink_to_fit () with C++11
}
```

```
void
Grob_array::filter_map_assign (const Grob_array &src,
                               Grob * (*map_fun) (Grob *))
{
 if (&src != this)
   {
     grobs_.resize (0);
     grobs_.reserve (src.grobs_.size ());
     for (vsize i = 0; i < src.grobs_.size (); i++)
```

```
      if (Grob *grob = map_fun (src.grobs_[i]))
        grobs_.push_back (grob);
    // could call grobs_.shrink_to_fit () with C++11
  }
 else
  filter_map (map_fun);
}
const char Grob_array::type_p_name_[] = "ly:grob-array? »;
```

3.3 Friends

The only `friend` declarations occur in `lily/context-property.cc`:

```
class Grob_properties : public Simple_smob<Grob_properties>
{
public:
 SCM mark_smob () const;
 static const char type_p_name_[];
private:
 friend class Grob_property_info;
 friend SCM ly_make_grob_properties (SCM);
 // alist_ may contain unexpanded nested overrides
 SCM alist_;
 // based_on_ is the cooked_ value from the next higher context that
 // alist_ is based on
 SCM based_on_;
 // cooked_ is a version of alist_ where nested overrides have been
 // expanded
 SCM cooked_;
 // cooked_from_ is the value of alist_ from which the expansion has
 // been done
 SCM cooked_from_;
 // nested_ is a count of nested overrides in alist_ Or rather: of
 // entries that must not appear in the cooked list and are
 // identified by having a "key" that is not a symbol.  Temporary
 // overrides and reverts also meet that description and have a
 // nominal key of #t/#f and a value of the original cons cell.
 int nested_;

 Grob_properties (SCM alist, SCM based_on) :
   alist_ (alist), based_on_ (based_on),
   // if the constructor was called with lists possibly containing
   // partial overrides, we would need to initialize with based_on in
   // order to trigger an initial update.  But this should never
   // happen, so we initialize straight with alist.
   cooked_ (alist), cooked_from_ (alist), nested_ (0) { }
};
```

4 Scheme integration

4.1 Scheme data types

From the Guile Reference Manual:

> "In Guile, this uniform representation of all Scheme values is the C type SCM. This is an opaque type and its size is typically equivalent to that of a pointer to void. Thus, SCM values can be passed around e ciently and they take up reasonably little storage on their own".

and:

> "`SCM` is the user level abstract C type that is used to represent all of Guile's Scheme objects, no matter what the Scheme object type is. No C operation except assignment is guaranteed to work with variables of type `SCM`, so you should only use macros and functions to work with `SCM` values. Values are converted between C data types and the `SCM` type with utility functions and macros".

The curious will find in `/usr/include/libguile/tags.h` the definition of type `SCM`, that may vary depending of the performance and debug goals when compiling Guile.

A smob (ScheMe OBject) is a C++ object that is encapsulated so it can be used as a Scheme object.

Function `unsmob()` is very often used:

```
[Type *] unsmob<Type> (SCM s)
This tries converting a Scheme object to a pointer of the desired kind.
If the Scheme object is of the wrong type, a pointer value of 0 is returned,
making this suitable for a Boolean test.
```

See `lily/include/smobs.hh` for more information.

Basic type checking functions such as `string?` and `int?` are defined by Scheme itself.

LilyPond's type checking function `ly:music?` is defined in `lily/music-scheme.cc`:

```
LY_DEFINE (ly_music_p, "ly:music?",
           1, 0, 0, (SCM obj),
           "Is @var{obj} a music object?")
{
  return ly_bool2scm (unsmob<Music> (obj));
}
```

Is is associated with the name `music` for use by warning and error messages in `scm/lily.scm`:

```
(define-public lilypond-exported-predicates
  `((,ly:book? . "book")
    (,ly:box? . "box")
    (,ly:context? . "context")
    ; ...
    (,ly:music? . "music")
    ; ...
    (,ly:unpure-pure-container? . "unpure/pure container")
    ))
```

4.2 Useful values

From the Guile Reference Manual:

- `SCM SCM_EOL`

  The Scheme empty list object, or "End Of List" object, usually written in Scheme as `'()`.

- `SCM SCM_EOF_VAL`

  The Scheme end-of-file value. It has no standard written representation, for obvious reasons.

- `SCM SCM_UNSPECIFIED`

  The value returned by some (but not all) expressions that the Scheme standard says return an "unspecified" value.

  This is sort of a weirdly literal way to take things, but the standard read-eval-print loop prints nothing when the expression returns this value, so it"s not a bad idea to return this when you can"t think of anything else helpful.

- `SCM SCM_UNDEFINED`

  The "undefined" value. Its most important property is that is not equal to any valid Scheme value. This is put to various internal uses by C code interacting with Guile. For example, when you write a C function that is callable from Scheme and which takes optional arguments, the interpreter passes `SCM_UNDEFINED` for any arguments you did not receive. We also use this to mark unbound variables.

4.3 Scheme code

From the Guile Reference Manual:

```
@code{int SCM_UNBNDP (SCM x)}

Return @code{true} if @code{@var{x}} is @code{SCM_UNDEFINED}.
Note that this is not a check to see if @code{@var{x}} is @code{SCM_UNBOUND}. History w
```

Ancillary functions are defined in `lily/lily-guile.cc`, such as:

```
/*
  STRINGS
 */
string
ly_scm2string (SCM str)
{
  assert (scm_is_string (str));
  string result;
  size_t len = scm_c_string_length (str);
  if (len)
    {
      result.resize (len);
      scm_to_locale_stringbuf (str, &result.at (0), len);
    }
  return result;
}

SCM
ly_string2scm (string const &str)
{
  return scm_from_locale_stringn (str.c_str (),
```

```
                                    str.length ());
}
char *
ly_scm2str0 (SCM str)
{
  return scm_to_utf8_string (str);
}
```

4.4 Scheme macros

They are defined in `lily/include/lily-guile-macros.hh`, among them:

```
#define LY_DEFINE_WITHOUT_DECL(INITPREFIX, FNAME, PRIMNAME, REQ, OPT, VAR, \
                               ARGLIST, DOCSTRING)                          \
  SCM FNAME ## _proc;                                                       \
  void                                                                      \
  INITPREFIX ## init ()                                                     \
  {                                                                         \
    FNAME ## _proc = scm_c_define_gsubr (PRIMNAME, REQ, OPT, VAR,           \
                                         (scm_t_subr) FNAME); \
    ly_check_name (#FNAME, PRIMNAME);\
    ly_add_function_documentation (FNAME ## _proc, PRIMNAME, #ARGLIST,  \
                                   DOCSTRING);                          \
    scm_c_export (PRIMNAME, NULL);                                      \
  }                                                                         \
  ADD_SCM_INIT_FUNC (INITPREFIX ## init_unique_prefix, INITPREFIX ## init); \
  SCM                                                                       \
  FNAME ARGLIST
#define LY_DEFINE(FNAME, PRIMNAME, REQ, OPT, VAR, ARGLIST, DOCSTRING)   \
  SCM FNAME ARGLIST;                                                    \
  LY_DEFINE_WITHOUT_DECL (FNAME, FNAME, PRIMNAME, REQ, OPT, VAR, ARGLIST, \
                          DOCSTRING)
#define LY_DEFINE_MEMBER_FUNCTION(CLASS, FNAME, PRIMNAME, REQ, OPT, VAR, \
                                  ARGLIST, DOCSTRING)                   \
SCM FNAME ARGLIST;                                                      \
LY_DEFINE_WITHOUT_DECL (CLASS ## FNAME, CLASS::FNAME, PRIMNAME, REQ, OPT, \
                        VAR, ARGLIST, DOCSTRING)
```

Function `LY_DEFINE_MEMBER_FUNCTION` is defined but not used:

```
#define LY_DEFINE_MEMBER_FUNCTION(CLASS, FNAME, PRIMNAME, REQ, OPT, VAR, \
                                  ARGLIST, DOCSTRING)                   \
  SCM FNAME ARGLIST;                                                    \
  LY_DEFINE_WITHOUT_DECL (CLASS ## FNAME, CLASS::FNAME, PRIMNAME, REQ, OPT, \
                          VAR, ARGLIST, DOCSTRING)
```

4.5 Scheme evaluation

It starts from `parser.yy`, after a `SCM_TOKEN` has been accepted, as in:

```
toplevel_expression:
  {
    parser->lexer_->add_scope (get_header (parser));
  } lilypond_header {
```

```
      parser->lexer_->set_identifier (ly_symbol2scm ("$defaultheader"), $2);
    }
    | /* ... */
    | SCM_TOKEN {
      // Evaluate and ignore #xxx, as opposed to \xxx
      parser->lexer_->eval_scm_token ($1, @1);
    | /* ... */
```

Function `eval_scm_token()` is defined in `lily/include/lily-lexer.hh`:

```
class Lily_lexer : public Smob<Lily_lexer>, public Includable_lexer
{
public:
  int print_smob (SCM, scm_print_state *) const;
  SCM mark_smob () const;
  static const char type_p_name_[];
  virtual ~Lily_lexer ();
private:
  // ...
  Lily_parser *parser_;
  Keyword_table *keytable_;
  SCM scopes_;
  SCM start_module_;
  Input override_input_;
  SCM eval_scm (SCM, Input, char extra_token = 0);
public:
  SCM eval_scm_token (SCM sval, Input w)
  {
    w.step_forward ();
    return eval_scm (sval, w, '#');
  }
  // ...
}
```

Member function `eval_scm()` is defined in `lily/lexer.ll`:

