python-ly Documentation

Release 0.9.5

Wilbert Berendsen

Sep 22, 2017

Contents

This package provides a Python library *ly* containing various Python modules to parse, manipulate or create documents in LilyPond format. A command line program *ly* is also provided that can be used to do various manipulations with LilyPond files.

The LilyPond format is a plain text input format that is used by the GNU music typesetter LilyPond (www.lilypond.org).

The python-ly package is Free Software, licensed under the GPL. This package is written by the Frescobaldi developers and is used extensively by the Frescobaldi project. The main author is Wilbert Berendsen.

Download from: https://pypi.python.org/pypi/python-ly
Development homepage: https://github.com/wbsoft/python-ly

This documentation describes python-ly 0.9.5.

Contents:

CHAPTER 1

The `ly` command

Usage:

```
ly [options] commands file, ...
```

A tool for manipulating LilyPond source files

Options

-v, --version show version number and exit

-h, --help show this help text and exit

-i, --in-place overwrite input files

-o, --output NAME output file name

-e, --encoding ENC (input) encoding (default UTF-8)

--output-encoding ENC output encoding (default to input encoding)

-l, --language NAME default pitch name language (default to “nederlands”)

-d <variable=value> set a variable

The special option −− considers the remaining arguments to be file names.

Arguments

The command is one argument with semicolon-separated commands. In most cases you’ll quote the command so that it is seen as one argument.

You can specify more than one LilyPond file. If you want to process many files and write the results of the operations on each file to a separate output file, you can use two special characters in the output filename: a ‘*’ will be replaced with the full path name of the current input file (without extension), and a ‘?’ will be replaced with the input filename

(without path and extension). If you don't want to have '*' or '?' replaced in the output filename, you can set `-d replace-pattern=false`.

If you don't specify input or output filenames, standard input is read and standard output is written to.

Commands

Informative commands that write information to standard output and do not change the file:

mode print the mode (guessing if not given) of the document

version print the LilyPond version, if set in the document

language print the pitch name language, if set in the document

Commands that change the file:

indent re-indent the file

reformat reformat the file

translate <language> translate the pitch names to the language

transpose <from> <to> transpose the file like LilyPond would do, pitches are given in the 'nederlands' language

abs2rel convert absolute music to relative

rel2abs convert relative music to absolute

simplify-accidentals replace notes with accidentals as much as possible with their natural neighbors

write [filename] write the file to the given filename or the output variable. If the last command was an editing command, write is automatically called.

Commands that export the file to another format:

musicxml [filename] export to MusicXML (in development, far from complete)

highlight [filename] export the document as syntax colored HTML

hl [filename] alias for highlight

Between commands, you can set or unset a variable using:

variable=value set a variable to value. Special values are true, false, which are interpreted as boolean values, or digits, which will be interpreted as integer values.

variable= unset a variable

Variables

The following variables can be set to influence the behaviour of commands. If there is a default value, it is written between brackets:

mode mode of the file to read (default automatic) can be one of: lilypond, scheme, latex, html, docbook, texinfo.

output [-] the output filename (also set by -o argument)

encoding [UTF-8] encoding to read (also set by -e argument)

default-language [nederlands] the pitch names language to use by default, when not specified otherwise in the document

output-encoding encoding to write (defaults to `encoding`, also set by the `--output-encoding` argument)

in-place [false] whether to overwrite input files (same as `-i`)

backup-suffix [~] suffix to use when editing files in-place, if set, backs up the original file before overwriting it

replace-pattern [true] whether to replace '*' and '?' in the output filename.

rel-startpitch [true] whether to write relative music with a startpitch

rel-absolute whether to assume that the first pitch in a relative expression without specified startpitch is absolute. If `false`, it is assumed to be relative to `c'`. If `true`, it is assumed to be absolute (in fact relative to `f`. If not set, this variable defaults to `true` only when the LilyPond version in the document >= 2.18.

indent-tabs [false] whether to use tabs for indent

indent-width [2] how many spaces for each indent level (if not using tabs)

full-html [True] if set to True a full document with syntax-highlighted HTML will be exported, otherwise only the bare content wrapped in an element configured by the `wrapper-` variables.

stylesheet filename to reference as an external stylesheet for syntax-highlighted HTML. This filename is literally used in the `<link rel="stylesheet">` tag.

inline-style [false] whether to use inline style attributes for syntax-highlighted HTML. By default a css stylesheet is embedded.

number-lines [false] whether to add line numbers when creating syntax-highlighted HTML.

wrapper-tag [pre] which tag syntax highlighted HTML will be wrapped in. Possible values: `div`, `pre`, `id` and `code`

wrapper-attribute [class] attribute used for the wrapper tag. Possible values: `id` and `class`.

document-id [lilypond] name applied to the wrapper-attribute. If the three last options use their default settings the highlighted HTML elements are wrapped in an element `<pre class="lilypond"></pre>`

linenumbers-id [linenumbers] if linenumbers are exported this is the name used for the `<td>` elements

These variables influence the output of information commands:

with-filename prints the filename next to information like version, etc. This is `true` by default if there is more than one file specified.

Examples

Here is an example to re-indent and transpose a LilyPond file:

```
ly "indent; transpose c d" -o output.ly file.ly
```

Examples using the '*' in the output file name:

```
ly "transpose c d" *.ly -o '*-transposed.ly'
ly highlight *.ly -o 'html/?.html'
```

CHAPTER 2

The `ly-server` command

Usage:

```
ly-server [options]
```

An HTTP server for manipulating LilyPond input code

Server Options

-v, --version show version number and exit

-h, --help show this help text and exit

-p, --port PORT port server listens to (default 5432)

-t, --timeout TIME If set, server shuts down automatically after -t seconds of inactivity
NOT IMPLEMENTED YET!

Command Options

-e, --encoding ENC (input) encoding (default UTF-8)

--output-encoding ENC output encoding (default to input encoding)

-l, --language NAME default pitch name language (default to “nederlands”)

-d <variable=value> set a variable (see below)

Command options define defaults for the execution of commands triggered by HTTP requests. These options can be overridden by the individual HTTP request.

HTTP Requests

GET Requests

Some GET requests will be implemented later to retrieve values or change some server settings based on the URL path.

POST Requests

The server accepts POST requests without (currently) making use of the URL path. As the request body it expects a single JSON string with the following elements:

`commands` (mandatory) An array with one or more commands to be executed subsequently. Each entry contains:

`command` (mandatory): A name for the command. It has to be one out of the list of available commands below.

`args` (optional): If a command requires arguments (e.g. the `transpose` command) they are given as a single string value.

`variables` (optional): A dictionary of variable assignments. Keys have to be from the list below, and proper value types are checked. If one or more variables are given they will be set *before* the command is executed. A variable may be modified again before the execution of the next command but it is not reset automatically. Giving a variable with a value of '' unsets the variable.

`options` (optional) A dictionary of option assignments. Keys have to be from the above list of Command Options, taking the long name without the leading hyphens, e.g. `{ "encoding": "UTF-16" }`. If an option is given here it overrides the default option given on the command line, but only for the current command.

`data` (mandatory) A single string containing the LilyPond input document.

The server will try to construct a series of commands from the request, and if anything is wrong with it send a "Bad Request" message with HTTP response code 400.

Response Object

If the commands execute successfully the response body will contain a serialized JSON object with the following elements:

`info` An array of entries with the result of "info" commands (see below). Each entry has a `command` and an `info` field.

`doc` An object with two fields:

`content` A string with the content of the document with all "edit" commands applied consecutively. (If no edit commands have been specified this contains the original input.

`commands` An array with the names of the commands that have been applied.

`exports` An array with entries for each applied "export" command. Each entry has the following fields:

`doc` A string with the content of the converted/exported document

`command` The name of the applied command

Commands

There are three types of commands whose results are handled independently:

- "info" commands retrieve metadata from the input document
- "edit" commands modify the document, subsequent edit commands cascade the modifications
- "export" commands that convert the input to another format. Subsequent commands are not affected by the result of export commands.

Informative commands that return information and do not change the file:

mode print the mode (guessing if not given) of the document

version print the LilyPond version, if set in the document

language print the pitch name language, if set in the document

Commands that modify the input:

indent re-indent the file

reformat reformat the file

translate <language> translate the pitch names to the language

transpose <from> <to> transpose the file like LilyPond would do, pitches are given in the 'nederlands' language

abs2rel convert absolute music to relative

rel2abs convert relative music to absolute

Commands that convert the input to another format:

musicxml convert to MusicXML (in development, far from complete)

highlight export the document as syntax colored HTML

Variables

The following variables can be set to influence the behaviour of commands. If there is a default value, it is written between brackets:

mode mode of the input to read (default automatic) can be one of: lilypond, scheme, latex, html, docbook, texinfo.

encoding [UTF-8] encoding to read (also set by -e argument)

default-language [nederlands] the pitch names language to use by default, when not specified otherwise in the document

output-encoding encoding to write (defaults to `encoding`, also set by the `--output-encoding` argument)

indent-tabs [false] whether to use tabs for indent

indent-width [2] how many spaces for each indent level (if not using tabs)

full-html [True] if set to True a full document with syntax-highlighted HTML will be exported, otherwise only the bare content wrapped in an element configured by the `wrapper-` variables.

stylesheet filename to reference as an external stylesheet for syntax-highlighted HTML. This filename is literally used in the `<link rel="stylesheet">` tag.

inline-style [false] whether to use inline style attributes for syntax-highlighted HTML. By default a css stylesheet is embedded.

number-lines [false] whether to add line numbers when creating syntax-highlighted HTML.

wrapper-tag [pre] which tag syntax highlighted HTML will be wrapped in. Possible values: `div`, `pre`, `id` and `code`

wrapper-attribute [class] attribute used for the wrapper tag. Possible values: `id` and `class`.

document-id [lilypond] name applied to the wrapper-attribute. If the three last options use their default settings the highlighted HTML elements are wrapped in an element `<pre class="lilypond"></pre>`

linenumbers-id [linenumbers] if linenumbers are exported this is the name used for the `<td>` elements

Examples

Here is the basic invocation, listening on port 5432:

```
ly-server
```

Specifying port and a timeout:

```
ly-server -p 4000 -t 5000
```

Sample Requests

The simplest request, just applying one edit command. In this case the result will be in `result['doc']['content']`:

```
{
    'commands': [
        {
            'command': 'indent'
        }
    ],
    'data' : "\relative c' { c ( d e f ) }"
}
```

Another simple request, this time applying an "info" command. The result will be in `result['info']`, containing `lilypond` in the `info` field and `mode` in the `command` field:

```
{
    'commands': [
        {
            'command': 'mode'
        }
    ],
    'data' : "\relative c' { c ( d e f ) }"
}
```

And a more complex example. This will first transpose the document and then convert the transposed version independently to highlighted HTML and MusicXML. Additionally it will retrieve the mode. This time the result will be in

all three places: the transposed document in `doc.content`, the mode in `info.info`, and HTML and MusicXML in `exports[0].doc` and `exports[1].doc`.:

```
{
    'commands': [
        {
            'command': 'transpose',
            'args': 'c d'
        },
        {
            'command': 'highlight',
            'variables': { 'full-html': 'false' }
        },
        { 'command': 'musicxml' },
        { 'command': 'mode' },
    ],
    'options': {
        'language': "deutsch"
    },
    'data': "\relative c' { c4 ( d e ) }"
}
```

CHAPTER 3

The `ly` package

Module contents

A package of modules dealing with LilyPond and the LilyPond format.

The ly module supports both Python2 and Python3. This is a short description of some modules:

- ly.slexer: generic tools to build parsers using regular expressions
- ly.node: a generic list-like node object to build tree structures with
- ly.document: a tokenized text document (LilyPond file)
- ly.docinfo: harvests and caches various information from a LilyPond document
- ly.lex: a parser for LilyPond, Scheme, and other formats, using slexer
- ly.music: a tree structure of the contents of a document
- ly.pitch: functions for translating, transposing etc
- ly.rhythm: functions dealing with rhythm
- ly.indent: indent LilyPond text
- ly.reformat: format LilyPond text
- ly.dom: (deprecated) tree structure to build LilyPond text from
- ly.words: words for highlighting and autocompletion
- ly.data: layout objects, properties, interfaces, font glyphs etc extracted from LilyPond
- ly.cli: the implementation of the command-line 'ly' script

A LilyPond document (source file) is held by a ly.document.Document. The Document class automatically parses and tokenizes the text, also when its contents are changed.

A music tree can be built from a document using the ly.music module. In the near future, music trees can be built from scratch and also generate LilyPond output from scratch. At that moment, ly.dom is deprecated.

The functions in ly.pitch, such as transpose and translate currently access the tokens in the document, but in the future they will work on the music tree.

Subpackages

ly.lex package

Module contents

This module is built on top of slexer and can parse LilyPond input and other formats.

The base functionality is delegated to modules with an underscore in this package. The modules describing parsing modes (filetypes) are the files without underscore.

Currently available are modes for lilypond, latex, html, texinfo, scheme, and docbook.

The 'underscored' modules should not be imported in application code. What is needed from them is available here, in the ly.lex namespace.

If you add new files for parsing other file types, you should add them in _mode.py. The _token.py module contains base Token types and Token mixin classes.

The State, Parser, FallthroughParser and Fridge classes from slexer are all slightly extended here,

Usage:

```
>>> import ly.lex
>>> txt = r"\relative c' { c d e f-^ g }"
>>> s = ly.lex.state("lilypond")
>>> for t in s.tokens(txt):
...     print(t, t.__class__.__name__)
\relative Command
  Space
c Name
' Unparsed
  Space
{ SequentialStart
  Space
c Note
  Space
d Note
  Space
e Note
  Space
f Note
- Direction
^ ScriptAbbreviation
  Space
g Note
  Space
} SequentialEnd
```

A State() is used to parse text. The text is given to the tokens() method, that returns an iterator iterating over Token instances as they are found. Each token has a 'pos' and an 'end' attribute describing its position in the original string.

While iterating over the tokens(), the State maintains information about what kind of text is parsed. (So don't iterate over more than one call to tokens() of the same State object at the same time.)

Use ly.lex.state("name") to get a state for a specific mode to start parsing with. If you don't know the type of text, you can use ly.lex.guessState(text), where text is the text you want to parse. A quick heuristic is then used to determine the type of the text.

See for more information the documentation of the slexer module.

class ly.lex.**State**(*initialParserClass*)
Bases: *ly.slexer.State*

endArgument()
Decrease argcount and leave the parser if it would reach 0.

mode()
Returns the mode attribute of the first parser (from current parser) that has it.

class ly.lex.**Parser**(*argcount=None*)
Bases: *ly.slexer.Parser*

argcount = 0

default
alias of *Unparsed*

freeze()

mode = None

re_flags = 40

class ly.lex.**FallthroughParser**(*argcount=None*)
Bases: *ly.lex.Parser*, *ly.slexer.FallthroughParser*

class ly.lex.**Fridge**(*stateClass=<class 'ly.lex.State'>*)
Bases: *ly.slexer.Fridge*

ly.lex.**guessMode**(*text*)
Tries to guess the type of the input text, using a quite fast heuristic.

Returns one of the strings also present as key in the modes dictionary.

ly.lex.**state**(*mode*)
Returns a State instance for the given mode.

ly.lex.**guessState**(*text*)
Returns a State instance, guessing the type of text.

class ly.lex.**Token**
Bases: *ly.slexer.Token*

class ly.lex.**Unparsed**
Bases: ly.lex._token.Token

Represents an unparsed piece of input text.

class ly.lex.**Space**
Bases: ly.lex._token.Token

A token containing whitespace.

rx = u'\\s+'

class ly.lex.**Newline**
Bases: ly.lex._token.Space

A token that is a single newline.

rx = u'\\n'

class ly.lex.**Comment**
Bases: ly.lex._token.Token

Base class for tokens that belong to a comment.

class ly.lex.**LineComment**
Bases: ly.lex._token.Comment

Base class for items that are a whole line comment.

class ly.lex.**BlockComment**
Bases: ly.lex._token.Comment

Base class for tokens that belong to a block/multiline comment.

class ly.lex.**BlockCommentStart**
Bases: ly.lex._token.BlockComment

Base class for tokens that start a block/multiline comment.

class ly.lex.**BlockCommentEnd**
Bases: ly.lex._token.BlockComment

Base class for tokens that end a block/multiline comment.

class ly.lex.**String**
Bases: ly.lex._token.Token

Base class for tokens that belong to a quote-delimited string.

class ly.lex.**StringStart**
Bases: ly.lex._token.String

Base class for tokens that start a quote-delimited string.

class ly.lex.**StringEnd**
Bases: ly.lex._token.String

Base class for tokens that end a quote-delimited string.

class ly.lex.**Character**
Bases: ly.lex._token.Token

Base class for tokens that are an (escaped) character.

class ly.lex.**Numeric**
Bases: ly.lex._token.Token

Base class for tokens that are a numerical value.

class ly.lex.**Error**
Bases: ly.lex._token.Token

Base class for tokens that represent erroneous input.

class ly.lex.**MatchStart**
Bases: object

Mixin class for tokens that have a matching token forward in the text.

The matchname attribute should give a unique name.

matchname = u'‘

class ly.lex.**MatchEnd**
Bases: object

Mixin class for tokens that have a matching token backward in the text.

The matchname attribute should give a unique name.

matchname = u'‘

class ly.lex.**Indent**
Bases: object

Mixin class for tokens that have the text on the next line indent more.

class ly.lex.**Dedent**
Bases: object

Mixin class for tokens that have the text on the next line indent less.