```
SCM
Lily_lexer::eval_scm (SCM readerdata, Input hi, char extra_token)
{
  SCM sval = SCM_UNDEFINED;

  if (!SCM_UNBNDP (readerdata))
  {
    sval = ly_eval_scm (readerdata,
            hi,
            be_safe_global && is_main_input_,
            parser_);
  }

  if (SCM_UNBNDP (sval))
  {
    error_level_ = 1;
    return SCM_UNSPECIFIED;
```

```
  }

  if (extra_token && SCM_VALUESP (sval))
  {
    sval = scm_struct_ref (sval, SCM_INUM0);

    if (scm_is_pair (sval)) {
      for (SCM p = scm_reverse (scm_cdr (sval));
           scm_is_pair (p);
           p = scm_cdr (p))
      {
        SCM v = scm_car (p);
        if (Music *m = unsmob<Music> (v))
        {
          if (!unsmob<Input> (m->get_property ("origin")))
            m->set_spot (override_input (here_input ()));
        }

        int token;
        switch (extra_token) {
        case '$':
          token = scan_scm_id (v);
          if (!scm_is_eq (yylval, SCM_UNSPECIFIED))
            push_extra_token (here_input (),
                  token, yylval);
          break;
        case '#':
          push_extra_token (here_input (),
                SCM_IDENTIFIER, v);
          break;
        }
      }
      sval = scm_car (sval);
    } else
      sval = SCM_UNSPECIFIED;
  }

  if (Music *m = unsmob<Music> (sval))
  {
    if (!unsmob<Input> (m->get_property ("origin")))
      m->set_spot (override_input (here_input ()));
  }

  return sval;
}
```

5 Python for LilyPond

5.1 python-ly

In `ly/musicxml/create_musicxml.py`:

```
class CreateMusicXML():
   """ Creates the XML nodes according to the Music XML standard. » » »
```

In `ly/musicxml/lymus2musxml.py`:

```
Using the tree structure from ly.music to initiate the conversion to MusicXML.

Uses functions similar to ly.music.items.Document.iter_music() to iter through
the node tree. But information about where a node branch ends
is also added. During the iteration the information needed for the conversion
is captured.
```

In `ly/cli/doc.py`:

Usage::

ly [options] commands file, ...

A tool for manipulating LilyPond source files

6 Compiler architecture

6.1 Terminology

Function `Performance::write_output` is defined in `lily/performance.cc`:

6.2 Analysis and synthesis

6.3 Automatic analyzer generation

7 Lexical analysis

Flex is called this way:

```
user@lilydev: ~/lilypond/build > grep flex config.make
FLEX = flex
user@lilydev: ~/lilypond/stepmake/stepmake > grep FLEX *
c++-rules.make: $(FLEX) -Cfe -p -p -o$@ $<
c-rules.make: $(FLEX) -Cfe -p -p -o$@ $<
c-rules.make:#  $(FLEX) -8 -Cf -o$@ $<
```

7.1 BOM (Byte Order Mark)

It is composed of the following three bytes:

```
BOM_UTF8  \357\273\277
```

7.2 Flex specification

The file `lily/lexer.ll` uses states (also known as modes):

```
user@lilydev: ~/lilypond/lily > grep %x lexer.ll
%x chords
%x figures
%x incl
%x lyrics
%x longcomment
%x maininput
%x markup
%x notes
%x quote
%x commandquote
%x sourcefileline
%x sourcefilename
%x version
```

Changing to some state is done with `yy_push_state()` and switching back to the previous state is done with `yy_pop_state()`.

The information describing the current token, if needed, is passed to the parser in `yylval`, accessed thru pointer `lexval_`:

```
#define yylval (*lexval_)

#define yylloc (*lexloc_)
```

The header file `lily/include/lily-lexer.hh` contains:

```
  SCM *lexval_;
  Input *lexloc_;
```

7.3 Some scanning details

Flex compiles `lily/lexer.ll` into `build/lily/out/lexer.cc`, that contains the code of the lexical analyzer:

```
#define YY_DECL int Lily_lexer::yylex()
// ...
/** The main scanner function which does all the work.
 */
YY_DECL
{
```

```
register yy_state_type yy_current_state;
register char *yy_cp, *yy_bp;
register int yy_act;
// ...
```

This function is the actual lexical analyzer and returns a value describing the current token. The token themselves are defined in `lily/parser.yy`:

```
user@lilydev: ~/lilypond/lily > grep %token parser.yy
%token END_OF_FILE 0 "end of input"
%token ACCEPTS "\\accepts"
%token ADDLYRICS "\\addlyrics"
%token ALIAS "\\alias"
%token ALTERNATIVE "\\alternative"
%token BOOK "\\book"
%token BOOKPART "\\bookpart"
%token CHANGE "\\change"
%token CHORDMODE "\\chordmode"
%token CHORDS "\\chords"
%token CONSISTS "\\consists"
%token CONTEXT "\\context"
%token DEFAULT "\\default"
%token DEFAULTCHILD "\\defaultchild"
%token DENIES "\\denies"
%token DESCRIPTION "\\description"
%token DRUMMODE "\\drummode"
%token DRUMS "\\drums"
%token ETC "\\etc"
%token FIGUREMODE "\\figuremode"
%token FIGURES "\\figures"
%token HEADER "\\header"
%token INVALID "\\version-error"
%token LAYOUT "\\layout"
%token LYRICMODE "\\lyricmode"
%token LYRICS "\\lyrics"
%token LYRICSTO "\\lyricsto"
%token MARKUP "\\markup"
%token MARKUPLIST "\\markuplist"
%token MIDI "\\midi"
%token NAME "\\name"
%token NOTEMODE "\\notemode"
%token OVERRIDE "\\override"
%token PAPER "\\paper"
%token REMOVE "\\remove"
%token REPEAT "\\repeat"
%token REST "\\rest"
%token REVERT "\\revert"
%token SCORE "\\score"
%token SCORELINES "\\score-lines"
%token SEQUENTIAL "\\sequential"
%token SET "\\set"
%token SIMULTANEOUS "\\simultaneous"
```

```
%token TEMPO "\\tempo"
%token TYPE "\\type"
%token UNSET "\\unset"
%token WITH "\\with"
%token NEWCONTEXT "\\new"
%token CHORD_BASS "/+"
%token CHORD_CARET "^"
%token CHORD_COLON ":"
%token CHORD_MINUS "-"
%token CHORD_SLASH "/"
%token ANGLE_OPEN "<"
%token ANGLE_CLOSE ">"
%token DOUBLE_ANGLE_OPEN "<<"
%token DOUBLE_ANGLE_CLOSE ">>"
%token E_BACKSLASH "\\"
%token E_EXCLAMATION "\\!"
%token E_PLUS "\\+"
%token EXTENDER "__"
%token FIGURE_CLOSE /* "\\>" */
%token FIGURE_OPEN /* "\\<" */
%token FIGURE_SPACE "_"
%token HYPHEN "--"
%token MULTI_MEASURE_REST
%token E_UNSIGNED
%token UNSIGNED
%token EXPECT_MARKUP "markup?"
%token EXPECT_SCM "scheme?"
%token BACKUP "(backed-up?)"
%token REPARSE "(reparsed?)"
%token EXPECT_MARKUP_LIST "markup-list?"
%token EXPECT_OPTIONAL "optional?"
%token EXPECT_NO_MORE_ARGS;
%token EMBEDDED_LILY "#{"
%token BOOK_IDENTIFIER
%token CHORD_MODIFIER
%token CHORD_REPETITION
%token CONTEXT_MOD_IDENTIFIER
%token DRUM_PITCH
%token PITCH_IDENTIFIER
%token DURATION_IDENTIFIER
%token EVENT_IDENTIFIER
%token EVENT_FUNCTION
%token FRACTION
%token LYRIC_ELEMENT
%token MARKUP_FUNCTION
%token MARKUP_LIST_FUNCTION
%token MARKUP_IDENTIFIER
%token MARKUPLIST_IDENTIFIER
%token MUSIC_FUNCTION
%token MUSIC_IDENTIFIER
%token NOTENAME_PITCH
%token NUMBER_IDENTIFIER
```

```
%token REAL
%token RESTNAME
%token SCM_ARG
%token SCM_FUNCTION
%token SCM_IDENTIFIER
%token SCM_TOKEN
%token STRING
%token SYMBOL_LIST
%token TONICNAME_PITCH
```

Some tokens are actually members of a class of tokens, such as strings, fractions and numbers. In this case, Scheme data describing the particular member of the class is stored into `yylval` for use by the parser:

```
{FRACTION}  {
  yylval =  scan_fraction (YYText ());
  return FRACTION;
}
```

The specification for `FRACTION` above becomes in `build/lily/out/lexer.cc` the code to handle one of the states of the finite state machine used by the lexical analyzer:

```
case 50:
YY_RULE_SETUP
#line 490 "/home/user/lilypond-git/lily/lexer.ll"
{
    yylval =  scan_fraction (YYText ());
    return FRACTION;
  }
  YY_BREAK
```

The code of `scan_fraction()` itself, placed after the second '%%' separator in `lily/lexer.ll`, is copied verbatim, without any treatment, into `build/lily/out/lexer.cc`:

```
/*
Convert "NUM/DEN" into a '(NUM . DEN) cons.
*/
SCM
scan_fraction (string frac)
{
  ssize i = frac.find ('/');
  string left = frac.substr (0, i);
  string right = frac.substr (i + 1, (frac.length () - i + 1));

  return scm_cons (scm_c_read_string (left.c_str ()),
       scm_c_read_string (right.c_str ()));
}
```

Note that the lexer provides `push_extra_token()` to force its argument to "seen" although it is not there:

```
/* Make the lexer generate a token of the given type as the next token.
 TODO: make it possible to define a value for the token as well */
void
Lily_lexer::push_extra_token (Input const &where, int token_type, SCM scm)
```

```
{
  extra_tokens_ = scm_cons (scm_cons2 (where.smobbed_copy (),
                scm_from_int (token_type),
                scm), extra_tokens_);
}
```

This facility is used both by the lexical and synctactical analyzers.

8 Syntactical analysis

Bison compiles `lily/parser.yy` into `build/lily/out/parser.hh` and `build/lily/out/parser.cc`. It is called this way:

```
user@lilydev: ~/lilypond/build > grep bison config.make
BISON = bison
YACC = bison -y

user@lilydev: ~/lilypond/stepmake/stepmake > grep BISON *
c++-rules.make: $(BISON) -d -o $(outdir)/$*.cc $<
c-rules.make: $(BISON) -d -o $(outdir)/$*.c $<
```

8.1 Pure LilyPond grammar

In can be found in `./build/Documentation/out-www/ly-grammar.txt`, with all the details of the states of the LALR(1) parser generated by Bison.

8.2 Bison specification

The type of symbols is defined as `SCM`:

```
#define YYSTYPE SCM
```

Hence all non-terminals are described by Scheme data.

The lexical analyzer `yylex()` is defined in `lily/parser.yy` this way:

```
int
yylex (YYSTYPE *s, YYLTYPE *loc, Lily_parser *parser)
{
  Lily_lexer *lex = parser->lexer_;

  lex->lexval_ = s;
  lex->lexloc_ = loc;
  int tok = lex->pop_extra_token ();
  if (tok >= 0)
    return tok;
  lex->prepare_for_next_token ();
  return lex->yylex ();
}
```

It is called thru macro `YYLEX`, defined this way:

```
/* YYLEX -- calling `yylex' with the right arguments.  */
#ifdef YYLEX_PARAM
# define YYLEX yylex (&yylval, &yylloc, YYLEX_PARAM)
#else
# define YYLEX yylex (&yylval, &yylloc, parser)
#endif
```

The axiom of the grammar is:

```
start_symbol:
  lilypond
  | EMBEDDED_LILY {
    SCM nn = parser->lexer_->lookup_identifier ("pitchnames");
```

```
  parser->lexer_->push_note_state (nn);
} embedded_lilypond {
  parser->lexer_->pop_state ();
              *retval = $3;
}
;
```

The "lilypond" non-terminal itself is defined as:

```
lilypond: /* empty */ { $$ = SCM_UNSPECIFIED; }
  | lilypond toplevel_expression {
  }
  | lilypond assignment {
  }
  | lilypond error {
  parser->error_level_ = 1;
  }
  | lilypond INVALID  {
  parser->error_level_ = 1;
  }
  ;
```

Terminal `INVALID` describes an invalid version number, and `error` is the pseudo-terminal indicating that a syntax error has occured, a situation handled by Bison in a special way.

An artificial token is handled for embedded LilyPond code:

```
/* An artificial token for parsing embedded Lilypond */
%token EMBEDDED_LILY "#{"
```

8.3 Some parsing details

The actual parsing of a LilyPond input file is done by `lily/Lily_parser::parse_file`:

```
/* Process one .ly file, or book.  */
void
Lily_parser::parse_file (const string &init, const string &name, const string &out_name
{
  output_basename_ = out_name;

  lexer_->main_input_name_ = name;

  message (_ ("Parsing..."));

  set_yydebug (0);

  lexer_->new_input (init, sources_);

  File_name f (name);
  string s = global_path.find (f.base_ + ".twy");
  s = gulp_file_to_string (s, false, -1);
  scm_eval_string (ly_string2scm (s));

  /* Read .ly IN_FILE, lex, parse, write \score blocks from IN_FILE to
     OUT_FILE (unless IN_FILE redefines output file name).  */
```

```
  SCM mod = lexer_->set_current_scope ();
  do
    {
      do_yyparse ();
    }
  while (!lexer_->is_clean ());

  ly_reexport_module (scm_current_module ());

  scm_set_current_module (mod);

  error_level_ = error_level_ | lexer_->error_level_;
  clear ();
}
```

Function `do_yyparse()` is defined in `lily/parser.yy`:

```
SCM
Lily_parser::do_yyparse ()
{
  return scm_c_with_fluid (Lily::f_parser,
         self_scm (),
         do_yyparse_trampoline,
         static_cast <void *>(this));
}

SCM
Lily_parser::do_yyparse_trampoline (void *parser)
{
  SCM retval = SCM_UNDEFINED;
  yyparse (static_cast <Lily_parser *>(parser), &retval);
  return retval;
}
```

Function `scm_c_with_fluid ()` is defined by Guile, see also `fluid.hh` (there is no `fluid.cc` though).

From the Guile Reference Manual:

> A fluid is an object that can store one value per dynamic state. Each thread has a current dynamic state, and when accessing a fluid, this current dynamic state is used to provide the actual value. In this way, fluids can be used for thread local storage, but they are in fact more flexible: dynamic states are objects of their own and can be made current for more than one thread at the same time, or only be made current temporarily, for example.
>
> Fluids can also be used to simulate the desirable effects of dynamically scoped variables. Dynamically scoped variables are useful when you want to set a variable to a value during some dynamic extent in the execution of your program and have them revert to their original value when the control flow is outside of this dynamic extent. See the description of with-fluids below for details.

New fluids are created with `make-fluid` and `fluid?` is used for testing whether an object is actually a fluid. The values stored in a fluid can be accessed with `fluid-ref` and `fluid-set!`.

From the Guile Reference Manual:

```
SCM scm_c_with_fluids (SCM fluids, SCM vals, SCM (*cproc)(void *), void *data) [C Function]
SCM scm_c_with_fluid (SCM fluid, SCM val, SCM (*cproc)(void *), void *data) [C Function]

The function scm_c_with_fluids is like scm_with_fluids except that it takes a C function to call in-
stead of a Scheme thunk.
The function scm_c_with_fluid is similar but only allows one fluid to be set instead of a list.
with-fluid* fluid value thunk [Scheme Procedure]
scm_with_fluid (fluid, value, thunk) [C Function]

Set fluid to value temporarily, and call thunk. thunk must be a procedure with no
argument.
```

Variable `f_parser` is a `fluid` used for parsing. It is declared in `lily-imports.hh`:

```
namespace Lily {
  extern Scm_module module;
  typedef Module_variable<module> Variable;
  // ...
  extern Variable f_parser;
  // ...
};
```

It is defined in `lily-imports.cc`:

```
amespace Lily {
  Scm_module module ("lily");
  // ...
  Variable f_parser ("%parser");
  // ...
}
```

Class `Module_variable` id defined in `lily-modules.hh`:

```
template <Scm_module &m>
class Module_variable : public Scm_variable
{
public:
  Module_variable (const char *name, SCM value = SCM_UNDEFINED)
    : Scm_variable (m, name, value)
  { }
};
```

Member function `self_scm()` is declared in `smobs.hh`. See the comment in the latter for more information about its use:

```
// The Smob_core class is not templated and contains material not
// depending on the Super class.

class Smob_core {
protected:
  SCM self_scm_;
```

```
  Smob_core () : self_scm_ (SCM_UNDEFINED) { };
public:
  SCM self_scm () const { return self_scm_; }
  Listener get_listener (SCM callback);
};
```

Only Member function `Smob_core::get_listener()` is defined in `smobs.cc`:

```
Listener
Smob_core::get_listener (SCM callback)
{
  return Listener (callback, self_scm ());
}
```

Function `yyparse()` is the syntactic analyzer synthesised by Bison in `build/lily/out/parser.cc` from `lily/parser.yy`:

```
/*----------.
| yyparse.  |
`----------*/

#ifdef YYPARSE_PARAM
#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
int
yyparse (void *YYPARSE_PARAM)
#else
int
yyparse (YYPARSE_PARAM)
    void *YYPARSE_PARAM;
#endif
#else /* ! YYPARSE_PARAM */
#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
int
yyparse (Lily_parser *parser, SCM *retval)
#else
int
yyparse (parser, retval)
    Lily_parser *parser;
    SCM *retval;
#endif
#endif
{
// ...
```

9 Control flow

9.1 Global variables

They are declared in `lily/include/main.hh` and defined in `lily/global-vars.cc`. Among them:

```
/* Current XML trace name. */
string xml_trace_name_global;

/* Compile XML file? */
bool compile_xml_file = false;
```

9.2 Arguments and options

They are decoded in `lily/main.cc`:

```
static Long_option_init options_static[]

// ...

static void
parse_argv (int argc, char **argv)
{
  bool show_help = false;
  option_parser = new Getopt_long (argc, argv, options_static);
  while (Long_option_init const *opt = (*option_parser) ())
    {
      switch (opt->shortname_char_)
        {
        // ...
```

The actual "main()" function is `main_with_guile()`, called by:

```
scm_boot_guile (argc, argv, main_with_guile, 0);
```

9.3 Main program

The actual launching is done by `Lily::lilypond_main()` in `lily/main.cc`:

```
static void
main_with_guile (void *, int, char **)
/*
 * main-with-guile is invoked as a callback via scm_boot_guile from
 * main.
 * scm_boot_guile will have passed its data, argc and argv parameters
 * to main_with_guile.
 */

 // ..