Submodules

ly.lex.docbook module

Parses and tokenizes DocBook input, recognizing LilyPond in DocBook.

class ly.lex.docbook.**ParseDocBook**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex._token.Space'>,)

mode = u'docbook'

pattern = <_sre.SRE_Pattern object>

ly.lex.html module

Parses and tokenizes HTML input, recognizing LilyPond in HTML.

class ly.lex.html.**AttrName**
Bases: ly.lex._token.Token

rx = u'\\w+([-_:]\\w+)?'

class ly.lex.html.**Comment**
Bases: ly.lex._token.Comment

class ly.lex.html.**CommentEnd**
Bases: *ly.lex.html.Comment*, ly.lex._token.Leaver, ly.lex._token.BlockCommentEnd

rx = u'–>'

class ly.lex.html.**CommentStart**
Bases: *ly.lex.html.Comment*, ly.lex._token.BlockCommentStart

rx = u'<!–‘

update_state(*state*)

class ly.lex.html.**EntityRef**
Bases: ly.lex._token.Character

rx = u'\\&(#\\d+|#[xX][0-9A-Fa-f]+|[A-Za-z_:][\\w.:_-]*);'

class ly.lex.html.**EqualSign**
Bases: ly.lex._token.Token

rx = u'='

update_state(*state*)

class ly.lex.html.**LilyPondCloseTag**
Bases: *ly.lex.html.LilyPondTag*, ly.lex._token.Leaver

rx = u'</lilypond>'

class ly.lex.html.**LilyPondFileTag**
Bases: *ly.lex.html.LilyPondTag*

rx = u'</?lilypondfile\\b'

update_state(*state*)

class ly.lex.html.**LilyPondFileTagEnd**
Bases: *ly.lex.html.LilyPondTag*, ly.lex._token.Leaver

rx = u'/?>'

class ly.lex.html.**LilyPondInlineTag**
Bases: *ly.lex.html.LilyPondTag*

rx = u'<lilypond\\b'

update_state(*state*)

class ly.lex.html.**LilyPondInlineTagEnd**
Bases: *ly.lex.html.LilyPondTag*, ly.lex._token.Leaver

rx = u'/?>'

class ly.lex.html.**LilyPondTag**
Bases: *ly.lex.html.Tag*

class ly.lex.html.**LilyPondTagEnd**
Bases: *ly.lex.html.LilyPondTag*

rx = u'>'

update_state(*state*)

class ly.lex.html.**LilyPondVersionTag**
Bases: *ly.lex.html.LilyPondTag*

rx = u'<lilypondversion/?>'

class ly.lex.html.**ParseAttr**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.html.TagEnd'>, <class 'ly.lex.html.AttrName'>, <class 'ly.lex.html.E

pattern = <_sre.SRE_Pattern object at 0x25ac310>

class ly.lex.html.**ParseComment**(*argcount=None*)
Bases: *ly.lex.Parser*

default
alias of *Comment*

items = (<class 'ly.lex.html.CommentEnd'>,)

pattern = <_sre.SRE_Pattern object>

class ly.lex.html.**ParseHTML**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.html.LilyPondVersionTag'>, <class 'ly.lex.html.LilyPondFileTag'>, <

mode = u'html'

pattern = <_sre.SRE_Pattern object at 0x3351490>

class ly.lex.html.**ParseLilyPond**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseGlobal*

items = (<class 'ly.lex.html.LilyPondCloseTag'>, <class 'ly.lex.lilypond.Book'>, <class 'ly.lex.lilypond.BookPart'>, <class

pattern = <_sre.SRE_Pattern object at 0x335a080>

class ly.lex.html.**ParseLilyPondAttr**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.html.AttrName'>, <class 'ly.lex.html.EqualSign'>, <class 'ly.lex.html

pattern = <_sre.SRE_Pattern object at 0x25ae980>

class ly.lex.html.**ParseLilyPondFileOptions**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.html.AttrName'>, <class 'ly.lex.html.EqualSign'>, <class 'ly.lex.html

pattern = <_sre.SRE_Pattern object at 0x3491750>

class ly.lex.html.**ParseLilyPondInline**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseMusic*

items = (<class 'ly.lex.html.LilyPondInlineTagEnd'>, <class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommen

pattern = <_sre.SRE_Pattern object at 0x2f69490>

class ly.lex.html.**ParseStringDQ**(*argcount=None*)
Bases: *ly.lex.Parser*

default
alias of *String*

items = (<class 'ly.lex.html.StringDQEnd'>, <class 'ly.lex.html.EntityRef'>)

pattern = <_sre.SRE_Pattern object at 0x2c73220>

class ly.lex.html.**ParseStringSQ**(*argcount=None*)
Bases: *ly.lex.Parser*

default
alias of *String*

items = (<class 'ly.lex.html.StringSQEnd'>, <class 'ly.lex.html.EntityRef'>)

pattern = <_sre.SRE_Pattern object at 0x283fa80>

class ly.lex.html.**ParseValue**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

Finds a value or drops back.

fallthrough(*state*)

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.html.Value'>)

pattern = <_sre.SRE_Pattern object>

class ly.lex.html.**SemiColon**
Bases: ly.lex._token.Token

rx = u':'

update_state(*state*)

class ly.lex.html.**String**
Bases: ly.lex._token.String

class ly.lex.html.**StringDQEnd**
Bases: *ly.lex.html.String*, ly.lex._token.StringEnd, ly.lex._token.Leaver

rx = u'""'

class ly.lex.html.**StringDQStart**
Bases: *ly.lex.html.String*, ly.lex._token.StringStart

rx = u'"'

update_state(*state*)

class ly.lex.html.**StringSQEnd**
Bases: *ly.lex.html.String*, ly.lex._token.StringEnd, ly.lex._token.Leaver

rx = u''''

class ly.lex.html.**StringSQStart**
Bases: *ly.lex.html.String*, ly.lex._token.StringStart

rx = u''''

update_state(*state*)

class ly.lex.html.**Tag**
Bases: ly.lex._token.Token

class ly.lex.html.**TagEnd**
Bases: *ly.lex.html.Tag*, ly.lex._token.Leaver

rx = u'/?>'

class ly.lex.html.**TagStart**
Bases: *ly.lex.html.Tag*

rx = u'</?\\w[-_:\\w]*\\b'

update_state(*state*)

class ly.lex.html.**Value**
Bases: ly.lex._token.Leaver

rx = u'\\w+'

ly.lex.latex module

Parses and tokenizes LaTeX input, recognizing LilyPond in LaTeX.

class ly.lex.latex.**ParseLaTeX**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex._token.Space'>,)

mode = u'latex'

pattern = <_sre.SRE_Pattern object>

ly.lex.lilypond module

Parses and tokenizes LilyPond input.

class ly.lex.lilypond.**Accidental**
Bases: ly.lex._token.Token

class ly.lex.lilypond.**AccidentalCautionary**
Bases: *ly.lex.lilypond.Accidental*

rx = u'\\?'

class ly.lex.lilypond.**AccidentalReminder**
Bases: *ly.lex.lilypond.Accidental*

rx = u'!'

class ly.lex.lilypond.**AccidentalStyle**
Bases: *ly.lex.lilypond.Command*

rx = u'\\\\accidentalStyle\\b'

update_state(*state*)

class ly.lex.lilypond.**AccidentalStyleSpecifier**
Bases: *ly.lex.lilypond.Specifier*

rx = u'\\b(default|voice|modern|modern-cautionary|modern-voice|modern-voice-cautionary|piano|piano-cautionary|neo-n

class ly.lex.lilypond.**AlterBroken**
Bases: *ly.lex.lilypond.Command*

rx = u'\\\\alterBroken\\b'

update_state(*state*)

class ly.lex.lilypond.**Articulation**
Bases: ly.lex._token.Token

Base class for articulation things.

class ly.lex.lilypond.**ArticulationCommand**
Bases: *ly.lex.lilypond.Articulation*, *ly.lex.lilypond.IdentifierRef*

classmethod test_match(*match*)

class ly.lex.lilypond.**BackSlashedContextName**
Bases: *ly.lex.lilypond.ContextName*

rx = u'\\\\(ChoirStaff|ChordNames|CueVoice|Devnull|DrumStaff|DrumVoice|Dynamics|FiguredBass|FretBoards|Global|G

class ly.lex.lilypond.**Beam**
Bases: ly.lex._token.Token

class ly.lex.lilypond.**BeamEnd**
Bases: *ly.lex.lilypond.Beam*, ly.lex._token.MatchEnd

matchname = u'beam'

rx = u'\\]'

class ly.lex.lilypond.**BeamStart**
Bases: *ly.lex.lilypond.Beam*, ly.lex._token.MatchStart

matchname = u'beam'

rx = u'\\['

class ly.lex.lilypond.**BlockComment**
Bases: *ly.lex.lilypond.Comment*, ly.lex._token.BlockComment

class ly.lex.lilypond.**BlockCommentEnd**
Bases: *ly.lex.lilypond.Comment*, ly.lex._token.BlockCommentEnd, ly.lex._token.Leaver

rx = u'%}'

class ly.lex.lilypond.**BlockCommentStart**
Bases: *ly.lex.lilypond.Comment*, ly.lex._token.BlockCommentStart

rx = u'%{'

update_state(*state*)

class ly.lex.lilypond.**Book**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\book\\b'

update_state(*state*)

class ly.lex.lilypond.**BookPart**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\bookpart\\b'

update_state(*state*)

class ly.lex.lilypond.**Change**
Bases: *ly.lex.lilypond.Translator*

rx = u'\\\\change\\b'

class ly.lex.lilypond.**Chord**
Bases: ly.lex._token.Token

Base class for Chord delimiters.

class ly.lex.lilypond.**ChordEnd**
Bases: *ly.lex.lilypond.Chord*, ly.lex._token.Leaver

rx = u'>'

class ly.lex.lilypond.**ChordItem**
Bases: ly.lex._token.Token

Base class for chordmode items.

class ly.lex.lilypond.**ChordMode**
Bases: *ly.lex.lilypond.InputMode*

rx = u'\\\\(chords|chordmode)\\b'

update_state(*state*)

class ly.lex.lilypond.**ChordModifier**
Bases: *ly.lex.lilypond.ChordItem*

rx = u'((?<![a-z])|^)(aug|dim|sus|min|maj|m)(?![a-z])'

class ly.lex.lilypond.**ChordSeparator**
Bases: *ly.lex.lilypond.ChordItem*

rx = u':|\\^|/\\+?'

class ly.lex.lilypond.**ChordStart**
Bases: *ly.lex.lilypond.Chord*

rx = u'<'

update_state(*state*)

class ly.lex.lilypond.**ChordStepNumber**
Bases: *ly.lex.lilypond.ChordItem*

rx = u'\\d+[-+]?'

class ly.lex.lilypond.**Clef**
Bases: *ly.lex.lilypond.Command*

rx = u'\\\\clef\\b'

update_state(*state*)

class ly.lex.lilypond.**ClefSpecifier**
Bases: *ly.lex.lilypond.Specifier*

rx = u'\\b(alto|baritone|bass|C|F|french|G|GG|mezzosoprano|percussion|soprano|subbass|tab|tenor|tenorG|treble|varbarit

update_state(*state*)

class ly.lex.lilypond.**CloseBracket**
Bases: *ly.lex.lilypond.Delimiter*, ly.lex._token.MatchEnd, ly.lex._token.Dedent

matchname = u'bracket'

rx = u'\\}'

update_state(*state*)

class ly.lex.lilypond.**CloseBracketMarkup**
Bases: *ly.lex.lilypond.CloseBracket*

update_state(*state*)

class ly.lex.lilypond.**CloseSimultaneous**
Bases: *ly.lex.lilypond.Delimiter*, ly.lex._token.MatchEnd, ly.lex._token.Dedent

matchname = u'simultaneous'

rx = u'>>'

update_state(*state*)

class ly.lex.lilypond.**Command**
Bases: ly.lex._token.Item, *ly.lex.lilypond.IdentifierRef*

classmethod test_match(*match*)

class ly.lex.lilypond.**Comment**
Bases: ly.lex._token.Comment

class ly.lex.lilypond.**Context**
Bases: *ly.lex.lilypond.Translator*

rx = u'\\\\context\\b'

class ly.lex.lilypond.**ContextName**
Bases: ly.lex._token.Token

rx = u'\\b(ChoirStaff|ChordNames|CueVoice|Devnull|DrumStaff|DrumVoice|Dynamics|FiguredBass|FretBoards|Global|G

class ly.lex.lilypond.**ContextProperty**
Bases: *ly.lex.lilypond.Variable*

rx = u'\\b(aDueText|accidentalGrouping|additionalPitchPrefix|alignAboveContext|alignBassFigureAccidentals|alignBelov

class ly.lex.lilypond.**DecimalValue**
Bases: *ly.lex.lilypond.Value*

rx = u'-?\\d+(\\.\\d+)?'

class ly.lex.lilypond.**Delimiter**
Bases: ly.lex._token.Token

class ly.lex.lilypond.**Direction**
Bases: ly.lex._token.Token

rx = u'[-_^]'

update_state(*state*)

class ly.lex.lilypond.**Dot**
Bases: *ly.lex.lilypond.Duration*

rx = u'\\.'

class ly.lex.lilypond.**DotChord**
Bases: *ly.lex.lilypond.ChordItem*

rx = u'\\.'

class ly.lex.lilypond.**DotPath**
Bases: *ly.lex.lilypond.Delimiter*

A dot in dotted path notation.

rx = u'\\.'

class ly.lex.lilypond.**DrumChordEnd**
Bases: *ly.lex.lilypond.ChordEnd*

class ly.lex.lilypond.**DrumChordStart**
Bases: *ly.lex.lilypond.ChordStart*

update_state(*state*)

class ly.lex.lilypond.**DrumMode**
Bases: *ly.lex.lilypond.InputMode*

rx = u'\\\\(drums|drummode)\\b'

update_state(*state*)

class ly.lex.lilypond.**DrumNote**
Bases: *ly.lex.lilypond.MusicItem*

rx = u'[a-z]+(?![A-Za-z])'

class ly.lex.lilypond.**Duration**
Bases: ly.lex._token.Token

class ly.lex.lilypond.**Dynamic**
Bases: ly.lex._token.Token

rx = u'\\\\[<!>]|\\\\(f{1,5}|p{1,5}|mf|mp|fp|spp?|sff?|sfz|rfz|cresc|decresc|dim|cr|decr)(?![A-Za-z])'

class ly.lex.lilypond.**EqualSign**
Bases: ly.lex._token.Token

rx = u'='

class ly.lex.lilypond.**Error**
Bases: ly.lex._token.Error

class ly.lex.lilypond.**ErrorInChord**
Bases: *ly.lex.lilypond.Error*

rx = u'[-_^][_.>|!+^-]|<<|>>|\\\\[\\\\\\]\\[\\(\\)()]|(\\\\(maxima|longa|breve)\\b|(1|2|4|8|16|32|64|128|256|512|1024|2048)(?!\\d))|

class ly.lex.lilypond.**ExpectBook**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectOpenBracket*

replace
alias of *ParseBook*

class ly.lex.lilypond.**ExpectBookPart**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectOpenBracket*

replace
alias of *ParseBookPart*

class ly.lex.lilypond.**ExpectChordMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectMusicList*

replace
alias of *ParseChordMode*

class ly.lex.lilypond.**ExpectContext**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectOpenBracket*

replace
alias of *ParseContext*

class ly.lex.lilypond.**ExpectDrumMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectMusicList*

replace
alias of *ParseDrumMode*

class ly.lex.lilypond.**ExpectFigureMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectMusicList*

replace
alias of *ParseFigureMode*

class ly.lex.lilypond.**ExpectGrobProperty**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x3328530>

update_state(*state*, *token*)

class ly.lex.lilypond.**ExpectHeader**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectOpenBracket*

replace
alias of *ParseHeader*

class ly.lex.lilypond.**ExpectLayout**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectOpenBracket*

replace
alias of *ParseLayout*

class ly.lex.lilypond.**ExpectLyricMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectMusicList*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x36bd770>

replace
alias of *ParseLyricMode*

class ly.lex.lilypond.**ExpectMidi**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectOpenBracket*

replace
alias of *ParseMidi*

class ly.lex.lilypond.**ExpectMusicList**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*, *ly.lex.lilypond.ParseLilyPond*

Waits for an OpenBracket or << and then replaces the parser with the class set in the replace attribute.

Subclass this to set the destination for the OpenBracket.

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x2f9c9b0>

update_state(*state*, *token*)

class ly.lex.lilypond.**ExpectNoteMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectMusicList*

replace
alias of *ParseNoteMode*

class ly.lex.lilypond.**ExpectOpenBracket**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*, *ly.lex.lilypond.ParseLilyPond*

Waits for an OpenBracket and then replaces the parser with the class set in the replace attribute.

Subclass this to set the destination for the OpenBracket.

default
alias of *Error*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object>

update_state(*state*, *token*)

class ly.lex.lilypond.**ExpectPaper**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectOpenBracket*

replace
alias of *ParsePaper*

class ly.lex.lilypond.**ExpectScore**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectOpenBracket*

replace
alias of *ParseScore*

class ly.lex.lilypond.**ExpectTranslatorId**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object>

update_state(*state*, *token*)

class ly.lex.lilypond.**ExpectWith**(*argcount=None*)
Bases: *ly.lex.lilypond.ExpectOpenBracket*

replace
alias of *ParseWith*

class ly.lex.lilypond.**Figure**
Bases: ly.lex._token.Token

Base class for Figure items.

class ly.lex.lilypond.**FigureAccidental**
Bases: *ly.lex.lilypond.Figure*

A figure accidental.

rx = u'[-+!]+'

class ly.lex.lilypond.**FigureBracket**
Bases: *ly.lex.lilypond.Figure*

rx = u'[][]'

class ly.lex.lilypond.**FigureEnd**
Bases: *ly.lex.lilypond.Figure*, ly.lex._token.Leaver

rx = u'>'

class ly.lex.lilypond.**FigureMode**
Bases: *ly.lex.lilypond.InputMode*

rx = u'\\\\(figures|figuremode)\\b'

update_state(*state*)

class ly.lex.lilypond.**FigureModifier**
Bases: *ly.lex.lilypond.Figure*

A figure modifier.

rx = u'\\\\[\\\\!+]|/'

class ly.lex.lilypond.**FigureStart**
Bases: *ly.lex.lilypond.Figure*

rx = u'<'

update_state(*state*)

class ly.lex.lilypond.**FigureStep**
Bases: *ly.lex.lilypond.Figure*

A step figure number or the underscore.

rx = u'_|\\d+'

class ly.lex.lilypond.**Fingering**
Bases: *ly.lex.lilypond.Articulation*, ly.lex._token.Leaver

rx = u'\\d+'

class ly.lex.lilypond.**Fraction**
Bases: *ly.lex.lilypond.Value*

rx = u'\\d+/\\d+'

class ly.lex.lilypond.**GrobName**
Bases: ly.lex._token.Token

rx = u'\\b(Accidental|AccidentalCautionary|AccidentalPlacement|AccidentalSuggestion|Ambitus|AmbitusAccidental|Aml

class ly.lex.lilypond.**GrobProperty**
Bases: *ly.lex.lilypond.Variable*

rx = u'\\b([a-z]+|[XY])(-([a-z]+|[XY]))*(?![\\w])'

class ly.lex.lilypond.**Header**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\header\\b'

update_state(*state*)

class ly.lex.lilypond.**HeaderVariable**
Bases: *ly.lex.lilypond.Variable*

A variable inside Header. Always follow this one by UserVariable.

classmethod test_match(*match*)

class ly.lex.lilypond.**Hide**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\hide\\b'

update_state(*state*)

class ly.lex.lilypond.**Identifier**
Bases: ly.lex._token.Token

A variable name, like `some-variable`.

rx = u'(?<![^\\W\\d])[^\\W\\d_]+([_-][^\\W\\d_]+)*(?![_-]?[^\\W\\d])'

class ly.lex.lilypond.**IdentifierRef**
Bases: ly.lex._token.Token

A reference to an identifier, e.g. `\some-variable`.

rx = u'\\\\[^\\W\\d_]+([_-][^\\W\\d_]+)*(?![_-]?[^\\W\\d])'

class ly.lex.lilypond.**InputMode**
Bases: *ly.lex.lilypond.Command*

class ly.lex.lilypond.**IntegerValue**
Bases: *ly.lex.lilypond.DecimalValue*

rx = u'\\d+'

class ly.lex.lilypond.**KeySignatureMode**
Bases: *ly.lex.lilypond.Command*

rx = u'\\\\(major|minor|ionian|dorian|phrygian|lydian|mixolydian|aeolian|locrian)(?![A-Za-z])'

class ly.lex.lilypond.**Keyword**
Bases: ly.lex._token.Item, *ly.lex.lilypond.IdentifierRef*

classmethod test_match(*match*)

class ly.lex.lilypond.**Layout**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\layout\\b'

update_state(*state*)

class ly.lex.lilypond.**LayoutContext**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\context\\b'

update_state(*state*)

class ly.lex.lilypond.**LayoutVariable**
Bases: *ly.lex.lilypond.Variable*

A variable inside Header. Always follow this one by UserVariable.

classmethod test_match(*match*)

class ly.lex.lilypond.**Length**
Bases: *ly.lex.lilypond.Duration*

rx = u'(\\\\(maxima|longa|breve)\\b|(1|2|4|8|16|32|64|128|256|512|1024|2048)(?!\\d))'

update_state(*state*)

class ly.lex.lilypond.**Ligature**
Bases: ly.lex._token.Token

class ly.lex.lilypond.**LigatureEnd**
Bases: *ly.lex.lilypond.Ligature*, ly.lex._token.MatchEnd

matchname = u'ligature'

rx = u'\\\\\\]'

class ly.lex.lilypond.**LigatureStart**
Bases: *ly.lex.lilypond.Ligature*, ly.lex._token.MatchStart

matchname = u'ligature'

rx = u'\\\\\\['

class ly.lex.lilypond.**LineComment**
Bases: *ly.lex.lilypond.Comment*, ly.lex._token.LineComment

rx = u'%.*$'

class ly.lex.lilypond.**Lyric**
Bases: ly.lex._token.Item

Base class for Lyric items.

class ly.lex.lilypond.**LyricExtender**
Bases: *ly.lex.lilypond.Lyric*

rx = u'__(?=($|[\\s\\\\]))'

class ly.lex.lilypond.**LyricHyphen**
Bases: *ly.lex.lilypond.Lyric*

rx = u'–(?=($|[\\s\\\\]))'

class ly.lex.lilypond.**LyricMode**
Bases: *ly.lex.lilypond.InputMode*

rx = u'\\\\(lyricmode|((old)?add)?lyrics|lyricsto)\\b'

update_state(*state*)

class ly.lex.lilypond.**LyricSkip**
Bases: *ly.lex.lilypond.Lyric*

rx = u'_(?=($|[\\s\\\\]))'

class ly.lex.lilypond.**LyricText**
Bases: *ly.lex.lilypond.Lyric*

rx = u'[^\\\\\\s\\d\\"]+'

class ly.lex.lilypond.**Markup**
Bases: ly.lex._token.Item

Base class for all markup commands.

class ly.lex.lilypond.**MarkupCommand**
Bases: *ly.lex.lilypond.Markup*, *ly.lex.lilypond.IdentifierRef*

A markup command.

classmethod test_match(*match*)

update_state(*state*)

class ly.lex.lilypond.**MarkupLines**
Bases: *ly.lex.lilypond.Markup*

rx = u'\\\\markuplines(?![_-]?[^\\W\\d])'

update_state(*state*)

class ly.lex.lilypond.**MarkupList**
Bases: *ly.lex.lilypond.Markup*

rx = u'\\\\markuplist(?![_-]?[^\\W\\d])'

update_state(*state*)

class ly.lex.lilypond.**MarkupScore**
Bases: *ly.lex.lilypond.Markup*

rx = u'\\\\score\\b'

update_state(*state*)

class ly.lex.lilypond.**MarkupStart**
Bases: *ly.lex.lilypond.Markup*, *ly.lex.lilypond.Command*

rx = u'\\\\markup(?![_-]?[^\\W\\d])'

update_state(*state*)

class ly.lex.lilypond.**MarkupUserCommand**
Bases: *ly.lex.lilypond.Markup*, *ly.lex.lilypond.IdentifierRef*

A user-defined markup (i.e. not in the words markupcommands list).

update_state(*state*)

class ly.lex.lilypond.**MarkupWord**
Bases: ly.lex._token.Item

rx = u'[^{}"\\\\\\s#%]+'

class ly.lex.lilypond.**Midi**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\midi\\b'

update_state(*state*)

class ly.lex.lilypond.**MusicItem**
Bases: ly.lex._token.Token

A note, rest, spacer, \skip or q.

class ly.lex.lilypond.**Name**
Bases: *ly.lex.lilypond.UserVariable*

A variable name without prefix.

class ly.lex.lilypond.**New**
Bases: *ly.lex.lilypond.Translator*

rx = u'\\\\new\\b'

class ly.lex.lilypond.**Note**
Bases: *ly.lex.lilypond.MusicItem*

rx = u'[a-x]+(?![A-Za-z])'

class ly.lex.lilypond.**NoteMode**
Bases: *ly.lex.lilypond.InputMode*

rx = u'\\\\(notes|notemode)\\b'

update_state(*state*)

class ly.lex.lilypond.**Octave**
Bases: ly.lex._token.Token

rx = u",+|'+"

class ly.lex.lilypond.**OctaveCheck**
Bases: ly.lex._token.Token

rx = u"=(,+|'+)?"

class ly.lex.lilypond.**Omit**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\omit\\b'

update_state(*state*)

class ly.lex.lilypond.**OpenBracket**
Bases: *ly.lex.lilypond.Delimiter*, ly.lex._token.MatchStart, ly.lex._token.Indent

An open bracket, does not enter different parser, subclass or reimplement Parser.update_state().

matchname = u'bracket'

rx = u'\\{'

class ly.lex.lilypond.**OpenBracketMarkup**
Bases: *ly.lex.lilypond.OpenBracket*

update_state(*state*)

class ly.lex.lilypond.**OpenSimultaneous**
Bases: *ly.lex.lilypond.Delimiter*, ly.lex._token.MatchStart, ly.lex._token.Indent

An open double French quote, does not enter different parser, subclass or reimplement Parser.update_state().

matchname = u'simultaneous'

rx = u'<<'

class ly.lex.lilypond.**Override**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\override\\b'

update_state(*state*)

class ly.lex.lilypond.**Paper**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\paper\\b'

update_state(*state*)

class ly.lex.lilypond.**PaperVariable**
Bases: *ly.lex.lilypond.Variable*

A variable inside Paper. Always follow this one by UserVariable.

classmethod test_match(*match*)

class ly.lex.lilypond.**ParseAccidentalStyle**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x370edb0>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseAlterBroken**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x3328530>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseBlockComment**(*argcount=None*)
Bases: *ly.lex.Parser*

default
alias of *BlockComment*

items = (<class 'ly.lex.lilypond.BlockCommentEnd'>,)

pattern = <_sre.SRE_Pattern object>

class ly.lex.lilypond.**ParseBook**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses the expression after `\book {`, leaving at `}`

items = (<class 'ly.lex.lilypond.CloseBracket'>, <class 'ly.lex.lilypond.MarkupStart'>, <class 'ly.lex.lilypond.MarkupLin

pattern = <_sre.SRE_Pattern object at 0x37040d0>

class ly.lex.lilypond.**ParseBookPart**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses the expression after \bookpart {, leaving at }

items = (<class 'ly.lex.lilypond.CloseBracket'>, <class 'ly.lex.lilypond.MarkupStart'>, <class 'ly.lex.lilypond.MarkupLin

pattern = <_sre.SRE_Pattern object at 0x3721560>

class ly.lex.lilypond.**ParseChord**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseMusic*

LilyPond inside chords < >

items = (<class 'ly.lex.lilypond.ErrorInChord'>, <class 'ly.lex.lilypond.ChordEnd'>, <class 'ly.lex._token.Space'>, <class

pattern = <_sre.SRE_Pattern object at 0x373a750>

class ly.lex.lilypond.**ParseChordItems**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex.lilypond.ChordSeparator'>, <class 'ly.lex.lilypond.ChordModifier'>, <class 'ly.lex.lilypond.ChordS

pattern = <_sre.SRE_Pattern object at 0x371eed0>

class ly.lex.lilypond.**ParseChordMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseInputMode*, *ly.lex.lilypond.ParseMusic*

Parser for \chords and \chordmode.

items = (<class 'ly.lex.lilypond.OpenBracket'>, <class 'ly.lex.lilypond.OpenSimultaneous'>, <class 'ly.lex._token.Space'>

pattern = <_sre.SRE_Pattern object at 0x3750a20>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseClef**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

argcount = 1

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x37399a0>

class ly.lex.lilypond.**ParseContext**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses the expression after (\layout {}) \context {, leaving at }

items = (<class 'ly.lex.lilypond.CloseBracket'>, <class 'ly.lex.lilypond.BackSlashedContextName'>, <class 'ly.lex.lilypond

pattern = <_sre.SRE_Pattern object at 0x376a910>

class ly.lex.lilypond.**ParseDecimalValue**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

Parses a decimal value without a # before it (if present).

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x289c540>

class ly.lex.lilypond.**ParseDrumChord**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseMusic*

LilyPond inside chords in drummode < >

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x375b960>

class ly.lex.lilypond.**ParseDrumMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseInputMode*, *ly.lex.lilypond.ParseMusic*

Parser for \drums and \drummode.

items = (<class 'ly.lex.lilypond.OpenBracket'>, <class 'ly.lex.lilypond.OpenSimultaneous'>, <class 'ly.lex._token.Space'>

pattern = <_sre.SRE_Pattern object at 0x3786870>

class ly.lex.lilypond.**ParseDuration**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

fallthrough(*state*)

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object>

class ly.lex.lilypond.**ParseDurationScaling**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseDuration*

fallthrough(*state*)

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object>

class ly.lex.lilypond.**ParseFigure**(*argcount=None*)
Bases: *ly.lex.Parser*

Parse inside < > in figure mode.

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x3783280>

class ly.lex.lilypond.**ParseFigureMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseInputMode*, *ly.lex.lilypond.ParseMusic*

Parser for \figures and \figuremode.

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x3716f10>

class ly.lex.lilypond.**ParseGlobal**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses LilyPond from the toplevel of a file.

items = (<class 'ly.lex.lilypond.Book'>, <class 'ly.lex.lilypond.BookPart'>, <class 'ly.lex.lilypond.Score'>, <class 'ly.lex.lil

pattern = <_sre.SRE_Pattern object at 0x3792b80>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseGlobalAssignment**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*, *ly.lex.lilypond.ParseLilyPond*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x37907a0>

class ly.lex.lilypond.**ParseGrobPropertyPath**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseHeader**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses the expression after `\header {`, leaving at `}`

items = (<class 'ly.lex.lilypond.CloseBracket'>, <class 'ly.lex.lilypond.MarkupStart'>, <class 'ly.lex.lilypond.MarkupLin

pattern = <_sre.SRE_Pattern object at 0x379f8b0>

class ly.lex.lilypond.**ParseHideOmit**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x37b1810>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseInputMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Base class for parser for mode-changing music commands.

classmethod update_state(*state*, *token*)

class ly.lex.lilypond.**ParseLayout**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses the expression after `\layout {`, leaving at `}`

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x37c3a50>

class ly.lex.lilypond.**ParseLilyPond**(*argcount=None*)
Bases: *ly.lex.Parser*

mode = u'lilypond'

class ly.lex.lilypond.**ParseLyricMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseInputMode*

Parser for `\lyrics`, `\lyricmode`, `\addlyrics`, etc.

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x37bc2b0>

class ly.lex.lilypond.**ParseMarkup**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex.lilypond.MarkupScore'>, <class 'ly.lex.lilypond.MarkupCommand'>, <class 'ly.lex.lilypond.Mark

pattern = <_sre.SRE_Pattern object at 0x37b10b0>

class ly.lex.lilypond.**ParseMidi**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses the expression after `\midi {`, leaving at `}`

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x37c3a50>

class ly.lex.lilypond.**ParseMusic**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses LilyPond music expressions.

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x37e5f30>

class ly.lex.lilypond.**ParseNoteMode**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseMusic*

Parser for \notes and \notemode. Same as Music itself.

class ly.lex.lilypond.**ParseOverride**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

argcount = 0

items = (<class 'ly.lex.lilypond.ContextName'>, <class 'ly.lex.lilypond.DotPath'>, <class 'ly.lex.lilypond.GrobName'>, <c

pattern = <_sre.SRE_Pattern object at 0x37ecde0>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParsePaper**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses the expression after \paper {, leaving at }

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x377f8f0>

class ly.lex.lilypond.**ParsePitchCommand**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

argcount = 1

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x2679df0>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseRepeat**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x37c3660>

class ly.lex.lilypond.**ParseRevert**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

parse the arguments of \revert

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x37e0620>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseScore**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses the expression after \score {, leaving at }

items = (<class 'ly.lex.lilypond.CloseBracket'>, <class 'ly.lex.lilypond.Header'>, <class 'ly.lex.lilypond.Layout'>, <class '

pattern = <_sre.SRE_Pattern object at 0x37f8120>

class ly.lex.lilypond.**ParseScriptAbbreviationOrFingering**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

argcount = 1

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object>

class ly.lex.lilypond.**ParseSet**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

argcount = 0

items = (<class 'ly.lex.lilypond.ContextName'>, <class 'ly.lex.lilypond.DotPath'>, <class 'ly.lex.lilypond.ContextProperty

pattern = <_sre.SRE_Pattern object at 0x38150b0>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseString**(*argcount=None*)
Bases: *ly.lex.Parser*

default
alias of *String*

items = (<class 'ly.lex.lilypond.StringQuotedEnd'>, <class 'ly.lex.lilypond.StringQuoteEscape'>)

pattern = <_sre.SRE_Pattern object>

class ly.lex.lilypond.**ParseTempo**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x38042c0>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseTempoAfterEqualSign**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object>

class ly.lex.lilypond.**ParseTranslator**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x38132b0>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseTranslatorId**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

argcount = 1

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x2885850>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseTremolo**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex.lilypond.TremoloDuration'>,)

pattern = <_sre.SRE_Pattern object>

class ly.lex.lilypond.**ParseTweak**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x37ced50>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseTweakGrobProperty**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x26982e0>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseUnset**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>, <class 'ly.lex.lilypond.LineCommen

pattern = <_sre.SRE_Pattern object at 0x383b9d0>

update_state(*state*, *token*)

class ly.lex.lilypond.**ParseWith**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseLilyPond*

Parses the expression after `\with {`, leaving at `}`

items = (<class 'ly.lex.lilypond.CloseBracket'>, <class 'ly.lex.lilypond.ContextName'>, <class 'ly.lex.lilypond.GrobName

pattern = <_sre.SRE_Pattern object at 0x382c400>

class ly.lex.lilypond.**Partial**
Bases: *ly.lex.lilypond.Command*

rx = u'\\\\partial\\b'

class ly.lex.lilypond.**PhrasingSlurEnd**
Bases: *ly.lex.lilypond.SlurEnd*

matchname = u'phrasingslur'

rx = u'\\\\\\)'

class ly.lex.lilypond.**PhrasingSlurStart**
Bases: *ly.lex.lilypond.SlurStart*

matchname = u'phrasingslur'

rx = u'\\\\\\('

class ly.lex.lilypond.**PipeSymbol**
Bases: *ly.lex.lilypond.Delimiter*

rx = u'\\|'

class ly.lex.lilypond.**PitchCommand**
Bases: *ly.lex.lilypond.Command*

rx = u'\\\\(relative|transpose|transposition|key|octaveCheck)\\b'

update_state(*state*)

class ly.lex.lilypond.**Q**
Bases: *ly.lex.lilypond.MusicItem*

rx = u'q(?![A-Za-z])'

class ly.lex.lilypond.**Repeat**
Bases: *ly.lex.lilypond.Command*

rx = u'\\\\repeat(?![A-Za-z])'

update_state(*state*)

class ly.lex.lilypond.**RepeatCount**
Bases: *ly.lex.lilypond.IntegerValue*, ly.lex._token.Leaver

class ly.lex.lilypond.**RepeatSpecifier**
Bases: *ly.lex.lilypond.Specifier*

rx = u'\\b(unfold|percent|volta|tremolo)(?![A-Za-z])'

class ly.lex.lilypond.**Rest**
Bases: *ly.lex.lilypond.MusicItem*

rx = u'[Rr](?![A-Za-z])'

class ly.lex.lilypond.**Revert**
Bases: *ly.lex.lilypond.Override*

rx = u'\\\\revert\\b'

update_state(*state*)

class ly.lex.lilypond.**Scaling**
Bases: *ly.lex.lilypond.Duration*

rx = u'*[\\t]*\\d+(/\\d+)?'