 /*
   Now execute the Scheme entry-point declared in
   lily.scm (lilypond-main)
 */
```

```
  // These commands moved to lily_guile_v2.scm
  // SCM rep_mod = scm_c_resolve_module ("system repl repl");
  // scm_c_use_module ("system repl repl");
  // SCM err_handling_mod = scm_c_resolve_module ("system repl error-handling");
  // SCM call_with_error_handling = scm_c_module_lookup (err_handling_mod, "call-with-e
  // SCM result = scm_call_1 (
  //         scm_variable_ref (call_with_error_handling),
  //         scm_call_1 (ly_lily_module_constant ("lilypond-main"), files));

  Lily::lilypond_main (files);

  /* Unreachable.  */
  exit (0);
}
```

This function is defined in `scm/lily.scm` as:

```
(define-public (lilypond-main files)
  "Entry point for LilyPond."
  ; ...
  (let ((failed (lilypond-all files)))
    (if (ly:get-option 'trace-scheme-coverage)
        (begin
          (coverage:show-all (lambda (f)
                               (string-contains f "lilypond")))))
    (if (pair? failed)
        (begin (ly:error (_ "failed files: ~S") (string-join failed))
               (ly:exit 1 #f))
        (begin
          (ly:exit 0 #f)))))
```

Function `lilypond-all()` handles each input file *x* in turn. `handler` is a lambda-expression to accumulate the filenames that caused a failure:

```
          (lilypond-file handler x)
```

So the actual handling of each file is done by:

```
(define (lilypond-file handler file-name)
  (catch 'ly-file-failed
         (lambda () (ly:parse-file file-name))
         (lambda (x . args) (handler x file-name))))
```

Function `ly:parse-file()` in turn launches the analysis and treatment of the LilyPond input file up to the production of the score. It is defined in `lily/lily-parser-scheme.cc`:

```
LY_DEFINE (ly_parse_file, "ly:parse-file",
           1, 0, 0, (SCM name),
           "Parse a single @code{.ly} file."
           "  Upon failure, throw @code{ly-file-failed} key.")
{
  LY_ASSERT_TYPE (scm_is_string, name, 1);
  string file = ly_scm2string (name);
  char const *extensions[] = {"ly", "", 0};
```

```
  string file_name = global_path.find (file, extensions);

  /* By default, use base name of input file for output file name,
     write output to cwd; do not use root and directory parts of input
     file name.  */
  File_name out_file_name (file_name);

  // ...

  bool error = false;
  if ((file_name != "-") && file_name.empty ())
    {
      warning (_f ("cannot find file: `%s'", file));
      error = true;
    }
  else
    {
      Sources sources;
      sources.set_path (&global_path);

      string mapped_fn = map_file_name (file_name);
      basic_progress (_f ("Processing `%s'", mapped_fn.c_str ()));

      Lily_parser *parser = new Lily_parser (&sources);

      parser->parse_file (init, file_name, out_file);

      error = parser->error_level_;

      parser->clear ();
      parser->unprotect ();
    }

  /*
    outside the if-else to ensure cleanup fo Sources object,
  */
  if (error)
    /* TODO: pass renamed input file too.  */
    scm_throw (ly_symbol2scm ("ly-file-failed"),
               scm_list_1 (ly_string2scm (file_name)));

  return SCM_UNSPECIFIED;
}
```

Function `basic_progress()` is defined in `flower/include/warn.hh` and defined in `flower/warn.cc`:

```
/* Display a success message.  */
void
basic_progress (const string &s, const string &location)
{
```

```
  print_message (LOG_BASIC, location, s + "\n", true);
}
```

The same can be called from Scheme code with:

```
LY_DEFINE (ly_basic_progress, "ly:basic-progress",
           1, 0, 1, (SCM str, SCM rest),
           "A Scheme callable function to issue a basic progress message @var{str}."
           "  The message is formatted with @code{format} and @var{rest}.")
{
  LY_ASSERT_TYPE (scm_is_string, str, 1);
  str = scm_simple_format (SCM_BOOL_F, str, rest);
  basic_progress (ly_scm2string (str));
  return SCM_UNSPECIFIED;
}
```

9.4 gdb

`gdb` is the GNU debugger. By default, LilyPond is built with debug symbols included.

9.4.1 LilyPond source code

Consider the following example:

```
\version "2.19.33"

music =   {
  \tag #'noCues R1
  \tag #'Cues {\cueDuring #"cueNotes" #DOWN  R1 }    % 1
  d''8( e'' f'') r r2                                % 2
}

cueNotes = \relative c'' {
  s4. <g d'~>8 d' s4.    % 1
}

\addQuote "cueNotes" \cueNotes

\score { \new Staff {\cueNotes }  }

\score { \new Staff { \removeWithTag#'Cues \music } }

\score { \new Staff { \keepWithTag#'Cues \music }  }
```

9.4.2 Compilation log

When compiling it with version 2.19.33, a crash occurs (that is quite rare!) due to a segmentation fault:

```
Starting lilypond 2.19.33 [Untitled (5)]...
Processing `/var/folders/jc/xrpy67_x6_vcjfzpzds_9_6m0000gn/T/frescobaldi-mmxLK7/tmpfpLEl6/document.ly'
Parsing...
Interpreting music...
Interpreting music...
Preprocessing graphical objects...
Interpreting music...
Preprocessing graphical objects...
Interpreting music...
```

```
Exited with exit status 1.
```

This has been registered as issue 4718, see `http://sourceforge.net/p/testlilyissues/issues/4718/`.

9.4.3 Backtrace

Using `gdb` shows the program control flow up to the crash. Blank lines have been added in the trace for readability:

```
user@lilydev: ~ > gdb lilypond
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-64.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /usr/local/bin/lilypond...done.

(gdb) set args LilyPondCrashExample.ly

(gdb) run
Starting program: /usr/local/bin/lilypond LilyPondCrashExample.ly
gobject.pyc: gdb was not built with custom backtrace support, disabling.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Missing separate debuginfo for /lib64/libgraphite2.so.3
Try: yum --enablerepo='*debug*' install /usr/lib/debug/.build-id/7b/9de7ee95d41699040c799d36143b37906a43fc.debu
GNU LilyPond 2.19.33
Processing `LilyPondCrashExample.ly'
Parsing...
Interpreting music...
Interpreting music...
Preprocessing graphical objects...
Interpreting music...
Preprocessing graphical objects...
Interpreting music...

Program received signal SIGSEGV, Segmentation fault.

Prob::internal_get_property (this=this@entry=0x0, sym=0x7ffff3b7c840)
    at /home/user/lilypond-git/lily/prob.cc:152

152   SCM s = scm_sloppy_assq (sym, mutable_property_alist_);

Missing separate debuginfos, use: debuginfo-install expat-2.1.0-8.el7.x86_64 fontconfig-2.10.95-7.el7.x86_64 fr

(gdb) bt

#0  Prob::internal_get_property (this=this@entry=0x0, sym=0x7ffff3b7c840)
    at /home/user/lilypond-git/lily/prob.cc:152

#1  0x00000000005cf2e0 in Music::transpose (this=0x0, delta=...)
    at /home/user/lilypond-git/lily/music.cc:241

#2  0x00000000004ed190 in transpose_music_list (lst=lst@entry=0x7ffff09e58c0, rq=...)
    at /home/user/lilypond-git/lily/music-sequence.cc:33

#3  0x00000000005cf641 in transpose_mutable (alist=<optimized out>, delta=...)
    at /home/user/lilypond-git/lily/music.cc:228

#4  0x00000000005be380 in Quote_iterator::process (this=0xfdc6b0, m=...)
    at /home/user/lilypond-git/lily/quote-iterator.cc:281
```

```
#5  0x0000000000437c98 in Sequential_iterator::process (this=0xfc2750, until=...)
    at /home/user/lilypond-git/lily/sequential-iterator.cc:221

#6  0x0000000000437c98 in Sequential_iterator::process (this=0xfc0600, until=...)
    at /home/user/lilypond-git/lily/sequential-iterator.cc:221

#7  0x0000000000437c98 in Sequential_iterator::process (this=0xfbe470, until=...)
    at /home/user/lilypond-git/lily/sequential-iterator.cc:221

#8  0x0000000000437c98 in Sequential_iterator::process (this=0xfbbd80, until=...)
    at /home/user/lilypond-git/lily/sequential-iterator.cc:221

#9  0x000000000053f56c in Music_wrapper_iterator::process (this=<optimized out>, m=...)
    at /home/user/lilypond-git/lily/music-wrapper-iterator.cc:70

#10 0x00000000006b48ad in Global_context::run_iterator_on_me (this=this@entry=0xfb4d50,
    iter=iter@entry=0xfb5820) at /home/user/lilypond-git/lily/global-context.cc:169

#11 0x00000000006b2883 in ly_interpret_music_expression (mus=mus@entry=0x7ffff1123660,
    ctx=ctx@entry=0x7ffff0a13a80) at /home/user/lilypond-git/lily/global-context-scheme.cc:118

#12 0x00000000006b2c1e in ly_run_translator (mus=0x7ffff1123660,
    output_def=output_def@entry=0x7ffff0a4a670)
    at /home/user/lilypond-git/lily/global-context-scheme.cc:145

#13 0x00000000005cd754 in Score::book_rendering (this=this@entry=0xe1c050, layoutbook=0xc72cd0,
    default_def=default_def@entry=0xdb6d10) at /home/user/lilypond-git/lily/score.cc:141

#14 0x000000000051f008 in Book::process_score (this=this@entry=0xc6b510, s=s@entry=0x7ffff10ab710,
    output_paper_book=output_paper_book@entry=0xc72c60, layout=layout@entry=0xdb6d10)
    at /home/user/lilypond-git/lily/book.cc:225

#15 0x000000000051f2f9 in Book::process (this=this@entry=0xc6b510, default_paper=<optimized out>,
    default_layout=0xdb6d10, parent_part=parent_part@entry=0x0)
    at /home/user/lilypond-git/lily/book.cc:302

#16 0x000000000051f3c7 in Book::process (this=this@entry=0xc6b510, default_paper=<optimized out>,
    default_layout=<optimized out>) at /home/user/lilypond-git/lily/book.cc:196

#17 0x00000000005fdf58 in ly_book_process (book_smob=<optimized out>, default_paper=0x7ffff1340cc0,
    default_layout=0x7ffff14b3d10, output=0x7ffff3900900)
    at /home/user/lilypond-git/lily/book-scheme.cc:75

#18 0x00007ffff792491f in scm_dapply (proc=0x7ffff3c9a4f0, arg1=0x7ffff4643150, args=0x7ffff10ecaf0,
    args@entry=0x404) at eval.c:4930

#19 0x00007ffff792581b in deval (x=<optimized out>, env=<optimized out>) at eval.c:4378

#20 0x00007ffff792ea73 in scm_c_with_fluid (fluid=0x7ffff3c58040, value=0x7ffff10ef520,
    cproc=0x44c080 <catch_protected_eval_body(void*)>, cdata=0x7fffffffa0f0) at fluids.c:463