class ly.lex.lilypond.**SchemeStart**
Bases: ly.lex._token.Item

rx = u'[#$](?![{}])'

update_state(*state*)

class ly.lex.lilypond.**Score**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\score\\b'

update_state(*state*)

class ly.lex.lilypond.**ScriptAbbreviation**
Bases: *ly.lex.lilypond.Articulation*, ly.lex._token.Leaver

rx = u'[+|!>._^-]'

class ly.lex.lilypond.**SequentialEnd**
Bases: *ly.lex.lilypond.CloseBracket*

class ly.lex.lilypond.**SequentialStart**
Bases: *ly.lex.lilypond.OpenBracket*

update_state(*state*)

class ly.lex.lilypond.**Set**
Bases: *ly.lex.lilypond.Override*

rx = u'\\\\set\\b'

update_state(*state*)

class ly.lex.lilypond.**SimultaneousEnd**
Bases: *ly.lex.lilypond.CloseSimultaneous*

class ly.lex.lilypond.**SimultaneousOrSequentialCommand**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\(simultaneous|sequential)\\b'

class ly.lex.lilypond.**SimultaneousStart**
Bases: *ly.lex.lilypond.OpenSimultaneous*

update_state(*state*)

class ly.lex.lilypond.**Skip**
Bases: *ly.lex.lilypond.MusicItem*

rx = u'\\\\skip(?![_-]?[^\\W\\d])'

class ly.lex.lilypond.**Slur**
Bases: ly.lex._token.Token

class ly.lex.lilypond.**SlurEnd**
Bases: *ly.lex.lilypond.Slur*, ly.lex._token.MatchEnd

matchname = u'slur'

rx = u'\\)'

class ly.lex.lilypond.**SlurStart**
Bases: *ly.lex.lilypond.Slur*, ly.lex._token.MatchStart

matchname = u'slur'

rx = u'\\('

class ly.lex.lilypond.**Spacer**
Bases: *ly.lex.lilypond.MusicItem*

rx = u's(?![A-Za-z])'

class ly.lex.lilypond.**Specifier**
Bases: ly.lex._token.Token

class ly.lex.lilypond.**String**
Bases: ly.lex._token.String

class ly.lex.lilypond.**StringNumber**
Bases: *ly.lex.lilypond.Articulation*

rx = u'\\\\\\d+'

class ly.lex.lilypond.**StringQuoteEscape**
Bases: ly.lex._token.Character

rx = u'\\\\[\\\\"]'

class ly.lex.lilypond.**StringQuotedEnd**
Bases: *ly.lex.lilypond.String*, ly.lex._token.StringEnd

rx = u''''

update_state(*state*)

class ly.lex.lilypond.**StringQuotedStart**
Bases: *ly.lex.lilypond.String*, ly.lex._token.StringStart

rx = u''''

update_state(*state*)

class ly.lex.lilypond.**Tempo**
Bases: *ly.lex.lilypond.Command*

rx = u'\\\\tempo\\b'

update_state(*state*)

class ly.lex.lilypond.**TempoSeparator**
Bases: *ly.lex.lilypond.Delimiter*

rx = u'[-~](?=\\s*\\d)'

class ly.lex.lilypond.**Tie**
Bases: *ly.lex.lilypond.Slur*

rx = u'~'

class ly.lex.lilypond.**Translator**
Bases: *ly.lex.lilypond.Command*

update_state(*state*)

class ly.lex.lilypond.**Tremolo**
Bases: ly.lex._token.Token

class ly.lex.lilypond.**TremoloColon**
Bases: *ly.lex.lilypond.Tremolo*

rx = u':'

update_state(*state*)

class ly.lex.lilypond.**TremoloDuration**
Bases: *ly.lex.lilypond.Tremolo*, ly.lex._token.Leaver

rx = u'\\b(8|16|32|64|128|256|512|1024|2048)(?!\\d)'

class ly.lex.lilypond.**Tweak**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\tweak\\b'

update_state(*state*)

class ly.lex.lilypond.**Unit**
Bases: *ly.lex.lilypond.Command*

rx = u'\\\\(mm|cm|in|pt)\\b'

class ly.lex.lilypond.**Unset**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\unset\\b'

update_state(*state*)

class ly.lex.lilypond.**UserCommand**
Bases: *ly.lex.lilypond.IdentifierRef*

class ly.lex.lilypond.**UserVariable**
Bases: *ly.lex.lilypond.Identifier*

class ly.lex.lilypond.**Value**
Bases: ly.lex._token.Item, ly.lex._token.Numeric

class ly.lex.lilypond.**Variable**
Bases: *ly.lex.lilypond.Identifier*

class ly.lex.lilypond.**VoiceSeparator**
Bases: *ly.lex.lilypond.Delimiter*

rx = u'\\\\\\\\'

class ly.lex.lilypond.**With**
Bases: *ly.lex.lilypond.Keyword*

rx = u'\\\\with\\b'

update_state(*state*)

ly.lex.scheme module

Parses and tokenizes Scheme input.

class ly.lex.scheme.**BlockComment**
Bases: *ly.lex.scheme.Comment*, ly.lex._token.BlockComment

class ly.lex.scheme.**BlockCommentEnd**
Bases: *ly.lex.scheme.Comment*, ly.lex._token.BlockCommentEnd, ly.lex._token.Leaver

rx = u'!#'

class ly.lex.scheme.**BlockCommentStart**
Bases: *ly.lex.scheme.Comment*, ly.lex._token.BlockCommentStart

rx = u'#!'

update_state(*state*)

class ly.lex.scheme.**Bool**
Bases: *ly.lex.scheme.Scheme*, ly.lex._token.Item

rx = u'#[tf]\\b'

class ly.lex.scheme.**Char**
Bases: *ly.lex.scheme.Scheme*, ly.lex._token.Item

rx = u'#\\\\([a-z]+|.)'

class ly.lex.scheme.**CloseParen**
Bases: *ly.lex.scheme.Scheme*, ly.lex._token.MatchEnd, ly.lex._token.Dedent

matchname = u'schemeparen'

rx = u'\\)'

update_state(*state*)

class ly.lex.scheme.**Comment**
Bases: ly.lex._token.Comment

class ly.lex.scheme.**Constant**
Bases: *ly.lex.scheme.Word*

classmethod test_match(*match*)

class ly.lex.scheme.**Dot**
Bases: *ly.lex.scheme.Scheme*

rx = u'\\.(?!\\S)'

class ly.lex.scheme.**Float**
Bases: *ly.lex.scheme.Number*

rx = u'-?((\\d+(\\.\\d*)|\\.\\d+)(E\\d+)?)(?=$|[)\\s])'

class ly.lex.scheme.**Fraction**
Bases: *ly.lex.scheme.Number*

rx = u'-?\\d+/\\d+(?=$|[)\\s])'

class ly.lex.scheme.**Function**
Bases: *ly.lex.scheme.Word*

classmethod test_match(*match*)

class ly.lex.scheme.**Keyword**
Bases: *ly.lex.scheme.Word*

classmethod test_match(*match*)

class ly.lex.scheme.**LilyPond**
Bases: ly.lex._token.Token

class ly.lex.scheme.**LilyPondEnd**
Bases: *ly.lex.scheme.LilyPond*, ly.lex._token.Leaver, ly.lex._token.MatchEnd, ly.lex._token.Dedent

matchname = u'schemelily'

rx = u'#}'

class ly.lex.scheme.**LilyPondStart**
Bases: *ly.lex.scheme.LilyPond*, ly.lex._token.MatchStart, ly.lex._token.Indent

matchname = u'schemelily'

rx = u'#{'

update_state(*state*)

class ly.lex.scheme.**LineComment**
Bases: *ly.lex.scheme.Comment*, ly.lex._token.LineComment

rx = u';.*$'

class ly.lex.scheme.**Number**
Bases: ly.lex._token.Item, ly.lex._token.Numeric

rx = u'(-?\\d+|#(b[0-1]+|o[0-7]+|x[0-9a-fA-F]+)|[-+]inf.0|[-+]?nan.0)(?=$|[)\\s])'

class ly.lex.scheme.**OpenParen**
Bases: *ly.lex.scheme.Scheme*, ly.lex._token.MatchStart, ly.lex._token.Indent

matchname = u'schemeparen'

rx = u'\\('

update_state(*state*)

class ly.lex.scheme.**ParseBlockComment**(*argcount=None*)
Bases: *ly.lex.Parser*

default
alias of *BlockComment*

items = (<class 'ly.lex.scheme.BlockCommentEnd'>,)

pattern = <_sre.SRE_Pattern object>

class ly.lex.scheme.**ParseLilyPond**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseMusic*

items = (<class 'ly.lex.scheme.LilyPondEnd'>, <class 'ly.lex._token.Space'>, <class 'ly.lex.lilypond.BlockCommentStart'>

pattern = <_sre.SRE_Pattern object at 0x35158d0>

class ly.lex.scheme.**ParseScheme**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex._token.Space'>, <class 'ly.lex.scheme.OpenParen'>, <class 'ly.lex.scheme.CloseParen'>, <class 'ly.l

mode = u'scheme'

pattern = <_sre.SRE_Pattern object at 0x31e1ea0>

class ly.lex.scheme.**ParseString**(*argcount=None*)
Bases: *ly.lex.Parser*

default
alias of *String*

items = (<class 'ly.lex.scheme.StringQuotedEnd'>, <class 'ly.lex.scheme.StringQuoteEscape'>)

pattern = <_sre.SRE_Pattern object>

class ly.lex.scheme.**Quote**
Bases: *ly.lex.scheme.Scheme*

rx = u"['`,]"

class ly.lex.scheme.**Scheme**
Bases: ly.lex._token.Token

Baseclass for Scheme tokens.

class ly.lex.scheme.**String**
Bases: ly.lex._token.String

class ly.lex.scheme.**StringQuoteEscape**
Bases: ly.lex._token.Character

rx = u'\\\\[\\\\"]'

class ly.lex.scheme.**StringQuotedEnd**
Bases: *ly.lex.scheme.String*, ly.lex._token.StringEnd

rx = u'"'

update_state(*state*)

class ly.lex.scheme.**StringQuotedStart**
Bases: *ly.lex.scheme.String*, ly.lex._token.StringStart

rx = u''''

update_state(*state*)

class ly.lex.scheme.**Variable**
Bases: *ly.lex.scheme.Word*

classmethod test_match(*match*)

class ly.lex.scheme.**VectorStart**
Bases: *ly.lex.scheme.OpenParen*

rx = u'#\\('

class ly.lex.scheme.**Word**
Bases: *ly.lex.scheme.Scheme*, ly.lex._token.Item

rx = u'[^()''{}\\s]+'

ly.lex.texinfo module

Parses and tokenizes Texinfo input, recognizing LilyPond in Texinfo.

class ly.lex.texinfo.**Accent**
Bases: *ly.lex.texinfo.EscapeChar*

rx = u'@[\''\',=^`~](\\{[a-zA-Z]\\}|[a-zA-Z]\\b)'

class ly.lex.texinfo.**Attribute**
Bases: ly.lex._token.Token

class ly.lex.texinfo.**Block**
Bases: ly.lex._token.Token

class ly.lex.texinfo.**BlockCommentEnd**
Bases: *ly.lex.texinfo.Comment*, ly.lex._token.Leaver, ly.lex._token.BlockCommentEnd

rx = u'@end\\s+ignore\\b'

class ly.lex.texinfo.**BlockCommentStart**
Bases: *ly.lex.texinfo.Comment*, ly.lex._token.BlockCommentStart

rx = u'@ignore\\b'

update_state(*state*)

class ly.lex.texinfo.**BlockEnd**
Bases: *ly.lex.texinfo.Block*, ly.lex._token.Leaver

rx = u'\\}'

class ly.lex.texinfo.**BlockStart**
Bases: *ly.lex.texinfo.Block*

rx = u'@[a-zA-Z]+\\{'

update_state(*state*)

class ly.lex.texinfo.**Comment**
Bases: ly.lex._token.Comment

class ly.lex.texinfo.**EscapeChar**
Bases: ly.lex._token.Character

rx = u'@[@{}]'

class ly.lex.texinfo.**Keyword**
Bases: ly.lex._token.Token

rx = u'@[a-zA-Z]+'

class ly.lex.texinfo.**LilyPondAttrEnd**
Bases: *ly.lex.texinfo.Attribute*, ly.lex._token.Leaver

rx = u'\\]'

class ly.lex.texinfo.**LilyPondAttrStart**
Bases: *ly.lex.texinfo.Attribute*

rx = u'\\['

update_state(*state*)

class ly.lex.texinfo.**LilyPondBlockEnd**
Bases: *ly.lex.texinfo.Block*, ly.lex._token.Leaver

rx = u'\\}'

class ly.lex.texinfo.**LilyPondBlockStart**
Bases: *ly.lex.texinfo.Block*

rx = u'@lilypond(?=(\\[[a-zA-Z,=0-9\\\\\\s]+\\])?\\{)'

update_state(*state*)

class ly.lex.texinfo.**LilyPondBlockStartBrace**
Bases: *ly.lex.texinfo.Block*

rx = u'\\{'

update_state(*state*)

class ly.lex.texinfo.**LilyPondEnvEnd**
Bases: *ly.lex.texinfo.Keyword*, ly.lex._token.Leaver

rx = u'@end\\s+lilypond\\b'

class ly.lex.texinfo.**LilyPondEnvStart**
Bases: *ly.lex.texinfo.Keyword*

rx = u'@lilypond\\b'

update_state(*state*)

class ly.lex.texinfo.**LilyPondFileStart**
Bases: *ly.lex.texinfo.Block*

rx = u'@lilypondfile\\b'

update_state(*state*)

class ly.lex.texinfo.**LilyPondFileStartBrace**
Bases: *ly.lex.texinfo.Block*

rx = u'\\{'

update_state(*state*)

class ly.lex.texinfo.**LineComment**
Bases: *ly.lex.texinfo.Comment*, ly.lex._token.LineComment

rx = u'@c\\b.*$'

class ly.lex.texinfo.**ParseBlock**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex.texinfo.BlockEnd'>, <class 'ly.lex.texinfo.Accent'>, <class 'ly.lex.texinfo.EscapeChar'>, <class 'ly.l

pattern = <_sre.SRE_Pattern object at 0x25a9660>

class ly.lex.texinfo.**ParseComment**(*argcount=None*)
Bases: *ly.lex.Parser*

default
alias of *Comment*

items = (<class 'ly.lex.texinfo.BlockCommentEnd'>,)

pattern = <_sre.SRE_Pattern object>

class ly.lex.texinfo.**ParseLilyPondAttr**(*argcount=None*)
Bases: *ly.lex.Parser*

default
alias of *Attribute*

items = (<class 'ly.lex.texinfo.LilyPondAttrEnd'>,)

pattern = <_sre.SRE_Pattern object>

class ly.lex.texinfo.**ParseLilyPondBlock**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseGlobal*

items = (<class 'ly.lex.texinfo.LilyPondBlockEnd'>, <class 'ly.lex.lilypond.Book'>, <class 'ly.lex.lilypond.BookPart'>, <cl

pattern = <_sre.SRE_Pattern object at 0x3aacfa0>

class ly.lex.texinfo.**ParseLilyPondBlockAttr**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex.texinfo.LilyPondAttrStart'>, <class 'ly.lex.texinfo.LilyPondBlockStartBrace'>)

pattern = <_sre.SRE_Pattern object>

class ly.lex.texinfo.**ParseLilyPondEnv**(*argcount=None*)
Bases: *ly.lex.lilypond.ParseGlobal*

items = (<class 'ly.lex.texinfo.LilyPondEnvEnd'>, <class 'ly.lex.lilypond.Book'>, <class 'ly.lex.lilypond.BookPart'>, <clas

pattern = <_sre.SRE_Pattern object at 0x3ab6500>

class ly.lex.texinfo.**ParseLilyPondEnvAttr**(*argcount=None*)
Bases: *ly.lex.FallthroughParser*

fallthrough(*state*)

items = (<class 'ly.lex.texinfo.LilyPondAttrStart'>,)

pattern = <_sre.SRE_Pattern object>

class ly.lex.texinfo.**ParseLilyPondFile**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex.texinfo.LilyPondAttrStart'>, <class 'ly.lex.texinfo.LilyPondFileStartBrace'>)

pattern = <_sre.SRE_Pattern object>

class ly.lex.texinfo.**ParseTexinfo**(*argcount=None*)
Bases: *ly.lex.Parser*

items = (<class 'ly.lex.texinfo.LineComment'>, <class 'ly.lex.texinfo.BlockCommentStart'>, <class 'ly.lex.texinfo.Accent'

mode = u'texinfo'

pattern = <_sre.SRE_Pattern object at 0x3ab4fb0>

class ly.lex.texinfo.**ParseVerbatim**(*argcount=None*)
Bases: *ly.lex.Parser*

default
alias of *Verbatim*

items = (<class 'ly.lex.texinfo.VerbatimEnd'>,)

pattern = <_sre.SRE_Pattern object>

class ly.lex.texinfo.**Verbatim**
Bases: ly.lex._token.Token

class ly.lex.texinfo.**VerbatimEnd**
Bases: *ly.lex.texinfo.Keyword*, ly.lex._token.Leaver

rx = u'@end\\s+verbatim\\b'

class ly.lex.texinfo.**VerbatimStart**
Bases: *ly.lex.texinfo.Keyword*

rx = u'@verbatim\\b'

update_state(*state*)

ly.music package

Module contents

An api to read music from the tokens of a ly.document.Document into a tree structure.

This is meant to quickly read music from a document, to perform modifications on the document, and to interpret music and markup and to convert or export it to other formats.

All nodes are a subclass of items.Item, (which inherits from node.WeakNode).

Tree structures are created from nested LilyPond structures, markup and scheme code. Some Item types have special methods to query information. The Music type, for example, has a length() method that returns the duration of the music fragment.

Using the Music.events() method and the events module, it is possible to iterate in musical time over the music tree, e.g. to convert music to another format.

This package is not yet capable to construct documents entirely from scratch. This needs to be developed. Until that time, the ly.dom module can be used instead.

Some Item types can have a list of child items, but the tree structure is as linear as possible.

A convenience function is available to create a ly.music.items.Document instance for the specified ly.document.Document.

Here is an example:

```
>>> import ly.document
>>> import ly.music
>>> d=ly.document.Document(r'''
\version "2.18.0"

music = \relative {
```

```
  \time 4/4
  \key d \minor
  d4 e f g
  a g f e
  d2
}

\score {
  \new Staff <<
    \music
  >>
}
''')
>>> m=ly.music.document(d)
>>> print m.dump()
<Document>
  <Version u'\\version'>
    <String u'"'>
  <Assignment u'music'>
    <Relative u'\\relative'>
      <MusicList u'{'>
        <TimeSignature u'\\time'>
        <KeySignature u'\\key'>
          <Note u'd'>
          <Command u'\\minor'>
        <Note u'd'>
        <Note u'e'>
        <Note u'f'>
        <Note u'g'>
        <Note u'a'>
        <Note u'g'>
        <Note u'f'>
        <Note u'e'>
        <Note u'd'>
  <Score u'\\score'>
    <Context u'\\new'>
      <MusicList u'<<'>
        <UserCommand u'\\music'>
>>> m[2][0][0]
<MusicList u'<<'>
>>> m[2][0][0].length()
Fraction(5, 2)
>>>
```

ly.music.**document**(*doc*)
Return a music.items.Document instance for the ly.document.Document.

Submodules

ly.music.event module

Translates a music.items.Document tree into lists of events.

class ly.music.event.**Events**
Bases: object

Traverses a music tree and records music events from it.

read(*node*, *time=0*, *scaling=1*)
Read events from the node and all its child nodes; return time.

traverse(*node*, *time*, *scaling*)
Traverse node and call event handlers; record and return the time.

unfold_repeats = False

ly.music.items module

The items a music expression is constructed with in a tree structure.

Whitespace and comments are left out.

All nodes (instances of Item) have a 'position' attribute that indicates where the item starts in the source text. Almost all items have the token that starts the expression in the 'token' attribute and possibly other tokens in the 'tokens' attribute, as a tuple.

The 'end_position()' method returns the position where the node (including its child nodes) ends.

You can get the whole tree structure of a LilyPond document by instantiating a Document with the ly.document.Document instance. (It will read all the tokens from the document using the Reader from the read module.) As a convenience, the ly.music.document(doc) function does this.

If you want to add new Item types, you should also add a method to read.Reader to construct those items.

class ly.music.items.**Absolute**(*parent=None*)
Bases: *ly.music.items.Music*

An absolute music expression. Has one child (normally Music).

class ly.music.items.**AfterGrace**(*parent=None*)
Bases: *ly.music.items.Music*

The afterGrace function with its two arguments.

Only the duration of the first is counted.

class ly.music.items.**Alternative**(*parent=None*)
Bases: *ly.music.items.Music*

An alternative expression.

class ly.music.items.**Articulation**(*parent=None*)
Bases: *ly.music.items.Item*

An articulation, fingering, string number, or other symbol.

class ly.music.items.**Assignment**(*parent=None*)
Bases: *ly.music.items.Item*

A variable = value construct.

name()
The variable name.

value()
The assigned value.

class ly.music.items.**Beam**(*parent=None*)
Bases: *ly.music.items.Item*

A [or].

event = None

class ly.music.items.**Book**(*parent=None*)
Bases: *ly.music.items.Container*

A book { ... } construct.

class ly.music.items.**BookPart**(*parent=None*)
Bases: *ly.music.items.Container*

A bookpart { ... } construct.

class ly.music.items.**Change**(*parent=None*)
Bases: *ly.music.items.Translator*

A change music expression.

class ly.music.items.**Chord**(*parent=None*)
Bases: *ly.music.items.Durable*, *ly.music.items.Container*

class ly.music.items.**ChordItem**(*parent=None*)
Bases: *ly.music.items.Item*

An item inside a ChordSpecifier, e.g. a number or modifier.

class ly.music.items.**ChordMode**(*parent=None*)
Bases: *ly.music.items.InputMode*

A chordmode or chords expression.

class ly.music.items.**ChordSpecifier**(*parent=None*)
Bases: *ly.music.items.Item*

Chord specifications after a note in chord mode.

Has children of Note or ChordItem class.

class ly.music.items.**Clef**(*parent=None*)
Bases: *ly.music.items.Item*

A clef item.

specifier()

class ly.music.items.**Command**(*parent=None*)
Bases: *ly.music.items.Item*

A LilyPond command.

class ly.music.items.**Container**(*parent=None*)
Bases: *ly.music.items.Item*

An item having a list of child items.

class ly.music.items.**Context**(*parent=None*)
Bases: *ly.music.items.Translator*, *ly.music.items.Music*

A new or context music expression.

class ly.music.items.**Document**(*doc*)
Bases: *ly.music.items.Item*

A toplevel item representing a ly.document.Document.

get_included_document_node(*node*)
Return a Document for the Include node. May return None.

get_music(*filename*)
Return the music Document for the specified filename.

This implementation loads a ly.document.Document using utf-8 encoding. Inherit from this class to implement other loading mechanisms or caching.

iter_music(*node=None*)
Iter over the music, following references to other assignments.

music_events_til_position(*position*)
Return a list of tuples.

Every tuple is a (parent, nodes, scaling). If an empty list is returned, there is no music expression at this position.

node(*position*, *depth=-1*)
Return the node at or just before the specified position.

resolve_filename(*filename*)
Resolve filename against our document and include_path.

substitute_for_node(*node*)
Returns a node that replaces the specified node (e.g. in music).

For example: a variable reference returns its value. Returns nothing if the node is not substitutable. Returns the node itself if it was substitutable, but the substitution failed.

time_length(*start*, *end*)
Return the length of the music between start and end positions.

Returns None if start and end are not in the same expression.

time_position(*position*)
Return the time position in the music at the specified cursor position.

The value is a fraction. If None is returned, we are not in a music expression.

class ly.music.items.**DrumMode**(*parent=None*)
Bases: *ly.music.items.InputMode*

A drummode or drums expression.

class ly.music.items.**DrumNote**(*parent=None*)
Bases: *ly.music.items.Durable*

class ly.music.items.**Durable**(*parent=None*)
Bases: *ly.music.items.Item*

An Item that has a musical duration, in the duration attribute.

duration = (0, 1)

events(*e*, *time*, *scaling*)
Let the event.Events instance handle the events. Return the time.

length()
Return the musical duration (our base * our scaling).

class ly.music.items.**Duration**(*parent=None*)
Bases: *ly.music.items.Item*

A written duration

class ly.music.items.**Dynamic**(*parent=None*)
Bases: *ly.music.items.Item*

Any dynamic symbol.

class ly.music.items.**FigureMode**(*parent=None*)
Bases: *ly.music.items.InputMode*

A figuremode or figures expression.

class ly.music.items.**Grace**(*parent=None*)
Bases: *ly.music.items.Music*

Music that has grace timing, i.e. 0 as far as computation is concerned.

events(*e*, *time*, *scaling*)
Let the event.Events instance handle the events. Return the time.

preceding(*node=None*)
Return a two-tuple (nodes, scaling).

The nodes are the nodes in time before the node (which must be a child), and the scaling is 0 for (because we have grace notes).

If node is None, all nodes that would precede a fictive node at the end are returned.

class ly.music.items.**Header**(*parent=None*)
Bases: *ly.music.items.Container*

A header { ... } construct.

class ly.music.items.**Include**(*parent=None*)
Bases: *ly.music.items.Item*

An include command (not changing the language).

filename()
Returns the filename.

class ly.music.items.**InputMode**(*parent=None*)
Bases: *ly.music.items.Music*

Base class for inputmode-changing commands.

class ly.music.items.**Item**(*parent=None*)
Bases: *ly.node.WeakNode*

Represents any item in the music of a document.

This can be just a token, or an interpreted construct such as a note, rest or sequential or simultaneous construct , etc.

Some Item instances just have one responsible token, but others have a list or tuple to tokens.

An Item also has a pointer to the Document it originates from.

document = None

end_position()
Return the end position of this node.

events(*e*, *time*, *scaling*)
Let the event.Events instance handle the events. Return the time.

has_output(*_seen_docs=None*)
Return True if this node has toplevel music, markup, book etc.

I.e. returns True when LilyPond would likely generate output. Usually you'll call this method on a Document, Score, BookPart or Book node.

You should not supply the _seen_docs argument; it is used internally to avoid traversing recursively nested include files.

iter_toplevel_items()
Yield the toplevel items of our Document node in backward direction.

Iteration starts with the node just before the node "self" is a descendant of.

iter_toplevel_items_include()
Same as iter_toplevel_items(), but follows include commands.

length()
Return the musical duration.

music_children(*depth=-1*)
Yield all the children that are new music expressions

(i.e. that are inside other constructions).

music_parent()
Walk up the parent tree until Music is found; return the outermost Music node.

Returns None is the node does not belong to any music expression (e.g. a toplevel Markup or Scheme object).

plaintext()
Return a plaintext value for this node.

This only makes sense for items like Markup or String. For other types, an empty string is returned

position = -1

token = None

tokens = ()

class ly.music.items.**KeySignature**(*parent=None*)
Bases: *ly.music.items.Item*

A key pitch mode command.

mode()
The mode, e.g. "major", "minor", etc.

pitch()
The ly.pitch.Pitch that denotes the pitch.

class ly.music.items.**Keyword**(*parent=None*)
Bases: *ly.music.items.Item*

A LilyPond keyword.

class ly.music.items.**Language**(*parent=None*)
Bases: *ly.music.items.Item*

A command (language or certain include commands) that changes the pitch language.

language = None

class ly.music.items.**Layout**(*parent=None*)
Bases: *ly.music.items.Container*

A layout { ... } construct.

class ly.music.items.**LayoutContext**(*parent=None*)
Bases: *ly.music.items.Container*

A context { ... } construct within Layout or Midi.

class ly.music.items.**LyricItem**(*parent=None*)
Bases: *ly.music.items.Item*

Another lyric item (skip, extender, hyphen or tie).

class ly.music.items.**LyricMode**(*parent=None*)
Bases: *ly.music.items.InputMode*

A lyricmode, lyrics or addlyrics expression.

class ly.music.items.**LyricText**(*parent=None*)
Bases: *ly.music.items.Durable*

A lyric text (word, markup or string), with a Duration.

class ly.music.items.**LyricsTo**(*parent=None*)
Bases: *ly.music.items.InputMode*

A lyricsto expression.

context_id()

class ly.music.items.**Markup**(*parent=None*)
Bases: *ly.music.items.Item*

A command starting markup (markup, -lines and -list).

plaintext()
Return the plain text value of this node.

class ly.music.items.**MarkupCommand**(*parent=None*)
Bases: *ly.music.items.Item*

A markup command, such as italic etc.

plaintext()
Return the plain text value of this node.

class ly.music.items.**MarkupList**(*parent=None*)
Bases: *ly.music.items.Item*

The group of markup items inside { and }. NOTE: *not* a markuplist.

plaintext()
Return the plain text value of this node.

class ly.music.items.**MarkupScore**(*parent=None*)
Bases: *ly.music.items.Item*

A score inside Markup.

class ly.music.items.**MarkupUserCommand**(*parent=None*)
Bases: *ly.music.items.Item*

A user-defined markup command

name()
Return the name of this user command (without the leading backslash).

value()
Find the value assigned to this variable.

class ly.music.items.**MarkupWord**(*parent=None*)
Bases: *ly.music.items.Item*

A MarkupWord token.

plaintext()

class ly.music.items.**Midi**(*parent=None*)
Bases: *ly.music.items.Container*

A midi { ... } construct.

class ly.music.items.**Music**(*parent=None*)
Bases: *ly.music.items.Container*

Any music expression, to be inherited of.

events(*e*, *time*, *scaling*)
Let the event.Events instance handle the events. Return the time.

length()
Return the musical duration.

preceding(*node=None*)
Return a two-tuple (nodes, scaling).

The nodes are the nodes in time before the node (which must be a child), and the scaling is the scaling this node applies (normally 1).

If node is None, all nodes that would precede a fictive node at the end are returned.

class ly.music.items.**MusicList**(*parent=None*)
Bases: *ly.music.items.Music*

A music expression, either << >> or { }.

events(*e*, *time*, *scaling*)
Let the event.Events instance handle the events. Return the time.

preceding(*node=None*)
Return a two-tuple (nodes, scaling).

The nodes are the nodes in time before the node (which must be a child), and the scaling is the scaling this node applies (normally 1).

If node is None, all nodes that would precede a fictive node at the end are returned.

simultaneous = False

class ly.music.items.**Note**(*parent=None*)
Bases: *ly.music.items.Durable*

A Note that has a ly.pitch.Pitch

accidental_token = None

octave_token = None

octavecheck_token = None

pitch = None

class ly.music.items.**NoteMode**(*parent=None*)
Bases: *ly.music.items.InputMode*

A notemode or notes expression.

class ly.music.items.**Number**(*parent=None*)
Bases: *ly.music.items.Item*

A numerical value, directly entered.

value()

class ly.music.items.**Override**(*parent=None*)
Bases: *ly.music.items.Item*

An override command.

context()

grob()

class ly.music.items.**Paper**(*parent=None*)
Bases: *ly.music.items.Container*

A paper { ... } construct.

class ly.music.items.**PartCombine**(*parent=None*)
Bases: *ly.music.items.Music*

The partcombine command with 2 music arguments.

events(*e*, *time*, *scaling*)
Let the event.Events instance handle the events. Return the time.

preceding(*node=None*)
Return a two-tuple (nodes, scaling).

The nodes are the nodes in time before the node (which must be a child), and the scaling is the scaling this node applies (normally 1).

If node is None, all nodes that would precede a fictive node at the end are returned.

class ly.music.items.**Partial**(*parent=None*)
Bases: *ly.music.items.Item*

A partial command.

duration = (0, 1)

partial_length()
Return the duration given as argument as a Fraction.

class ly.music.items.**PathItem**(*parent=None*)
Bases: *ly.music.items.Item*

An item in the path of an override or revert command.

class ly.music.items.**PhrasingSlur**(*parent=None*)
Bases: *ly.music.items.Item*

A (or).

event = None

class ly.music.items.**PipeSymbol**(*parent=None*)
Bases: *ly.music.items.Item*

A pipe symbol: |

class ly.music.items.**Postfix**(*parent=None*)
Bases: *ly.music.items.Item*

Any item that is prefixed with a _, - or ^ direction token.

class ly.music.items.**Q**(*parent=None*)
Bases: *ly.music.items.Durable*

class ly.music.items.**Relative**(*parent=None*)
Bases: *ly.music.items.Music*

A relative music expression. Has one or two children (Note, Music).

class ly.music.items.**Repeat**(*parent=None*)
Bases: *ly.music.items.Music*

A repeat expression.

events(*e*, *time*, *scaling*)
Let the event.Events instance handle the events. Return the time.

repeat_count()

specifier()

class ly.music.items.**Rest**(*parent=None*)
Bases: *ly.music.items.Durable*

class ly.music.items.**Revert**(*parent=None*)
Bases: *ly.music.items.Item*

A revert command.

context()

grob()

class ly.music.items.**Scaler**(*parent=None*)
Bases: *ly.music.items.Music*

A music construct that scales the duration of its contents.

In the numerator and denominator attributes the values specified for LilyPond are stored, e.g. with times 3/2 { c d e }, the numerator is integer 3 and the denominator is integer 2. Note that for tuplet and times the meaning of these numbers is reversed.

The algebraic scaling is stored in the scaling attribute.

denominator = 0

events(*e*, *time*, *scaling*)
Let the event.Events instance handle the events. Return the time.

numerator = 0

preceding(*node=None*)
Return a two-tuple (nodes, scaling).

The nodes are the nodes in time before the node (which must be a child), and the scaling is the scaling this node applies.

If node is None, all nodes that would precede a fictive node at the end are returned.

scaling = 1

class ly.music.items.**Scheme**(*parent=None*)
Bases: *ly.music.items.Item*

A Scheme expression inside LilyPond.

get_fraction()
A basic way to get one (may be fractional) numerical value.

get_int()
A basic way to get one integer value.

get_list_ints()
A basic way to get a list of integer values.

get_ly_make_moment()
A basic way to get a ly:make-moment fraction.

get_pair_ints()
Very basic way to get two integers specified as a pair.

get_string()
A basic way to get a quoted string value (without the quotes).

plaintext()
A crude way to get the plain text in this node.

class ly.music.items.**SchemeItem**(*parent=None*)
Bases: *ly.music.items.Item*

Any scheme token.

class ly.music.items.**SchemeLily**(*parent=None*)
Bases: *ly.music.items.Container*

A music expression inside #{ and #}.

class ly.music.items.**SchemeList**(*parent=None*)
Bases: *ly.music.items.Container*

A (...) expression.

class ly.music.items.**SchemeQuote**(*parent=None*)
Bases: *ly.music.items.Item*

A ‘ in scheme.

class ly.music.items.**Score**(*parent=None*)
Bases: *ly.music.items.Container*

A score { ... } construct.

class ly.music.items.**Set**(*parent=None*)
Bases: *ly.music.items.Item*

A set command.

context()
The context, if specified.

property()
The property.

value()
The value, given as argument. This is simply the child element.

class ly.music.items.**Skip**(*parent=None*)
Bases: *ly.music.items.Durable*

class ly.music.items.**Slur**(*parent=None*)
Bases: *ly.music.items.Item*

A (or).

event = None

class ly.music.items.**String**(*parent=None*)
Bases: *ly.music.items.Item*

A double-quoted string.

plaintext()
Return the plaintext value of this string, without escapes and quotes.

value()

class ly.music.items.**StringTuning**(*parent=None*)
Bases: *ly.music.items.Item*

A stringTuning command (with a chord as argument).

class ly.music.items.**Tag**(*parent=None*)
Bases: *ly.music.items.Music*

A tag, keepWithTag or removeWithTag command.

events(*e*, *time*, *scaling*)
Let the event.Events instance handle the events. Return the time.

preceding(*node=None*)
Return a two-tuple (nodes, scaling).