#21 0x000000000044c1f3 in ly_eval_scm (form=form@entry=0x7ffff10f00b0, i=..., safe=safe@entry=false,
    parser=parser@entry=0xc69870) at /home/user/lilypond-git/lily/parse-scm.cc:181

#22 0x0000000000709a07 in Lily_lexer::eval_scm (this=this@entry=0xc69ac0,
---Type <return> to continue, or q <return> to quit---
    readerdata=readerdata@entry=0x7ffff10f00b0, hi=..., extra_token=extra_token@entry=35 '#')
    at /home/user/lilypond-git/lily/lexer.ll:1081

#23 0x000000000071d7c9 in Lily_lexer::eval_scm_token (this=0xc69ac0, sval=0x7ffff10f00b0, w=...)
    at /home/user/lilypond-git/lily/include/lily-lexer.hh:61

#24 0x0000000000713519 in yyparse (parser=parser@entry=0xc69870, retval=retval@entry=0x7fffffffbf90)
    at /home/user/lilypond-git/lily/parser.yy:447
```

```
#25 0x0000000000071d754 in Lily_parser::do_yyparse_trampoline (parser=parser@entry=0xc69870)
    at /home/user/lilypond-git/lily/parser.yy:3866

#26 0x00007ffff792ea73 in scm_c_with_fluid (fluid=0x7ffff3c580c0, value=0x7ffff2040e20,
    cproc=0x71d730 <Lily_parser::do_yyparse_trampoline(void*)>, cdata=0xc69870) at fluids.c:463

#27 0x00000000006cbbe0 in Lily_parser::parse_file (this=this@entry=0xc69870, init="init.ly",
    name="LilyPondCrashExample.ly", out_name="LilyPondCrashExample")
    at /home/user/lilypond-git/lily/lily-parser.cc:123

#28 0x00000000005ec365 in ly_parse_file (name=<optimized out>)
    at /home/user/lilypond-git/lily/lily-parser-scheme.cc:121

#29 0x00007ffff7926aff in deval (x=<optimized out>, x@entry=0x7ffff21e2d30, env=<optimized out>,
    env@entry=0x7ffff2040eb0) at eval.c:4232

#30 0x00007ffff7924d37 in scm_dapply (proc=0x7ffff2040fc0, arg1=<optimized out>, args=0x7ffff2040eb0)
    at eval.c:5012

#31 0x00007ffff797b198 in scm_c_catch (tag=<optimized out>,
    body=body@entry=0x7ffff797ac50 <scm_body_thunk>, body_data=body_data@entry=0x7fffffffc760,
    handler=0x7ffff797ac60 <scm_handle_by_proc>, handler_data=handler_data@entry=0x7fffffffc758,
    pre_unwind_handler=0x0, pre_unwind_handler_data=pre_unwind_handler_data@entry=0x7fffffffc750)
    at throw.c:203

#32 0x00007ffff797b39e in scm_catch_with_pre_unwind_handler (key=<optimized out>,
    thunk=<optimized out>, handler=0x7ffff2040f30, pre_unwind_handler=0x204) at throw.c:587

#33 0x00007ffff792491f in scm_dapply (proc=0x7ffff3c9a4f0, arg1=0x7ffff468d8e0, args=0x7ffff2040f00,
    args@entry=0x404) at eval.c:4930

#34 0x00007ffff792581b in deval (x=<optimized out>, env=<optimized out>, env@entry=0x7ffff2041170)
    at eval.c:4378

#35 0x00007ffff79259c0 in deval (x=0x7ffff21e6c40, x@entry=0x7ffff21e7270, env=0x7ffff2041170,
    env@entry=0x7ffff2043750) at eval.c:3397

#36 0x00007ffff7924d37 in scm_dapply (proc=0x7ffff2043e00, arg1=<optimized out>, args=0x7ffff2043750)
    at eval.c:5012
#37 0x00007ffff27b50e8 in scm_srfi1_for_each (proc=0x7ffff2043c80, arg1=0x7ffff20427a0,
    args=<optimized out>) at srfi-1.c:1516
#38 0x00007ffff7925860 in deval (x=<optimized out>, env=<optimized out>, env@entry=0x7ffff20420f0)
    at eval.c:4509
#39 0x00007ffff79259c0 in deval (x=0x7ffff21e60f0, env=0x7ffff20420f0, env@entry=0x7ffff2042760)
    at eval.c:3397
#40 0x00007ffff7926747 in deval (x=0x7ffff2042320, x@entry=0x7ffff21ed440,
    env=env@entry=0x7ffff2042760) at eval.c:3648
#41 0x00007ffff7924d37 in scm_dapply (proc=0x7ffff21ece00, arg1=<optimized out>, args=0x7ffff2042760)
    at eval.c:5012
#42 0x00000000005c81cd in operator() (arg1=<optimized out>, this=<optimized out>)
    at /home/user/lilypond-git/lily/include/lily-modules.hh:73

#43 main_with_guile () at /home/user/lilypond-git/lily/main.cc:537

#44 0x00007ffff793f33f in invoke_main_func (body_data=0x7fffffffd430) at init.c:367
---Type <return> to continue, or q <return> to quit---

#45 0x00007ffff79179ca in c_body (d=d@entry=0x7fffffffd3d0) at continuations.c:349

#46 0x00007ffff797b198 in scm_c_catch (tag=tag@entry=0x104, body=body@entry=0x7ffff79179c0 <c_body>,
    body_data=body_data@entry=0x7fffffffd3d0, handler=handler@entry=0x7ffff79179e0 <c_handler>,
    handler_data=handler_data@entry=0x7fffffffd3d0, pre_unwind_handler=pre_unwind_handler@entry=
    0x7ffff797b790 <scm_handle_by_message_noexit>,
    pre_unwind_handler_data=pre_unwind_handler_data@entry=0x0) at throw.c:203
```

```
#47 0x00007ffff7917f53 in scm_i_with_continuation_barrier (body=body@entry=0x7ffff79179c0 <c_body>,
    body_data=body_data@entry=0x7fffffffd3d0, handler=handler@entry=0x7ffff79179e0 <c_handler>,
    handler_data=handler_data@entry=0x7fffffffd3d0, pre_unwind_handler=
    0x7ffff797b790 <scm_handle_by_message_noexit>,
    pre_unwind_handler_data=pre_unwind_handler_data@entry=0x0) at continuations.c:325

#48 0x00007ffff7917fe0 in scm_c_with_continuation_barrier (
    func=func@entry=0x7ffff793f320 <invoke_main_func>, data=data@entry=0x7fffffffd430)
    at continuations.c:367

#49 0x00007ffff79797c9 in scm_i_with_guile_and_parent (func=0x7ffff793f320 <invoke_main_func>,
    data=0x7fffffffd430, parent=<optimized out>) at threads.c:733

#50 0x00007ffff793f475 in scm_boot_guile (argc=<optimized out>, argv=<optimized out>,
    main_func=<optimized out>, closure=<optimized out>) at init.c:350

#51 0x0000000000041f890 in main (argc=2, argv=0x7fffffffd5a8, envp=<optimized out>)
    at /home/user/lilypond-git/lily/main.cc:829
(gdb)
```

9.4.4 Incriminated source file

Here is line 152 in file `lily/prob.cc` where the application crashed:

```
140
141  SCM
142  Prob::internal_get_property (SCM sym) const
143  {
144  #ifdef DEBUG
145    if (profile_property_accesses)
146      note_property_access (&prob_property_lookup_table, sym);
147  #endif
148
149    /*
150      TODO: type checking
151     */
152    SCM s = scm_sloppy_assq (sym, mutable_property_alist_);
153    if (scm_is_true (s))
154      return scm_cdr (s);
155
156    s = scm_sloppy_assq (sym, immutable_property_alist_);
157    return scm_is_false (s) ? SCM_EOL : scm_cdr (s);
158  }
159
```

10 Music interpretation

10.1 Root function

This function is `ly_interpret_music_expression()` in C++ and `ly:interpret-music-expression()` in Scheme. It is located in `lily/global-context-scheme.cc`:

```
LY_DEFINE (ly_interpret_music_expression, "ly:interpret-music-expression",
           2, 0, 0, (SCM mus, SCM ctx),
           "Interpret the music expression @var{mus} in the global context"
           " @var{ctx}.  The context is returned in its final state.")
{
  LY_ASSERT_SMOB (Music, mus, 1);
  LY_ASSERT_SMOB (Global_context, ctx, 2);

  Music *music = unsmob<Music> (mus);
  if (!music)
    {
      warning (_ ("no music found in score"));
      return SCM_BOOL_F;
    }

  Global_context *g = unsmob<Global_context> (ctx);

  Cpu_timer timer;

  message (_ ("Interpreting music..."));

  SCM protected_iter = Music_iterator::get_static_get_iterator (music);
  Music_iterator *iter = unsmob<Music_iterator> (protected_iter);

  iter->init_context (music, g);
  iter->construct_children ();

  if (!iter->ok ())
    {
      warning (_ ("no music found in score"));
      /* todo: should throw exception. */
      return SCM_BOOL_F;
    }

  g->run_iterator_on_me (iter);

  iter->quit ();
  scm_remember_upto_here_1 (protected_iter);

  send_stream_event (g, "Finish", 0, 0);

  debug_output (_f ("elapsed time: %.2f seconds", timer.read ()));

  return ctx;
```

```
}
```

Function `ly_interpret_music_expression()` is called from:

```
LY_DEFINE (ly_run_translator, "ly:run-translator",
           2, 1, 0, (SCM mus, SCM output_def),
           "Process @var{mus} according to @var{output-def}.  An"
           " interpretation context is set up, and @var{mus} is"
           " interpreted with it.  The context is returned in its"
           " final state.\n"
           "\n"
           "Optionally, this routine takes an object-key to"
           " to uniquely identify the score block containing it.")
{
  LY_ASSERT_SMOB (Music, mus, 1);
  LY_ASSERT_SMOB (Output_def, output_def, 2);

  SCM glob = ly_make_global_context (output_def);
  ly_make_global_translator (glob);
  ly_interpret_music_expression (mus, glob);
  return glob;
}
```

Function `ly:interpret-music-expression()` is called in `part-combiner.scm`:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define-public (recording-group-emulate music odef)
  "Interpret @var{music} according to @var{odef}, but store all events
in a chronological list, similar to the @code{Recording_group_engraver} in
LilyPond version 2.8 and earlier."
; ...
```

Function `ly_run_translator()` in turn is called from `lily/score-scheme.cc`:

```
LY_DEFINE (ly_score_embedded_format, "ly:score-embedded-format",
           2, 0, 0, (SCM score, SCM layout),
           "Run @var{score} through @var{layout} (an output definition)"
           " scaled to correct output-scale already, returning a list of"
           " layout-lines.")
{
// ...
```

It is also called in `lily/score.cc`:

```
/*
  Format score, return list of Music_output objects.

  LAYOUTBOOK should be scaled already.
*/
SCM
Score::book_rendering (Output_def *layoutbook,
                       Output_def *default_def)
{
// ...
```

Function `ly:run-translator()` is not called actually, but mentionned in a comment in `score-engraver.cc`.