The nodes are the nodes in time before the node (which must be a child), and the scaling is the scaling this node applies (normally 1).

If node is None, all nodes that would precede a fictive node at the end are returned.

class ly.music.items.**Tempo**(*parent=None*)
Bases: *ly.music.items.Item*

duration = (0, 1)

fraction()
Return the note value as a fraction given before the equal sign.

tempo()
Return a list of integer values describing the tempo or range.

text()
Return the text, if set. Can be Markup, Scheme, or String.

class ly.music.items.**Tie**(*parent=None*)
Bases: *ly.music.items.Item*

A tie.

class ly.music.items.**TimeSignature**(*parent=None*)
Bases: *ly.music.items.Item*

A time command.

beatstructure()
The scheme expressions denoting the beat structure, if specified.

fraction()
The lower number as a Fraction (e.g. for 3/2 it returns 1/2).

measure_length()
The length of one measure in this time signature as a Fraction.

numerator()
The upper number (e.g. for 3/2 it returns 3).

class ly.music.items.**Token**(*parent=None*)
Bases: *ly.music.items.Item*

Any token that is not otherwise recognized

class ly.music.items.**Translator**(*parent=None*)
Bases: *ly.music.items.Item*

Base class for a change, new, or context music expression.

context()

context_id()
The context id, if specified after an equal sign.

class ly.music.items.**Transpose**(*parent=None*)
Bases: *ly.music.items.Music*

A transpose music expression. Has normally three children (Note, Note, Music).

class ly.music.items.**Tremolo**(*parent=None*)
Bases: *ly.music.items.Item*

A tremolo item ":". The duration attribute is a tuple (base, scaling).

duration = (0, 1)

class ly.music.items.**Tweak**(*parent=None*)
Bases: *ly.music.items.Item*

A tweak command.

class ly.music.items.**Unpitched**(*parent=None*)
Bases: *ly.music.items.Durable*

A "note" without pitch, just a standalone duration.

pitch = None

class ly.music.items.**Unset**(*parent=None*)
Bases: *ly.music.items.Item*

An unset command.

context()
The context, if specified.

property()
The property.

class ly.music.items.**UserCommand**(*parent=None*)
Bases: *ly.music.items.Music*

A user command, most probably referring to music.

events(*e*, *time*, *scaling*)
Let the event.Events instance handle the events. Return the time.

name()
Return the name of this user command (without the leading backslash).

value()
Find the value assigned to this variable.

class ly.music.items.**Version**(*parent=None*)
Bases: *ly.music.items.Item*

A version command.

version()
The version as a tuple of ints.

version_string()
The version as a string.

class ly.music.items.**VoiceSeparator**(*parent=None*)
Bases: *ly.music.items.Item*

A voice separator: \

class ly.music.items.**With**(*parent=None*)
Bases: *ly.music.items.Container*

A with ... construct.

ly.music.read module

The Reader is used to construct a tree structure of (musical and other) items in a LilyPond document. The item types that are used are in the items module.

You instantiate a Reader with a ly.document.Source that reads tokens from any (part of a) ly.document.Document.

Whitespace and comments are left out.

All nodes (instances of Item) have a 'position' attribute that indicates where the item starts in the source text. Almost all items have the token that starts the expression in the 'token' attribute and possibly other tokens in the 'tokens' attribute, as a tuple.

The 'end_position()' method returns the position where the node (including its child nodes) ends.

class ly.music.read.**Reader**(*source*)
Bases: object

Reads tokens from a Source and builds a meaningful tree structure.

add_bracketed(*item*, *source*)
Append the arguments between brackets to the item.

Returns True if that succeeded, else False.

add_duration(*item*, *token=None*, *source=None*)
Add a duration attribute to the item.

When there are tokens, a Duration item is also appended to the item.

add_markup_arguments(*item*)
Add markup arguments to the item.

consume(*last_token=None*)
Yield the tokens from our source until a parser is exit.

If last_token is given, it is called with the last token that is read.

factory(*cls*, *token=None*, *consume=False*, *position=None*)
Create Item instance for token.

If consume is True, consume()s the source into item.tokens. If you don't specify a token, you must specify the position (>= 0).

handle_absolute(*t*, *source*)
handle absolute

handle_after_grace(*t*, *source*)
handle afterGrace

handle_alternative(*t*, *source*)
handle alternative

handle_articulation(*t*, *source=None*)
handle Articulation

handle_beam(*t*, *source=None*)
handle Beam

handle_bracketed(*t*, *source*)
handle with, layout, paper, context, book, bookpart, score, midi, header

handle_chord_start(*t*, *source*)
handle ChordStart

handle_clef(*t*, *source*)
handle clef

handle_direct_items(*t*, *source*)
Tokens that directly translate to an Item.

handle VoiceSeparator, Tie, Dynamic, PipeSymbol

handle_direction(*t*, *source*)
handle Direction

handle_grace(*t*, *source*)
handle grace, acciaccatura, appoggiatura, slashedGrace

handle_include(*t*, *source*)
handle include

handle_inputmode(*t*, *source*)
handle notes, figuremode, notemode, drums, chords, drummode, chordmode, figures

handle_key(*t*, *source*)
handle key

handle_language(*t*, *source*)
handle language

handle_length(*t*, *source*)
handle Length

handle_lyricmode(*t*, *source*)
handle addlyrics, lyricmode, lyrics, oldaddlyrics, lyricsto

handle_markup(*t*, *source=None*)
handle markup, markuplist, markuplines

handle_markup_command(*t*)
handle MarkupCommand

handle_markup_open_bracket(*t*)
handle OpenBracketMarkup

handle_markup_score(*t*)
handle MarkupScore

handle_markup_user_command(*t*)
handle MarkupUserCommand

handle_markup_word(*t*)
handle MarkupWord

handle_music_item(*t*, *source*)
handle MusicItem

handle_music_list(*t*, *source*)
handle OpenBracket, OpenSimultaneous, SimultaneousOrSequentialCommand

handle_name(*t*, *source*)
handle Name, ContextProperty

handle_number_class(*t*, *source=None*)
handle DecimalValue, IntegerValue, Fraction

handle_override(*t*, *source*)
handle override

handle_partcombine(*t*, *source=None*)
handle partcombine

handle_partial(*t*, *source*)
handle partial

handle_relative(*t*, *source*)
handle relative

handle_repeat(*t*, *source*)
handle repeat

handle_revert(*t*, *source*)
handle revert

handle_scaler(*t*, *source*)
handle times, tuplet, scaleDurations

handle_scheme_lilypond_start(*t*)
handle LilyPondStart

handle_scheme_open_parenthesis(*t*)
handle OpenParen

handle_scheme_quote(*t*)
handle Quote

handle_scheme_start(*t*, *source=None*)
handle SchemeStart

handle SchemeStart

handle_scheme_token(*t*)
handle Dot, Bool, Char, Word, Number, Fraction, Float

handle_set(*t*, *source*)
handle set

handle_slurs(*t*, *source=None*)
handle Slur

handle_string_start(*t*, *source=None*)
handle StringStart

handle StringStart

handle StringStart

handle_string_tuning(*t*, *source*)
handle stringTuning

handle_tag(*t*, *source*)
handle tag, keepWithTag, removeWithTag, appendToTag, pushToTag

handle_tempo(*t*, *source*)
handle tempo

handle_time(*t*, *source*)
handle time

handle_translator(*t*, *source*)
handle new, context, change

handle_transpose(*t*, *source*)
handle transpose

handle_tweak(*t*, *source*)
handle tweak

handle_unset(*t*, *source*)
handle unset

handle_variable_assignment(*t*, *source*)
handle PaperVariable, LayoutVariable, HeaderVariable, UserVariable

handle_version(*t*, *source*)
handle version

read(*source=None*)
Yield Item instances reading from source.

read_assignment(*t*)
Read an assignment from the variable name. May return None.

read_chord_specifier(*t*, *source=None*)
Read stuff behind notes in chordmode.

handle ChordSeparator

read_command(*t*, *source*)
Read the rest of a command given in t from the source.

handle Command

read_item(*t*, *source=None*)
Return one Item that starts with token t. May return None.

read_keyword(*t*, *source*)
Read the rest of a keyword given in t from the source.

handle Keyword

read_lyric_item(*t*)
Read one lyric item. Returns None for tokens it does not handle.

read_markup(*t*)
Read LilyPond markup (recursively).

read_music_item(*t*, *source*)
Read one music item (note, rest, s, skip, or q) from t and source.

read_scheme(*t*)
Return a Scheme item from the token t.

read_scheme_item(*t*)
Reads a Scheme expression (just after the # in LilyPond mode).

read_tremolo(*t*, *source=None*)
Read a tremolo.

handle TremoloColon

read_user_command(*t*, *source*)
Read a user command, this can be a variable reference.

handle UserCommand

set_language(*lang*)
Changes the pitch name language to use.

Called internally when language or include tokens are encountered with a valid language name/file.

Sets the language attribute to the language name.

test_music_list(*t*)
Test whether a music list ({ ... }, << ... >>, starts here.

Also handles simultaneous { ... } and sequential { ... } correctly. These obscure commands are not even highlighted by lex, but they exist in LilyPond... simultaneous { ... } is like << ... >> but sequential << ... >> just behaves like << ... >>

Returns a two-tuple(item; iterable), both may be None. If item is not None, it can be either a UserCommand or a MusicList. If iterable is None, the item is a UserCommand (namely simultaneous or sequential, but not followed by a { or <<); else the item is a MusicList, and the iterable should be read fully to get all the tokens inside the MusicList. If item is None, there is no MusicList and no token is read.

This way you can handle the { ... } and << ... >> transparently in every input mode.

class ly.music.read.**dispatcher**
Bases: object

Decorator creator to dispatch commands, keywords, etc. to a method.

method(*token*)
The registered method to call for the token. May return None.

read_arg(*a*)

class ly.music.read.**dispatcher_class**
Bases: *ly.music.read.dispatcher*

Decorator creator to dispatch the class of a token to a method.

method(*token*)
The registered method to call for the token's class. May return None.

read_arg(*a*)

ly.music.read.**skip**(*source*, *what=(<class 'ly.lex._token.Space'>, <class 'ly.lex._token.Comment'>)*)
Yield tokens from source, skipping items of classes specified in what.

By default, comments and whitespace are skipped.

ly.musicxml package

Module contents

MusicXML functionality.

This subpackage is created to convert LilyPond text to MusicXML with the help of the tree structure created by ly.music. But it is constructed in such a way that you can use some of the submodules for generic MusicXML creation and manipulation.

ly.musicxml.**writer**()
Convert LilyPond text to MusicXML

Example:

```
import ly.musicxml
e = ly.musicxml.writer()
e.parse_text(lilypond_text)
xml = e.musicxml()
xml.write(filename)          # or: xml.tostring()
# xml.tree is the ElementTree xml tree.
```

Submodules

ly.musicxml.create_musicxml module

Uses xml.etree to create a Music XML document.

Example:

```
musxml = create_musicxml.CreateMusicXML()
musxml.create_part()
musxml.create_measure(divs=1)
musxml.new_note('C', 4, 'whole', 4)
xml = musxml.musicxml()
xml.write(filename)
```

class ly.musicxml.create_musicxml.**CreateMusicXML**
Creates the XML nodes according to the Music XML standard.

add_accidental(*alter*, *caut=False*, *parenth=False*)
Create accidental.

add_articulations()
Common function for creating all types of articulations.

add_backup(*duration*)

add_bar_style(*multirest=None*)

add_barline(*bl_type*, *repeat=None*)

add_beam(*nr*, *beam_type*)
Add beam.

add_chord()

add_clef(*sign*, *line*, *nr=0*, *oct_ch=0*)

add_creator(*creator*, *name*)
Add creator to score info.

add_direction(*pos=u'above'*)

add_dirwords(*words*)
Add words in direction, e. g. a tempo mark.

add_div_duration(*divdur*)
Create new duration

add_divisions(*div*)

add_dot()
Create a dot

add_duration_type(*durtype*)
Create new type

add_dynamic_dashes(*text*)
Add dynamics dashes.

add_dynamic_mark(*dyn*)
Add specified dynamic mark.

add_dynamic_text(*text*)
Add dynamic text.

add_dynamic_wedge(*wedge_type*)
Add dynamic wedge/hairpin.

add_fingering(*finger_nr*)

add_gliss(*linetype*, *endtype*, *nr*)

add_grace(*slash*)
Create grace node

add_key(*key*, *mode*)

add_lyric(*txt*, *syll*, *nr*, *ext=False*)
Add lyric element.

add_mark(*mark*)
Add rehearsal mark in direction

add_metron_dir(*unit*, *beats*, *dots*)

add_mordent()

add_named_artic(*artic*)
Add articulation with specified name.

add_named_notation(*notate*)
Fermata, etc.

add_notations()

add_octave_shift(*plac*, *octdir*, *size*)
Add octave shift.

add_ornaments()

add_pitch(*step*, *alter*, *octave*)
Create new pitch.

add_prall()

add_rest()
Create rest.

add_rest_w_pos(*step*, *octave*)
Create rest with display position.

add_rights(*rights*)
Add rights to score info.

add_skip(*duration*, *forward=True*)

add_slur(*nr*, *sl_type*)
Add slur.

add_sound_dir(*midi_tempo*)

add_staff(*staff*)

add_staves(*staves*)

add_technical()

add_tie(*tie_type*)
Create node tie (used for sound of tie)

add_tied(*tie_type*)
Create node tied (used for notation of tie)

add_time(*timesign*)

add_time_modify(*fraction*)
Create time modification

add_tremolo(*trem_type*, *lines*)

add_trill()

add_tuplet_type(*nr*, *ttype*, *actnr=0*, *acttype=u''*, *normnr=0*, *normtype=u''*)
Create tuplet with type attribute

add_turn()

add_unpitched(*step*, *octave*)
Create unpitched.

add_voice(*voice*)

add_wavyline(*end_type*)

adjust_time_modify(*timemod_node*, *fraction*)
Adjust existing time-modification node.

change_div_duration(*newdura*)
Set new duration when tuplet

create_bar_attr()
Create node attributes

create_measure(*pickup=False*, ***bar_attrs*)
Create new measure

create_new_node(*parentnode*, *nodename*, *txt*)
The Music XML language is extensive. This function can be used to create a non basic node not covered elsewhere in this script.

TODO: add attributes

create_note()
Create new note.

create_part(*name=u'unnamed'*, *abbr=False*, *midi=False*)
Create a new part

create_partgroup(*gr_type*, *num*, *name=None*, *abbr=None*, *symbol=None*)
Create a new part group.

create_score_info(*tag*, *info*, *attr={}*)
Create score info.

create_tempo(*words*, *metronome*, *sound*, *dots*)

create_title(*title*)
Create score title.

get_time_modify()
Check if time-modification node already exists.

musicxml(*prettyprint=True*)

new_adv_ornament(*ornament*, *args*)
Add more complex ornament.

new_articulation(*artic*)
Add specified articulation.

new_bar_attr(*clef=0*, *mustime=0*, *key=None*, *mode=0*, *divs=0*, *multirest=0*)
Create all bar attributes set.

new_note(*step*, *octave*, *durtype*, *divdur*, *alter=0*, *acc_token=0*, *voice=1*, *dot=0*, *chord=0*, *grace=(0, 0)*, *stem_dir=0*)
Create all nodes needed for a normal note.

new_rest(*duration*, *durtype*, *pos*, *dot*, *voice*)
Create all nodes needed for a rest.

new_simple_ornament(*ornament*)
Add specified ornament.

new_system(*force_break*)

new_unpitched_note(*step*, *octave*, *durtype*, *divdur*, *voice=1*, *dot=0*, *chord=0*, *grace=(0, 0)*)
Create all nodes needed for an unpitched note.

set_stem_dir(*dir*)

tie_note(*tie_type*)

tuplet_note(*fraction*, *bs*, *ttype*, *nr*, *divs*, *atyp=u''*, *ntyp=u''*)
Convert current note to tuplet

class ly.musicxml.create_musicxml.**MusicXML**(*tree*)
Bases: object

Represent a generated MusicXML tree.

indent(*indent=u' '*)
Add indent and linebreaks to the created XML tree.

tostring(*encoding=u'UTF-8'*)
Output etree as a XML document.

write(*file*, *encoding=u'UTF-8'*, *doctype=True*)
Write XML to a file (file obj or filename).

ly.musicxml.create_musicxml.**get_tag_index**(*node*, *tag*)
Return the (first) index of tag in node.

If tag is not found, -1 is returned.

ly.musicxml.ly2xml_mediator module

The information of the parsed source is organised into an object structure with help of the classes in ly.musicxml.xml_objs.

class ly.musicxml.ly2xml_mediator.**Mediator**
Help class that acts as mediator between the ly source parser and the XML creating modules.

add_break()

add_snippet(*snippet_name*)
Adds snippet to previous barlist. A snippet can be shorter than a full bar, so this can also mean continuing a previous bar.

add_staff_id(*staff_id*)

add_to_bar(*obj*)

bijective(*n*)
encodes an int to a sequence of letters

calc_tupl_den(*tfraction*, *length*)
Calculate the tuplet denominator from fraction and duration of tuplet.

change_group_bracket(*system_start*)

change_to_tuplet(*tfraction*, *ttype*, *nr*, *length=None*)
Change the current note into a tuplet note.

change_tuplet_type(*index*, *newtype*)

check_current_note(*rel=False*, *rest=False*, *is_unpitched=False*)
Perform checks common for all new notes and rests.

check_divs()
The new duration is checked against current divisions

check_duration(*rest*)
Check the duration for the current note.

check_lyrics(*voice_id*)
Check the finished lyrics section and merge it into the referenced voice.

check_name(*name*, *nr=1*)

check_part()
Adds the latest active section to the part.

check_score()
Check score

If no part were created, place first variable (fallback) as part.