Function `ly_score_embedded_format()` is not called actually, and `ly:score-embedded-format()` is called from `score-lines()` in `define-markup-commands.scm`:

```
(define-markup-list-command (score-lines layout props score)
  (ly:score?)
  "This is the same as the \\score markup but delivers its
systems as a list of lines.  Its score argument is entered in
braces like it would be for \\score."
  (let ((output (ly:score-embedded-format score layout)))

    (if (ly:music-output? output)
        (map
         (lambda (paper-system)
           ;; shift such that the refpoint of the bottom staff of
           ;; the first system is the baseline of the score
           (ly:stencil-translate-axis
            (paper-system-stencil paper-system)
            (- (car (paper-system-staff-extents paper-system)))
            Y))
         (vector->list (ly:paper-score-paper-systems output)))
        (begin
          (ly:warning (_"no systems found in \\score markup, does it have a \\lay-
out block?"))
          '())))
```

11 The libmusicxml2 library

`libmusicxml2` is a C++ library developped by Grame with the following copyright notice:

```
/*
  MusicXML Library
  Copyright (C) Grame 2006-2013

  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/.

  Grame Research Laboratory, 11, cours de Verdun Gensoul 69002 Lyon - France
  research@grame.fr
*/
```

This chapter presents the code structure of the library and the way it is used to translate MusicXML source files to LilyPond output files.

Contrary to the other chapters, all the access pathes to files in this chapter are relative to the folder created by cloning the `libmusicxml2` GIT repository, `${HOME}/libmusicxml-git` in this case.

11.1 A MusicXML example

We shall use the following minimal example, describing music made of one measure with 4 quater notes in `4/4` time.

It can be written this way in LilyPond syntax:

```
\version "2.19.30"

\relative {
  \clef "bass"
  \time 6/8
  \key f \major
  a4 bes8 ~ bes ( c b )
}
```

An equivalent version in MusicXML is:

```
user@lilydev: ~/libmusicxml-git/xmlSamples > cat MusicXMLSample.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 2.0 Partwise//EN"
                                "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.0">
  <identification>
    <encoding>
      <software>Frescobaldi 2.18.1</software>
      <encoding-date>2015-10-29</encoding-date>
    </encoding>
  </identification>
  <part-list>
    <score-part id="P1" />
  </part-list>
  <part id="P1">
    <measure number="1">
```

```
<attributes>
  <divisions>2</divisions>
  <key>
    <fifths>-1</fifths>
    <mode>major</mode>
  </key>
  <time>
    <beats>6</beats>
    <beat-type>8</beat-type>
  </time>
  <clef>
    <sign>F</sign>
    <line>4</line>
  </clef>
</attributes>
<note>
  <pitch>
    <step>A</step>
    <octave>3</octave>
  </pitch>
  <duration>2</duration>
  <type>quarter</type>
  <voice>1</voice>
</note>
<note>
  <pitch>
    <step>B</step>
    <alter>-1</alter>
    <octave>3</octave>
  </pitch>
  <duration>1</duration>
  <type>eighth</type>
  <voice>1</voice>
  <accidental>flat</accidental>
  <tie type="start" />
  <notations>
    <tied type="start" />
  </notations>
</note>
<note>
  <pitch>
    <step>B</step>
    <alter>-1</alter>
    <octave>3</octave>
  </pitch>
  <duration>1</duration>
  <type>eighth</type>
  <voice>1</voice>
  <accidental>flat</accidental>
  <tie type="stop" />
  <notations>
    <tied type="stop" />
```

```
            <slur number="1" type="start" />
          </notations>
        </note>
        <note>
          <pitch>
            <step>C</step>
            <octave>4</octave>
          </pitch>
          <duration>1</duration>
          <type>eighth</type>
          <voice>1</voice>
        </note>
        <note>
          <pitch>
            <step>B</step>
            <octave>3</octave>
          </pitch>
          <duration>1</duration>
          <type>eighth</type>
          <voice>1</voice>
          <notations>
            <slur number="1" type="stop" />
          </notations>
        </note>
      </measure>
    </part>
  </score-partwise>
  user@lilydev: ~/libmusicxml-git/xmlSamples >
```

Note that the `DOCTYPE` of this file is `score-partwise`. In this format, the bulk of music is represent as a list of parts, each containing a list of measures, as used basically in LilyPond.

Another MusicXML format named `score-timewise` exists, in which the music is represented as a list of measures, each containing a list of parts truncated to the corresponding measure. This is akin to `\parallelMusic` in LilyPond.

Is seems that virtually nobody to use the `score-timewise` format, so we don't support it.

11.2 Installation on LilyDev 4

Create a virtual machine with the LilyDev 4 ISO image, and a lamda user with sudo rights. In this chapter, the hostame is "lilydev" and the username is "user".

Then take the following actions in order:

1. Install the prerequisites:

   ```
   sudo apt-get install ssh
   sudo apt-get install cmake
   sudo apt-get install install-info
   sudo apt-get install gdb
   ```

2. Clone the library from the repository: `git clone git://github.com/dfober/libmusicxml libmusicxml-git`

3. Build the library:

   ```
   user@lilydev: ~/libmusicxml-git > cat Build_libmusicxml.bash
   #!/bin/bash
   ```

```
echo

cd cmake
pwd
echo

echo '--> cmake .'
echo
cmake .
echo

echo '--> make'
echo
make
echo

echo '--> sudo make install'
echo
sudo make install
echo

echo '--> find /usr -name "*libmusicxml*"'
echo
find /usr -name "*libmusicxml*"
echo

cd ../samples
pwd
echo

echo '--> make'
echo
make
echo

echo '--> ls -sal'
echo
ls -sal
echo

cd ../cmake
pwd
echo

echo '--> ls -sal'
echo
ls -sal
echo
```

11.3 Library version number

The version number is defined in `src/elements/versions.cpp` by:

```
int     versions::libVersion()        { return 210; }
const char* versions::libVersionStr()     { return "2.1.0"; }
```

11.4 Fundamental types

The library makes heavy use of so-called smart pointers, that encapsulates memory management of subclasses instances through reference counting. The implementation is in `src/lib/smartpointer.h`:

```
/*!
\brief the base class for smart pointers implementation

  Any object that want to support smart pointers should
  inherit from the smartable class which provides reference counting
  and automatic delete when the reference count drops to zero.
*/
class EXP smartable {
...
}
```

This allows for template class `SMARTP` to be implemented:

```
/*!
\brief the smart pointer implementation

  A smart pointer is in charge of maintaining the objects reference count
  by the way of pointers operators overloading. It supports class
  inheritance and conversion whenever possible.
\n  Instances of the SMARTP class are supposed to use \e smartable types (or at least
  objects that implements the \e addReference and \e removeReference
  methods in a consistent way).
*/
template<class T> class SMARTP {
  private:
    //! the actual pointer to the class
    T* fSmartPtr;
...
}
```

The file `elements/typedefs.h`, that is generated automatically from the MusicXML DTDs, contains the declarations of smart pointers mapped one to one to MusicXML markups, such as:

```
typedef SMARTP<musicxml<k_barline> >    S_barline;
```

11.5 MusicXML lexical analysis

It is generated by Flex from the specification in `src/parser/xml.l`, with the following initial comment:

```
/*
  Basic relaxed xml lexical definition.
  This is a basic definition of the lexical elements necessary to cover
  the MusicXML format. It is a simplified form based on the XML document
  grammar as defined in
  "XML in a nutshell - 2nd edition" E.R.Harold and W.S.Means,
```

```
  O'Reilly, June 2002, pp:366--371
*/
```

There's not much complexity in this file.

11.6 MusicXML syntactical analysis

The parser is generated by Bison from the specification in `src/parser/xml.y`. The initial comment containts:

```
/*
  Basic xml grammar definition
  This is a basic definition of the xml grammar necessary to cover
  the MusicXML format. It is a simplified form based on the XML document
  grammar as defined in
  "XML in a nutshell - 2nd edition" E.R.Harold and W.S.Means,
  O'Reilly, June 2002, pp:366--371
*/
```

The very simple grammar used by XML makes this file contents rather simple too.

11.7 MusicXML data representation

The presentation of the library in `doc/presentation/libmusicxml2.pdf` mentions the dilemma: how can one represent more that 300 element types in memory without an overwhelming number of classes?

The solution is to rely on a small set of foundation classes:

- `xmlattribute`

  This stores pairs of name/value pairs represented as strings:

```
/*!
\brief A generic xml attribute representation.

  An attribute is represented by its name and its value.
*/
//_____________________________________________________________________________
class EXP xmlattribute : public smartable {
  //! the attribute name
  std::string fName;
  //! the attribute value
  std::string   fValue;
```

- `xmlelement`

  Such and element is represented by its:

  - type
  - name
  - value
  - attributes

  leading to methods:

```
getType()
getName()
getValue()
getAttribute(name)
getAttributeValue(name)
```

- `ctree` in `lib/ctree.h`

  A tree of instances with iteration capabilities:

```
template <typename T> class EXP treeIterator : public std::iterator<std::input_iter
{
...
}

/*!
\brief a simple tree representation
*/
//_____________________________________________________________________________
template <typename T> class EXP ctree : virtual public smartable
{
  public:
    typedef SMARTP<T>                  treePtr;   ///< the node sub elements type
    typedef std::vector<treePtr>       branchs;   ///< the node sub elements contai
    typedef typename branchs::iterator literator; ///< the current level iterator t
    typedef treeIterator<treePtr>      iterator;  ///< the top -> bottom iterator t
...
}
```

The various MusicXML elements are described by classes using templates such as in `elements/types.h`:

```
template <int elt> class musicxml
```

The MusicXML DTDs are automatically analyzed to generate source code, types and constants. '-' are replaced with '_' in MusicXML elements or attribute names to comply to the C/C++ identifiers lexical definition.

A makefile and shell script `libmusicxml-git/src/elements/templates/elements` are used for DTDs analysis and the generation of templates, placed in `src/elements/templates`. The following filges are automatically generated by the DTDs analyser and should not be modified:

- `elements.h`
- `typedefs.h`
- `factory.cpp`

In this generation process from the DTDs:

```
<!ELEMENT part-name>
```

leads to:

- class: `S_part_name`
- constant: `k_part_name`

and:

```
<!ATTLIST measure
number CDATA #REQUIRED
...
>
```

allows for code such as:

```
measure->getAttributeValue("number")
measure->getAttributeIntValue("number",default)
```

11.8 The Visitors design pattern

The idea behind visitors is to have light-weight data elements stored in some data structure, and the processing of those elements moved elsewhere. A detailed presentation can be found at `https://sourcemaking.com/design_patterns/visitor`.

In our case, we proceed as follows: first define an abstract class `basevisitor`:

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

and a template class with two pure virtual methods `visitorStart()` and `visitorEnd()`, each taking a reference to an instance of the class used as the template parameter:

```
template<class C> class visitor : virtual public basevisitor
{
    public:
    virtual ~visitor() {}
    virtual void visitStart( C& elt ) {};
    virtual void visitEnd  ( C& elt ) {};
};
```

The reason for having two methods instead of just:

```
    virtual void visit( C& elt ) {};
```

as mentioned in the previous reference will be explained shortly. In fact, there could even be more if needed.

Now we provide a base class `visitable` with virtual methods `acceptIn()` and `acceptOut()` in files `src/visitors/visitable.h`:

```
 * \brief base class for visitable objects
 */
class visitable
{
  public:
    virtual ~visitable() {}
    virtual void acceptIn(basevisitor& visitor)  {}
    virtual void acceptOut(basevisitor& visitor) {}
};
```

Then we provide two virtual methods `acceptIn()` and `acceptOut()` to the base class of the data reprentation in files `src/elements/xml.h/.cpp`:

```
class EXP xmlelement : public ctree<xmlelement>, public visitable
{
  //! the element name
  std::string fName;
  //! the element value
  std::string fValue;
  //! list of the element attributes
  std::vector<Sxmlattribute> fAttributes;

    protected:
    //! the element type
    int fType;
```

```
    xmlelement() : fType(0) {}
    virtual ~xmlelement() {}

  public:
    typedef ctree<xmlelement>::iterator     iterator;

    static SMARTP<xmlelement> create();

    virtual void acceptIn(basevisitor& visitor);
    virtual void acceptOut(basevisitor& visitor);
```

When a tree element receives an `acceptIn()` or `acceptOut()` message, a `visitorStart()` or `visitorEnd()` message is sent to an adequate visitor, with the element as argument:

```
void xmlelement::acceptIn(basevisitor& v) {
  visitor<Sxmlelement>* p = dynamic_cast<visitor<Sxmlelement>*>(&v);
  if (p) {
    Sxmlelement xml = this;
    p->visitStart (xml);
  }
}

void xmlelement::acceptOut(basevisitor& v) {
  visitor<Sxmlelement>* p = dynamic_cast<visitor<Sxmlelement>*>(&v);
  if (p) {
    Sxmlelement xml = this;
    p->visitEnd (xml);
  }
}
```

There is thus a double so-called "dispatch", with two messages being sent when operating on the data is to be done.

We can now define a template subclass `musicxml` of `xmlelement` providing suitable concrete implementations of `acceptIn()` and `acceptOut()`. This is done in `src/elements/types.h`:

```
template <int elt> class musicxml : public xmlelement
{
    protected:
    musicxml() { fType = elt; }

  public:
    static SMARTP<musicxml<elt> > new_musicxml()
      { musicxml<elt>* o = new musicxml<elt>; assert(o!=0); return o; }
    static SMARTP<musicxml<elt> > new_musicxml(const std::vector<Sxmlelement>& elts)
      { musicxml<elt>* o = new musicxml<elt>(elts); assert(o!=0); return o; }

    virtual void acceptIn(basevisitor& v) {
      if (visitor<SMARTP<musicxml<elt> > >* p =
       dynamic_cast<visitor<SMARTP<musicxml<elt> > >*>(&v)) {
        SMARTP<musicxml<elt> > sptr = this;
        p->visitStart(sptr);
      }
      else xmlelement::acceptIn(v);
    }
```

```
    virtual void acceptOut(basevisitor& v) {
      if (visitor<SMARTP<musicxml<elt> > >* p =
       dynamic_cast<visitor<SMARTP<musicxml<elt> > >*>(&v)) {
        SMARTP<musicxml<elt> > sptr = this;
        p->visitEnd(sptr);
      }
      else xmlelement::acceptOut(v);
    }
};
```

A number of visitors are implemented in `src/visitors`:

```
basevisitor.h        metronomevisitor.cpp    timesignvisitor.h
clefvisitor.cpp      metronomevisitor.h      transposevisitor.cpp
clefvisitor.h        midicontextvisitor.cpp  transposevisitor.h
clonevisitor.cpp     midicontextvisitor.h    unrolled_clonevisitor.cpp
clonevisitor.h       notevisitor.cpp         unrolled_clonevisitor.h
keysignvisitor.cpp   notevisitor.h           visitable.h
keysignvisitor.h     partsummary.cpp         visitor.h
keyvisitor.cpp       partsummary.h           xmlvisitor.cpp
keyvisitor.h         timesignvisitor.cpp     xmlvisitor.h
```

11.9 Browsing MusicXML trees

In order to get the visitors executed on each element of the tree, we use the template tree browser class `browser` in `src/lib/browser.h`:

```
template <typename T> class browser {
  public:
    virtual ~browser() {}
    virtual void browse (T& t) = 0;
};
```

and its generic subclass `tree_browser` in `src/lib/tree_browser.h`:

```
template <typename T> class EXP tree_browser : public browser<T>
{
  protected:
    basevisitor*  fVisitor;

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

  public:
    typedef typename ctree<T>::treePtr treePtr;

    tree_browser(basevisitor* v) : fVisitor(v) {}
    virtual ~tree_browser() {}

    virtual void set (basevisitor* v) {  fVisitor = v; }
    virtual void browse (T& t) {
      enter(t);
      typename ctree<T>::literator iter;
      for (iter = t.lbegin(); iter != t.lend(); iter++)
        browse(**iter);
      leave(t);
```

```
    }
};
```

This is where the approach with dual `acceptIn()`/`acceptOut()` methods calling `visitStart()`/`visitEnd()` is justified: `enter()` causes `acceptIn()` to be called before the handling of the tree elements, and `leave()` causes `acceptOut()` to be called after such handling. Another call to `browse()` is done for each element of the tree in the iterator loop.

These methods pairs contribute to a kind of two-pass scheme : each element in the tree visited twice, with `visitorStart()` triggered the first time and `visitorEnd()` the second time. All these method take a reference to a smart pointer as argument.

This allows for the generation of bracketed LilyPond code such as '`{ ... }`': the opening bracket can be generated by `visitorStart()` and the closing one by `visitorEnd()`. One can also postpone the handling of data build by `visitorStart(S_someType& elt)` until `visitorEnd(S_someOtherType& elt)`, for example to change the order of elements in the generated LilyPond code.

Note that not all element types need to have both methods, making the approach a very flexible partial two-pass scheme.

Then class `xml_tree_browser` is declared in `src/elements/xml_tree_browser.h` as:

```
class EXP xml_tree_browser : public tree_browser<xmlelement>
{
  public:
    xml_tree_browser(basevisitor* v) : tree_browser<xmlelement>(v) {}
    virtual ~xml_tree_browser() {}
    virtual void browse (xmlelement& t);
};
```

with its `browse()` method defined in `src/elements/xml_tree_browser.cpp` as:

```
void xml_tree_browser::browse (xmlelement& t) {
  enter(t);
  ctree<xmlelement>::literator iter;
  for (iter = t.lbegin(); iter != t.lend(); iter++)
    browse(**iter);
  leave(t);
}
```

The `tree_browser::enter()` and `tree_browser::leave())` methods use:

```
void xmlelement::acceptIn(basevisitor& v) {
  visitor<Sxmlelement>* p = dynamic_cast<visitor<Sxmlelement>*>(&v);
  if (p) {
    Sxmlelement xml = this;
    p->visitStart (xml);
  }
}
```

and:

```
void xmlelement::acceptOut(basevisitor& v) {
  visitor<Sxmlelement>* p = dynamic_cast<visitor<Sxmlelement>*>(&v);
  if (p) {
    Sxmlelement xml = this;
    p->visitEnd (xml);
  }
}
```

respectively from `src/elements/xml.h/.cpp`.