Apply the latest global section.

check_simultan()
Check done after simultanoues (<< >>) section.

check_voices()
Checks active sections. The two latest created are merged. Also checks for empty sections.

check_voices_by_nr()
Used for snippets. Merges all active snippets created after the stored voice number.

chord_end()
Actions when chord is parsed.

clear_chord()

close_group()

copy_barnote_basics(*bar_note*)
Create a copy of a xml_objs.BarNote.

copy_prev_chord(*duration*)

create_barline(*bl*)

create_barnote_from_note(*note*)
Create a xml_objs.BarNote from ly.music.items.Note.

create_unpitched(*unpitched*)
Create a xml_objs.Unpitched from ly.music.items.Unpitched.

do_action_onnext(*note*)
Perform the stored action on the next note.

duration_from_tokens(*tokens*)
Calculate dots and multibar rests from tokens.

end_gliss(*note*, *line*)

get_first_var()

get_part_by_id(*pid*, *partholder=None*)

get_var_byname(*name*)

increase_bar_dura(*duration*)

new_articulation(*art_token*)
An articulation, fingering, string number, or other symbol.

Grouped as articulations, ornaments, technical and others.

new_bar(*fill_prev=True*)

new_chord(*note*, *duration=None*, *rel=False*, *chord_base=True*)

new_chord_grace(*slash=0*)

new_chordbase(*note*, *duration*, *rel=False*)

new_chordnote(*note*, *rel*)

new_clef(*clefname*)

new_duration_token(*token*, *tokens*)

new_dynamics(*dynamics*)

new_gliss(*line=None*)

new_grace(*slash=0*)

new_group()

new_header_assignment(*name*, *value*)
Distributing header information.

new_iso_dura(*note*, *rel=False*, *is_unpitched=False*)
Isolated durations in music sequences.

An isolated duration in LilyPond is rendered as a normal note but the pitch information is missing and has to be filled in by some other means, usually by the previous pitch. (RhythmicStaff is an exception since it ignores specified pitches anyway).

new_key(*key_name*, *mode*)

new_lyric_nr(*num*)

new_lyric_section(*name*, *voice_id*)

new_lyrics_item(*item*)

new_lyrics_text(*txt*)

new_mark(*num_mark=None*)

new_note(*note*, *rel=False*, *is_unpitched=False*)

new_ottava(*octdiff*)

new_part(*pid=None*, *to_part=None*, *piano=False*)

new_repeat(*rep*)

new_rest(*rest*)

new_section(*name*, *glob=False*)

new_snippet(*name*)

new_tempo(*unit*, *dur_tokens*, *tempo*, *string*)

new_time(*num*, *den*, *numeric=False*)

new_trill_spanner(*end=None*)

new_word(*word*)

note2rest()
Note used as rest position transformed to rest.

part_not_empty()

revert_voicenr()

scale_rest(*bs*)
create multiple whole bar rests

sections = None
default and initial values

set_by_property(*prprty*, *value*, *group=False*)
Generic setter for different properties.

set_groupabbr(*abbr*)

set_groupname(*name*)

set_mult_rest()

set_mult_rest_bar(*dur*)
add multiple-rest attribute to bar

set_octave(*relative*)
Set octave by getting the octave of an absolute note + 3.

set_ottava(*note*, *plac*, *octdir*, *size*)

set_partabbr(*abbr*)

set_partmidi(*midi*)

set_partname(*name*)

set_pickup()

set_relative(*note*)

set_slur(*nr*, *slur_type*)
Set the slur start or stop for the current note.

set_staffnr(*staffnr*, *staff_id=None*)

set_tremolo(*trem_type=u'single'*, *duration=0*, *repeats=0*)

set_tuplspan_dur(*token=None*, *tokens=None*, *fraction=None*)
Catch duration set by the tupletSpannerDuration property.

Set the fraction directly or calculate it from tokens.

set_voicenr(*command=None*, *add=False*, *nr=0*, *piano=0*)

stem_direction(*direction*)

tie_to_next()

unset_tuplspan_dur()
Reset tuplet duration sum and tuplet spanner duration.

ly.musicxml.ly2xml_mediator.**artic_token2xml_name**(*art_token*)
From Articulations in ly.music.items. Grouped as articulations, ornaments and others.

To add an articulation look up the name or abbreviation in LilyPond and the corresponding node name in musicXML. Add it to the python dictionary below.

ly.musicxml.ly2xml_mediator.**calc_trem_dur**(*repeats*, *base_scaling*, *duration*)
Calculate tremolo duration from number of repeats and initial duration.

ly.musicxml.ly2xml_mediator.**clefname2clef**(*clefname*)
To add a clef look up the clef name in LilyPond and the corresponding definition in musicXML. Add it to the python dictionary below.

ly.musicxml.ly2xml_mediator.**durval2type**(*durval*)
Convert LilyPond duration to MusicXML duration type.

ly.musicxml.ly2xml_mediator.**getNoteName**(*index*)

ly.musicxml.ly2xml_mediator.**get_fifths**(*key*, *mode*)

ly.musicxml.ly2xml_mediator.**get_group_symbol**(*lily_sys_start*)

ly.musicxml.ly2xml_mediator.**get_line_style**(*style*)

ly.musicxml.ly2xml_mediator.**get_mult**(*num*, *den*)

ly.musicxml.ly2xml_mediator.**get_voice**(*c*)

ly.musicxml.ly2xml_mediator.**get_xml_alter**(*alter*)
Convert alter to the specified format, i e int if it's int and float otherwise. Also multiply with 2.

ly.musicxml.lymus2musxml module

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.

class ly.musicxml.lymus2musxml.**End**(*node*)
Extra class that gives information about the end of Container elements in the node list.

class ly.musicxml.lymus2musxml.**ParseSource**
creates the XML-file from the source code according to the Music XML standard

Articulation(*art*)
An articulation, fingering, string number, or other symbol.

Assignment(*a*)
Variables should already have been substituted so this need only cover other types of assignments.

Beam(*beam*)

Change(*change*)
A change music expression. Changes the staff number.

Chord(*chord*)

ChordMode(*chordmode*)
A chordmode or chords expression.

Clef(*clef*)
Clef clef

Command(*command*)
bar, rest etc

Context(*context*)
context

DrumMode(*drummode*)
A drummode or drums expression.

If the shorthand form drums is found, DrumStaff is implicit.

DrumNote(*drumnote*)
A note in DrumMode.

Duration(*duration*)
A written duration

Dynamic(*dynamic*)
Any dynamic symbol.

End(*end*)

FigureMode(*figmode*)
A figuremode or figures expression.

Grace(*grace*)

KeySignature(*key*)

LyricItem (*lyrics_item*)
Another lyric item (skip, extender, hyphen or tie).

LyricMode (*lyricmode*)
A lyricmode, lyrics or addlyrics expression.

LyricText (*lyrics_text*)
A lyric text (word, markup or string), with a Duration.

LyricsTo (*lyrics_to*)
A lyricsto expression.

Markup (*markup*)

MarkupList (*markuplist*)

MarkupWord (*markupWord*)

MusicList (*musicList*)

Note (*note*)
notename, e.g. c, cis, a bes ...

NoteMode (*notemode*)
A notemode or notes expression.

Number (*number*)

Override (*override*)
An override command.

Partial (*partial*)

PathItem (*item*)
An item in the path of an override or revert command.

PhrasingSlur (*phrslur*)
A (or).

PipeSymbol (*barcheck*)
PipeSymbol = |

Postfix (*postfix*)

Q (*q*)

Relative (*relative*)
A relative music expression.

Repeat (*repeat*)

Rest (*rest*)
rest, r or R. Note: NOT by command, i.e. rest

Scaler (*scaler*)
times tuplet scaleDurations

Scheme (*scheme*)
A Scheme expression inside LilyPond.

SchemeItem (*item*)
Any scheme token.

SchemeQuote (*quote*)
A ‘ in scheme.

Set(*cont_set*)
A set command.

Skip(*skip*)
invisible rest/spacer rest (s or command skip)

Slur(*slur*)
Slur, '(' = start, ')' = stop.

String(*string*)

Tempo(*tempo*)
Tempo direction, e g '4 = 80'

Tie(*tie*)
tie ~

TimeSignature(*timeSign*)

Tremolo(*tremolo*)
A tremolo item ":".

Unpitched(*unpitched*)
A note without pitch, just a standalone duration.

UserCommand(*usercommand*)
Music variables are substituted so this must be something else.

VoiceSeparator(*voice_sep*)

With(*cont_with*)
A with ... construct.

check_context(*context*, *context_id=None*, *token=u''*)
Check context and do appropriate action (e.g. create new part).

check_note(*note*)
Generic check for all notes, both pitched and unpitched.

check_tuplet()
Generic tuplet check.

find_score_sub(*doc*)
Find substitute for scorenode. Takes first music node that isn't an assignment.

gen_med_caller(*func_name*, **args*)
Call any function in the mediator object.

get_previous_node(*node*)
Returns the nodes previous node or false if the node is first in its branch.

get_score(*node*)
Returns (first) Score node or false if no Score is found.

iter_header(*tree*)
Iter only over header nodes.

iter_score(*scorenode*, *doc*)
Iter over score.

Similarly to items.Document.iter_music user commands are substituted.

Furthermore repeat unfold expressions are unfolded.

look_ahead(*node*, *find_node*)
Looks ahead in a container node and returns True if the search is successful.

look_behind(*node*, *find_node*)
Looks behind on the parent node(s) and returns True if the search is successful.

musicxml(*prettyprint=True*)

parse_document(*ly_doc*, *relative_first_pitch_absolute=False*)
Parse the LilyPond source specified as a ly.document document.

If relative_first_pitch_absolute is set to True, the first pitch in a

elative expression without startpitch is considered to be absolute (LilyPond 2.18+ behaviour).

parse_nodes(*nodes*)
Work through all nodes by calling the function with the same name as the nodes class.

parse_text(*ly_text*, *filename=None*)
Parse the LilyPond source specified as text.

If you specify a filename, it can be used to resolve include commands correctly.

parse_tree(*mustree*)
Parse the LilyPond source as a ly.music node tree.

simple_node_gen(*node*)
Unlike iter_score are the subnodes yielded without substitution.

unfold_repeat(*repeat_node*, *repeat_count*, *doc*)
Iter over node which represent a repeat unfold expression and do the unfolding directly.

ly.musicxml.xml_objs module

Classes that holds information about a musical score, suitable for converting to musicXML.

When the score structure is built, it can easily be used to create a musicXML.

Example:

```
from ly.musicxml import create_musicxml, xml_objs

musxml = create_musicxml.CreateMusicXML()

score = xml_objs.Score()
part = xml_objs.ScorePart()
score.partlist.append(part)
bar = xml_objs.Bar()
part.barlist.append(bar)
ba = xml_objs.BarAttr()
ba.set_time([4,4])
bar.obj_list.append(ba)
c = xml_objs.BarNote('c', 0, 0, (1,1))
c.set_octave(4)
c.set_durtype(1)
bar.obj_list.append(c)

xml_objs.IterateXmlObjs(score, musxml, 1)
xml = musxml.musicxml()
xml.write(filename)
```

class ly.musicxml.xml_objs.**Bar**
Representing the bar/measure. Contains also information about how complete it is.

add(*obj*)

create_backup()
Calculate and create backup object.

has_attr()
Check if bar contains attribute.

has_music()
Check if bar contains music.

inject_voice(*new_voice*, *override=False*)
Adding new voice to bar. Omitting double or conflicting bar attributes as long as override is false. Omitting also bars with only skips.

is_skip(*obj_list=None*)
Check if bar has nothing but skips.

class ly.musicxml.xml_objs.**BarAttr**
object that keep track of bar attributes, e.g. time sign, clef, key etc

add_break(*force_break*)

has_attr()

merge_attr(*barattr*, *override=False*)
Merge in attributes (from another bar). Existing attributes will only be replaced when override is set to true.

set_barline(*bl*)

set_clef(*clef*)

set_key(*muskey*, *mode*)

set_mark(*mark*)

set_multp_rest(*size=0*)

set_tempo(*unit=0*, *unittype=u''*, *beats=0*, *dots=0*, *text=u''*)

set_time(*fractlist*, *numeric=True*)

set_word(*words*)

class ly.musicxml.xml_objs.**BarBackup**(*duration*)
Object that stores duration for backup

class ly.musicxml.xml_objs.**BarMus**(*duration*, *voice=1*)
Common class for notes and rests.

add_dot()

add_other_notation(*other*)

has_attr()

set_dynamics_dashes(*sign*, *before=True*)

set_dynamics_mark(*sign*, *before=True*)

set_dynamics_text(*sign*, *before=True*)

set_dynamics_wedge(*sign*, *before=True*)

set_oct_shift(*plac*, *octdir*, *size*)

set_staff(*staff*)

set_tuplet(*fraction*, *ttype*, *nr*, *acttype=u''*, *normtype=u''*)

class ly.musicxml.xml_objs.**BarNote**(*pitch_note*, *alter*, *accidental*, *duration*, *voice=1*)
Bases: *ly.musicxml.xml_objs.BarMus*

object to keep track of note parameters

add_adv_ornament(*ornament*, *end_type=u'start'*)

add_articulation(*art_name*)

add_fingering(*finger_nr*)

add_lyric(*lyric_list*)

add_ornament(*ornament*)

change_lyric_nr(*index*, *nr*)

change_lyric_syll(*index*, *syll*)

set_duration(*duration*, *durtype=u''*)

set_durtype(*durtype*)

set_gliss(*line*, *endtype=u'start'*, *nr=1*)

set_grace(*slash*)

set_octave(*octave*)

set_slur(*nr*, *slur_type*)

set_stem_direction(*direction*)

set_tie(*tie_type*)

set_tremolo(*trem_type*, *duration=False*)

class ly.musicxml.xml_objs.**BarRest**(*duration*, *voice=1*, *show_type=True*, *skip=False*, *pos=0*)
Bases: *ly.musicxml.xml_objs.BarMus*

object to keep track of different rests and skips

set_duration(*duration*, *durtype=u''*)

set_durtype(*durtype*)

class ly.musicxml.xml_objs.**Dynamics**(*sign*, *before=True*)
Stores information about dynamics.

class ly.musicxml.xml_objs.**DynamicsDashes**(*sign*, *before=True*)
Bases: *ly.musicxml.xml_objs.Dynamics*

Dynamics dashes.

class ly.musicxml.xml_objs.**DynamicsMark**(*sign*, *before=True*)
Bases: *ly.musicxml.xml_objs.Dynamics*

A dynamics mark.

class ly.musicxml.xml_objs.**DynamicsText**(*sign*, *before=True*)
Bases: *ly.musicxml.xml_objs.Dynamics*

A dynamics text.

class ly.musicxml.xml_objs.**DynamicsWedge**(*sign*, *before=True*)
Bases: *ly.musicxml.xml_objs.Dynamics*

A dynamics wedge/hairpin.

class ly.musicxml.xml_objs.**IterateXmlObjs**(*score*, *musxml*, *div*)
A ly.musicxml.xml_objs.Score object is iterated and the Music XML node tree is constructed.

after_note(*obj*)
Xml-nodes after note.

before_note(*obj*)
Xml-nodes before note.

count_duration(*base_scaling*, *divs*)

gener_xml_mus(*obj*)
Nodes generic for both notes and rests.

iterate_bar(*bar*)
The objects in the bar are output to the xml-file.

iterate_part(*part*)
The part is iterated.

iterate_partgroup(*group*)
Loop through a group, recursively if nested.

new_xml_bar_attr(*obj*)
Create bar attribute xml-nodes.

new_xml_note(*obj*)
Create note specific xml-nodes.

new_xml_rest(*obj*)
Create rest specific xml-nodes.

class ly.musicxml.xml_objs.**LyricsSection**(*name*, *voice_id*)
Bases: *ly.musicxml.xml_objs.ScoreSection*

Holds the lyrics information. Will eventually be merged to the corresponding note in the section set by the voice id.

class ly.musicxml.xml_objs.**OctaveShift**(*plac*, *octdir*, *size*)
Class for octave shifts.

class ly.musicxml.xml_objs.**Score**
Object that keep track of a whole score.

debug_score(*attr=[]*)
Loop through score and print all elements for debugging purposes.

Additionally print element attributes by adding them to the argument 'attr' list.

is_empty()
Check if score is empty.

merge_globally(*section*, *override=False*)
Merge section to all parts.

class ly.musicxml.xml_objs.**ScorePart**(*staves=0*, *part_id=None*, *to_part=None*, *name=u''*)
Bases: *ly.musicxml.xml_objs.ScoreSection*

object to keep track of part

extract_global_to_section(*name*)
Extract only elements that is relevant for the score globally into a given section.

merge_part_to_part()
Merge the part with the one indicated.

set_first_bar(*divisions*)

class ly.musicxml.xml_objs.**ScorePartGroup**(*num*, *bracket*)
Object to keep track of part group.

merge_voice(*voice*, *override=False*)
Merge in a ScoreSection into all parts.

set_bracket(*bracket*)

class ly.musicxml.xml_objs.**ScoreSection**(*name*, *glob=False*)
object to keep track of music section

merge_lyrics(*lyrics*)
Merge in lyrics in music section.

merge_voice(*voice*, *override=False*)
Merge in other ScoreSection.

class ly.musicxml.xml_objs.**Slur**(*nr*, *slurtype*)
Stores information about slur.

class ly.musicxml.xml_objs.**Snippet**(*name*, *merge_into*)
Bases: *ly.musicxml.xml_objs.ScoreSection*

Short section intended to be merged. Holds reference to the barlist to be merged into.

class ly.musicxml.xml_objs.**TempoDir**(*unit*, *unittype*, *beats*, *dots*, *text*)
Object that stores tempo direction information

set_midi_tempo(*unit*, *beats*, *dots*)

class ly.musicxml.xml_objs.**Tuplet**(*fraction*, *ttype*, *nr*, *acttype*, *normtype*)
Stores information about tuplet.

class ly.musicxml.xml_objs.**Unpitched**(*duration*, *step=None*, *voice=1*)
Bases: *ly.musicxml.xml_objs.BarNote*

Object to keep track of unpitched notes.

ly.musicxml.xml_objs.**convert_barl**(*bl*)

ly.musicxml.xml_objs.**dur2lines**(*dur*)

ly.pitch package

Module contents

Pitch manipulation.

class ly.pitch.**LanguageName**
Bases: ly.lex._token.Token

A Token that denotes a language name.

class ly.pitch.**Pitch**(*note=0*, *alter=0*, *octave=0*, *accidental=u''*, *octavecheck=None*)
Bases: object

A pitch with note, alter and octave attributes.