11.10 A visitor example

One can find in `samples/countnotes.cpp` a visitor of `note` elements with only `visitStart()` implemented. At each visit along the traversal, the counter is incremented by 1:

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

    countnotes() : fCount(0) {}
    virtual ~countnotes() {}
    void visitStart( S_note& elt )    { fCount++; }
};
```

Reading a MusicXML file accessed throuth file descriptor `fd` is done as follows:

```
#define use_visitor

//...

static int read(FILE * fd)
{
  int count = 0;
  xmlreader r;
  SXMLFile file = r.read(fd);
  if (file) {
    Sxmlelement elt = file->elements();
    if (elt) {
#ifdef use_visitor
      countnotes v;
      xml_tree_browser browser(&v);
      browser.browse(*elt);
      count = v.fCount;
#else // use iterator
      predicate p;
      count = count_if(elt->begin(), elt->end(), p);
#endif
    }
  }
  else count = -1;
  return count;
}
```

11.11 LilyPond data representation

There are several classes implemented in `lilypond/lilypond.h/.cpp`. The basic ones is `lilypondelement`, that uses `lilypondparam` to store parameters as strings:

```
/*!
\brief A lilypondcmd parameter representation.

  A parameter is represented by its value.
*/
class EXP lilypondparam : public smartable {
```

```
  public:
        static SMARTP<lilypondparam> create(std::string value, bool quote=true);
        static SMARTP<lilypondparam> create(long value, bool quote=true);

    //! the parameter value
    void set (std::string value, bool quote=true);
    void set (long value, bool quote=true);
    std::string get () const          { return fValue; }
    bool   quote () const             { return fQuote; }

  protected:
    lilypondparam(std::string value, bool quote);
    lilypondparam(long value, bool quote);
    virtual ~lilypondparam ();

  private:
    std::string   fValue;
    bool  fQuote;
};

typedef SMARTP<lilypondparam>     Slilypondparam;
```

Class `lilypondelement` is used to store a name, start and end markers, a separator, enclosed elements and optional parameters:

```
/*!
\brief A generic lilypond element representation.

  An element is represented by its name and the
  list of its enclosed elements plus optional parameters.
*/
class EXP lilypondelement : public smartable {
  public:
        static SMARTP<lilypondelement> create(std::string name, std::string sep=" ");

    long add (Slilypondelement& elt);
    long add (Slilypondparam& param);
    long add (Slilypondparam param);
    void print (std::ostream& os);

    //! the element name
    void  setName (std::string name)      { fName = name; }
    std::string   getName () const        { return fName; }
    std::string   getStart () const       { return fStartList; }
    std::string   getEnd () const         { return fEndList; }
    std::string   getSep () const         { return fSep; }
        std::vector<Slilypondelement>& elements()   { return fElements; }
    const std::vector<Slilypondelement>& elements() const   { return fElements; }
        const std::vector<Slilypondparam>& parameters() const   { return fParams; }

    bool empty () const { return fElements.empty(); }

  protected:
```

```
    lilypondelement(std::string name, std::string sep=" ");
    virtual ~lilypondelement();

    std::string fName;
    //! the contained element start marker (default to empty)
    std::string fStartList;
    //! the contained element end marker (default to empty)
    std::string fEndList;
    //! the element separator (default to space)
    std::string fSep;
    //! list of the enclosed elements
    std::vector<Slilypondelement> fElements;
    //! list of optional parameters
    std::vector<Slilypondparam> fParams;
};

typedef SMARTP<lilypondelement>   Slilypondelement;
```

In particular, `lilypondelement::print()`:

```
    void print (std::ostream& os);
```

is where the generated LilyPond code is written to the output stream.

Then there are subclasses of `lilypondelement`:

```
class EXP lilypondnote : public lilypondelement { ... }
/*!
\brief Represents the current status of notes duration and octave.

    Octave and duration may be ommitted for lilypond notes. If so,
    they are infered from preceeding notes (or rests), within the same
    sequence or chord, or assumed to have standard values.
\n
  The object is defined as a multi-voices singleton: a single
    object is allocated for a specific voice and thus it will
  not operate correctly in case of parallel formatting
    operations on a given voice.
*/
class EXP lilypondnotestatus { ... }
/*!
\brief The lilypond sequence element
*/
class EXP lilypondseq : public lilypondelement { ... }
/*!
\brief The lilypond chord element
*/
class EXP lilypondchord : public lilypondelement { ... }
/*!
\brief A lilypond command representation.

  A command is represented by its name and optional parameters.
  A range command contains enclosed elements. //USER ???
*/
class EXP lilypondcmd : public lilypondelement { ... }
```

The corresponding smart pointer types are declared, such as:

```
typedef SMARTP<lilypondelement>   Slilypondelement;
```

11.12 xml2lilypond

The LilyPond code generation can be take a number of options represented by `translationSwitches` in file `interface/musicxml2lilypond.h`:

```
/*!
  \brief The lilypond code generation switches.

  A structure is used to avoid passing arguments one by one
  to the various methods that need them.
*/
typedef struct {
  bool fTrace;
  bool fGenerateAbsoluteCode;
  bool fGenerateComments;
  bool fGenerateBars;
  bool fGenerateStems;
  bool fGeneratePositions;
} translationSwitches;
```

The `main()` function is defined in `samples/xml2lilypond.cpp`. It uses functions `musicxmlfd2lilypond()` and `musicxmlfile2lilypond()`, declared and defined in files `src/interface/musicxml2lilypond..h/.cpp`. Both of these call function `xml2lilypond()`, declared and defined as `static` in files `src/interface/musicxml2lilypond..h/.cpp` too.

At this point, an instance of class `xml2lilypondvisitor` implemented in files `src/interface/xml2lilypondvisitor.h/.cpp` is instantiated and used to convert the MusicXML data to LilyPond code. This is done in a function private to `src/interface/musicxml2lilypond.cpp` :

```
//________________________________________________________________________
/*
 * The method that converts the file contents to LilyPond code
 * and  writes the result to the output stream
*/
static xmlErr xml2lilypond(
  SXMLFile& xmlfile,
  translationSwitches sw,
  ostream& out,
  const char* file)
{
  // build xmlelement tree from the file contents
  Sxmlelement st = xmlfile->elements();

  if (st) {
    // create an xml2lilypondvisitor
    xml2lilypondvisitor v(sw);

    // use the visitor to convert the xmlelement tree into a lilypondelement tree
    Slilypondelement    ly = v.convert(st);

    // output the general information about the conversion
```

```
      out << "%{" << std::endl;
      if (file) { // USER
        out <<
          "  LilyPond code converted from '" << file << "'" << std::endl;
      }
      else
        out <<
          "  LilyPond code converted from standard input" << std::endl;
      out <<
        "  using libmusicxml2" << //<< musicxmllibVersionStr() <<
        " and its embedded xml2lilypond converter " << //<< musicxml2lilypondVersionStr()
        std::endl <<
        "%}" <<
        std::endl << std::endl;

      // output the lilypondelement tree resulting from the conversion
      // thru lilypondelement::print()
      out << ly << endl;

      return kNoErr;
    }
    return kInvalidFile;
  }
```

The conversion is done in `src/lilypond/xml2lilypondvisitor.h/.cpp` by:

```
  Slilypondelement xml2lilypondvisitor::convert (const Sxmlelement& xml )
  {
    Slilypondelement ly;
    if (xml) {
      // create a browser on this xml2lilypondvisitor
      tree_browser<xmlelement> browser(this);
      // browse the xmlelement tree
      browser.browse(*xml);
      // the stack top contains the resulting lilypondelement tree
      ly = scoreStackTop();
    }
    return ly;
  }
```

11.13 Visitors in practise

In `src/lilypond/xml2lilypondvisitor.h/.cpp` there are some `visitStart()`/`visitEnd())` methods belonging to class `xml2lilypondvisitor`:

```
      virtual void visitStart( S_score_partwise& elt);
      virtual void visitStart( S_movement_title& elt);
      virtual void visitStart( S_creator& elt);
      virtual void visitStart( S_score_part& elt);
      virtual void visitStart( S_part_name& elt);
      virtual void visitStart( S_part& elt);
```

Others are in `src/lilypond/xmlpart2lilypond.h/.cpp`, belonging to class `xmlpart2lilypond`:

```
      virtual void visitStart( S_backup& elt);
```

```
virtual void visitStart( S_barline& elt);
virtual void visitStart( S_coda& elt);
virtual void visitStart( S_direction& elt);
virtual void visitStart( S_divisions& elt);
virtual void visitStart( S_dynamics& elt);
virtual void visitStart( S_forward& elt);
virtual void visitStart( S_measure& elt);
virtual void visitStart( S_note& elt);
virtual void visitStart( S_octave_shift& elt);
virtual void visitStart( S_part& elt);
virtual void visitStart( S_segno& elt);
virtual void visitStart( S_wedge& elt);

virtual void visitEnd  ( S_clef& elt);
virtual void visitEnd  ( S_direction& elt);
virtual void visitEnd  ( S_ending& elt);
virtual void visitEnd  ( S_key& elt);
virtual void visitEnd  ( S_measure& elt);
virtual void visitEnd  ( S_metronome& elt);
virtual void visitEnd  ( S_note& elt);
virtual void visitEnd  ( S_repeat& elt);
virtual void visitEnd  ( S_sound& elt);
virtual void visitEnd  ( S_time& elt);
```

Appendix A Index

This index lists the terms and identifiers contained in this document, with links to those sections of the manual which describe or discuss that topic.

B

BOM 16

F

flower 4, 26

I

int? 9

L

lexer.ll 16
ly:music? 9

M

main_with_guile 26
`Manuals` 1

P

parser.yy 21
python-ly 2

S

scm_boot_guile 26
smob 9
string? 9

T

tail function calls 7
trampoline 7

U

unsmob() 9

Y

yy_pop_state() 16
yy_push_state() 16
yylex() 21