Attributes may be manipulated directly.

classmethod c0()
Returns a pitch c.

classmethod c1()
Returns a pitch c'.

copy()
Returns a new instance with our attributes.

classmethod f0()
Return a pitch f.

makeAbsolute(*lastPitch*)
Makes ourselves absolute, i.e. sets our octave from lastPitch.

makeRelative(*lastPitch*)
Makes ourselves relative, i.e. changes our octave from lastPitch.

output(*language=u'nederlands'*)
Returns our string representation.

class ly.pitch.**PitchIterator**(*source*, *language=u'nederlands'*)
Bases: object

Iterate over notes or pitches in a source.

pitches()
Yields all tokens, but collects Note and Octave tokens.

When a Note is encountered, also reads octave and octave check and then a Pitch is yielded instead of the tokens.

position(*t*)
Returns the cursor position for the given token or Pitch.

read(*token*)
Reads the token and returns (note, alter) or None.

setLanguage(*lang*)
Changes the pitch name language to use.

Called internally when language or include tokens are encountered with a valid language name/file.

Sets the language attribute to the language name and the read attribute to an instance of ly.pitch.PitchReader.

tokens()
Yield all the tokens from the source, following the language.

write(*pitch*, *language=None*)
Output a changed Pitch.

The Pitch is written in the Source's document.

To use this method reliably, you must instantiate the PitchIterator with a ly.document.Source that has tokens_with_position set to True.

exception ly.pitch.**PitchNameNotAvailable**(*language*)
Bases: exceptions.Exception

Exception raised when there is no name for a pitch.

Can occur when translating pitch names, if the target language e.g. does not have quarter-tone names.

class ly.pitch.**PitchReader**(*names*, *accs*, *replacements=()*)
Bases: object

class ly.pitch.**PitchWriter**(*names*, *accs*, *replacements=()*)
Bases: object

language = u'unknown'

ly.pitch.**octaveToNum**(*octave*)
Converts string octave to an integer:

"" -> 0 ; "," -> -1 ; "'" -> 3 ; etc.

ly.pitch.**octaveToString**(*octave*)
Converts numeric octave to a string with apostrophes or commas.

0 -> "" ; 1 -> "'" ; -1 -> "," ; etc.

ly.pitch.**pitchReader**(*language*)
Returns a PitchReader for the specified language.

ly.pitch.**pitchWriter**(*language*)
Returns a PitchWriter for the specified language.

Submodules

ly.pitch.abs2rel module

Convert absolute music to relative music.

ly.pitch.abs2rel.**abs2rel**(*cursor*, *language=u'nederlands'*, *startpitch=True*, *first_pitch_absolute=False*)
Converts pitches from absolute to relative.

language: language to start reading pitch names in

startpitch: if True, write a starting pitch before the opening bracket of a relative expression.

first_pitch_absolute: this option only makes sense when startpitch is False. If first_pitch_absolute is True, the first pitch of a relative expression is written as absolute. This mimics the behaviour of LilyPond >= 2.18. (In fact, the starting pitch is then assumed to be f.)

If False, the first pitch is written as relative to c' (LilyPond < 2.18 behaviour).

Existing relative expressions are not changed.

ly.pitch.rel2abs module

Convert relative music to absolute music.

ly.pitch.rel2abs.**rel2abs**(*cursor*, *language=u'nederlands'*, *first_pitch_absolute=False*)
Converts pitches from relative to absolute.

language: language to start reading pitch names in

first_pitch_absolute: if True, the first pitch of a relative expression is regarded as absolute, when no starting pitch was given. This mimics the behaviour of LilyPond >= 2.18. (In fact, the starting pitch is then assumed to be f.)

If False, the starting pitch, when not given, is assumed to be c' (LilyPond < 2.18 behaviour).

ly.pitch.translate module

Translating the language of pitch names

`ly.pitch.translate.`**`insert_language`**(*document*, *language*, *version=None*)
Inserts a language command in the document.

The command is inserted at the top or just below the version line.

If the LilyPond version specified < (2, 13, 38), the include command is used, otherwise the newer language command.

`ly.pitch.translate.`**`translate`**(*cursor*, *language*, *default_language=u'nederlands'*)
Changes the language of the pitch names.

May raise ly.pitch.PitchNameNotAvailable if the current pitch language has no quarter tones.

Returns True if there also was a language or include language command that was changed. If not and the cursor specified only a part of the document, you could warn the user that a language or include command should be added to the document. Or you could call insert_language to add a language command to the top of the document.

ly.pitch.transpose module

Transposing music.

class `ly.pitch.transpose.`**`ModalTransposer`**(*numSteps=1*, *scaleIndex=0*)
Bases: `object`

Transpose pitches by number of steps within a given scale.

Instantiate with the number of steps (+/-) in the scale to transpose by, and a mode index. The mode index is the index of the major scale in the circle of fifths (C Major = 0).

static `getKeyIndex`(*text*)
Get the index of the key in the circle of fifths.

'Cb' returns 0, 'C' returns 7, 'B#' returns 14.

`transpose`(*pitch*)

class `ly.pitch.transpose.`**`ModeShifter`**(*key*, *scale*)
Bases: *`ly.pitch.transpose.Transposer`*

Shift pitches to optional mode/scale.

The scale should be formatted in analogy to the scale in the Transposer parent class.

The key should be an instance of ly.pitch.Pitch.

`closestPitch`(*pitch*)
Get closest pitch from scale.

If only one scale note with the same base step exist that is returned. Otherwise the closest is calculated.

`transpose`(*pitch*)
Shift to closest scale pitch if not already in scale.

class ly.pitch.transpose.**Simplifier**(*scale=None*)
Bases: *ly.pitch.transpose.Transposer*

Make complicated accidentals simpler by substituting naturals where possible.

transpose(*pitch*)

class ly.pitch.transpose.**Transposer**(*fromPitch*, *toPitch*, *scale=None*)
Bases: object

Transpose pitches.

Instantiate with a from- and to-Pitch, and optionally a scale. The scale is a list with the pitch height of the unaltered step (0 .. 6). The default scale is the normal scale: C, D, E, F, G, A, B.

scale = (0, 1, 2, Fraction(5, 2), Fraction(7, 2), Fraction(9, 2), Fraction(11, 2))

transpose(*pitch*)

ly.pitch.transpose.**transpose**(*cursor*, *transposer*, *language=u'nederlands'*, *relative_first_pitch_absolute=False*)
Transpose pitches using the specified transposer.

If relative_first_pitch_absolute is True, the first pitch in a relative expression is considered to be absolute, when a startpitch is not given. This is LilyPond >= 2.18 behaviour.

If relative_first_pitch_absolute is False, the first pitch in a relative expression is considered to be relative to c', is no startpitch is given. This is LilyPond < 2.18 behaviour.

Currently, relative_first_pitch_absolute defaults to False.

ly.xml package

Introduction

This package is concerned with representing a LilyPond structure (e.g. music) as an XML tree.

The structure of the tree closely follows LilyPond's data structures. The tree can be parsed or analysed, and could be used to convert the music to other formats like Mei of MusicXML.

This package tries to define the exact format (dtd or schema) of the tree.

There will be three ways to build the tree:

- by hand, by creating XML elements (maybe with use of some helper methods).
- from a tokenized document
- by LilyPond, using the `xml-export.ily` script that is included.

The latter case is very interesting as all the music parsing and handling is already done by LilyPond. The exported XML nearly contains all information of a score or music object. Below, some more information about the XML.

Note that this is all in heavy development.

Module contents

Routines that manipulate an XML mapping very similar to the Scheme music structure used by LilyPond itself.

The mapping can also be generated from within LilyPond and then parsed by other programs.

While designing the mapping, I decided to use xml elements for almost everything, only values that are very simple in all cases, are attributes.

Code could be written to convert such an XML music tree to other formats.

Also code is added to build such trees from scratch and from tokenized documents.

Code will be added to print LilyPond source. When all is finished, *ly.dom* is deprecated and *ly.music* will probably use these xml tree for storage.

A single LilyPond file *xml-export.ily* is also included with this module; it can be included in a LilyPond document and exports the `\displayLilyXML` music function.

The `xml-export.ily` file

Written by Wilbert Berendsen, jan-feb 2015

This LilyPond module defines a function (`xml-export`) that converts LilyPond datastructures to XML. For convenience, a `\displayLilyXML` music function is added that converts a music expression to XML.

Usage e.g.:

```
\include "/path/to/xml-export.ily"
\displayLilyXML { c d e f }
```

The XML closely follows the LilyPond music structure.

All (`make-music 'MusicName ...`) objects translate to a `<music type="MusicName">` tag. The music in the `'element` and `'elements` properties is put in the `<element>` and `<elements>` tags. (LilyPond uses `'element` when there is a single music argument, and `'elements` for a list of music arguments, but for example `\repeat` uses both: `'element` for the repeated music and `'elements` for the `\alternatives`.)

Thus `<element>`, if there, always has one `<music>` child. `<elements>`, if there, can have more than one `<music>` child.

Besides `'element` and `'elements`, the following properties of music objects are handled specially:

- `'origin` => `<origin>` element with `filename`, `line` and `char` attributes
- `'pitch` => `<pitch>` element with `octave`, `notename` and `alteration` attributes
- `'duration` => `<duration>` element with `log`, `dots`, `numer` and `denom` attributes
- `'articulations` => `<articulations>` element containing `<music>` elements
- `'tweaks` => `<tweaks>` element containing pairs (`symbol . value`)

All other properties a music object may have, are translated to a `<property>` element with a `name` attribute. The value is the child element and can be any object (string, list, pair, symbol, number etc.). (Note that the LilyPond command `\displayMusic` does not display all properties.)

Markup objects are also converted to XML, where a toplevel `<markup>` element is used. The individual markup commands are converted to an `<m>` element, with the name in the `name` attribute (e.g. `<m name="italic"><string value="Hi there!"/></m>`). Arguments to markup commands may be other commands, or other objects (markup `\score` even has a score argument, which is also supported).

Example

This LilyPond music:

```
\relative {
  c d e
}
```

maps to Scheme (using `\displayMusic`):

```
(make-music
  'RelativeOctaveMusic
  'element
  (make-music
    'SequentialMusic
    'elements
    (list (make-music
            'NoteEvent
            'pitch
            (ly:make-pitch -1 0 0)
            'duration
            (ly:make-duration 2 0 1))
          (make-music
            'NoteEvent
            'pitch
            (ly:make-pitch -1 1 0)
            'duration
            (ly:make-duration 2 0 1))
          (make-music
            'NoteEvent
            'pitch
            (ly:make-pitch -1 2 0)
            'duration
            (ly:make-duration 2 0 1)))))
```

and maps to XML (using `\displayLilyXML`):

```
<music name="RelativeOctaveMusic">
  <origin filename="/home/wilbert/dev/python-ly/ly/xml/xml-export.ily" line="244"␣
↪char="17"/>
  <element>
    <music name="SequentialMusic">
      <origin filename="/home/wilbert/dev/python-ly/ly/xml/xml-export.ily" line="244"␣
↪char="27"/>
      <elements>
        <music name="NoteEvent">
          <origin filename="/home/wilbert/dev/python-ly/ly/xml/xml-export.ily" line=
↪"245" char="4"/>
          <pitch octave="-1" notename="0" alteration="0"/>
          <duration log="2" dots="0" numer="1" denom="1"/>
        </music>
        <music name="NoteEvent">
          <origin filename="/home/wilbert/dev/python-ly/ly/xml/xml-export.ily" line=
↪"245" char="6"/>
          <pitch octave="-1" notename="1" alteration="0"/>
          <duration log="2" dots="0" numer="1" denom="1"/>
        </music>
        <music name="NoteEvent">
          <origin filename="/home/wilbert/dev/python-ly/ly/xml/xml-export.ily" line=
↪"245" char="8"/>
          <pitch octave="-1" notename="2" alteration="0"/>
          <duration log="2" dots="0" numer="1" denom="1"/>
        </music>
      </elements>
    </music>
  </element>
</music>
```

By default, the XML is written to standard output.

To automatically export a full LilyPond document to an XML representation, use the `xml-export-init.ly` script with the `--init` LilyPond option. That script automatically sets up LilyPond to output one XML document with a `<document>` root element, containing a `<book>` element for every book in the LilyPond file. (LilyPond always creates at least one book, collecting all the music or markup at the toplevel.)

The `xml-export-init.ly` script is intended to be used via the `--init` option. It automatically converts every `\book` in the score to an XML document. In this case the XML is also written to standard output by default, but you can specify another file with `-dxml-export=<filename>`.

So, to convert a LilyPond source file to an XML file containing the LilyPond music structure in XML format, use the following command:

```
lilypond --init /path/to/xml-export-init.ly -dxml-export=song.xml song.ly
```

The XML document has a `<document>` root element, containing a `<book>` element for every book in the LilyPond file.

ly.data package

Module contents

Query functions to get data from the LilyPond-generated _data.py module.

`ly.data.`**`all_grob_properties`**`()`
Returns the list of all properties.

`ly.data.`**`all_scheme_words`**`()`
Returns the list of all scheme words.

`ly.data.`**`context_properties`**`()`
Returns the list of context properties.

`ly.data.`**`engravers`**`()`
Returns the list of engravers and performers.

`ly.data.`**`grob_interface_properties`**`(`*iface*`)`
Returns the list of properties an interface supports.

`ly.data.`**`grob_interfaces`**`(`*grob*, *prop=None*`)`
Returns the list of interfaces a grob supports.

If prop is given, only returns the interfaces that define prop.

`ly.data.`**`grob_interfaces_for_property`**`(`*prop*`)`
Returns the list of interfaces that define the property.

Most times returns one, but several interface names may be returned.

`ly.data.`**`grob_properties`**`(`*grob*`)`
Returns the list of properties the named grob supports.

`ly.data.`**`grob_properties_with_interface`**`(`*grob*`)`
Returns a list of two-tuples (property, interface).

`ly.data.`**`grobs`**`()`
Returns the sorted list of all grob names.

ly.data.**music_glyphs**()
Returns the list of glyphs in the emmentaler font.

ly.data.**scheme_constants**()
Returns the list of scheme constants.

ly.data.**scheme_functions**()
Returns the list of scheme functions.

ly.data.**scheme_keywords**()
Returns the list of guile keywords.

ly.data.**scheme_variables**()
Returns the list of scheme variables.

Submodules

ly.data.makeschemedata module

generate all scheme words in the _scheme_data.py file

ly.data.makeschemedata.**main**()

ly.data.makeschemedata.**replace**(*word*)

ly.data.makeschemedata.**writeList**(*file_*, *name*, *lst*)

ly.cli package

Module contents

The commandline interface of the 'ly' command.

Submodules

ly.cli.command module

The commands that are available to the command line.

class ly.cli.command.**abs2rel**
Bases: ly.cli.command._edit_command

convert absolute music to relative

run(*opts*, *cursor*, *output*)

class ly.cli.command.**highlight**(*output=None*)
Bases: ly.cli.command._export_command

write syntax colored HTML.

run(*opts*, *cursor*, *output*)

ly.cli.command.**hl**
alias of *highlight*

class ly.cli.command.**indent**
Bases: ly.cli.command._edit_command

run the indenter

indenter(*opts*)
Get a ly.indent.Indenter initialized with our options.

run(*opts*, *cursor*, *output*)

class ly.cli.command.**language**
Bases: ly.cli.command._info_command

print language to stdout

get_info(*info*)

class ly.cli.command.**mode**
Bases: ly.cli.command._info_command

print mode to stdout

get_info(*info*)

class ly.cli.command.**musicxml**(*output=None*)
Bases: ly.cli.command._export_command

run(*opts*, *cursor*, *output*)

class ly.cli.command.**reformat**
Bases: *ly.cli.command.indent*

reformat the document

run(*opts*, *cursor*, *output*)

class ly.cli.command.**rel2abs**
Bases: ly.cli.command._edit_command

convert relative music to absolute

run(*opts*, *cursor*, *output*)

class ly.cli.command.**set_variable**(*arg*)
Bases: ly.cli.command._command

set a variable to a value

run(*opts*, *cursor*, *output*)

class ly.cli.command.**simplify_accidentals**
Bases: ly.cli.command._edit_command

replace notes with accidentals as much as possible with their natural neighbors

run(*opts*, *cursor*, *output*)

class ly.cli.command.**translate**(*language*)
Bases: ly.cli.command._edit_command

translate pitch names

run(*opts*, *cursor*, *output*)

class ly.cli.command.**transpose**(*arg*)
Bases: ly.cli.command._edit_command

transpose music

run(*opts*, *cursor*, *output*)

class ly.cli.command.**version**
Bases: ly.cli.command._info_command

print version to stdout

get_info(*info*)

class ly.cli.command.**write**(*output=None*)
Bases: ly.cli.command._command

write the source file.

run(*opts*, *cursor*, *output*)

ly.cli.main module

The entry point for the 'ly' command.

class ly.cli.main.**Options**
Bases: object

Store all the startup options and their defaults.

set_variable(*name*, *value*)

class ly.cli.main.**Output**
Bases: object

Object living for a whole file/command operation, handling the output.

When opening a file it has already opened earlier, the file is appended to (like awk).

file(**args*, ***kwds*)
Return a context manager for writing to.

If you set encoding to "binary" or False, the file is opened in binary mode and you should encode the data you write yourself.

get_filename(*opts*, *filename*)
Queries the output attribute from the Options and returns it.

If replace_pattern is True (by default) and the attribute contains a '*', it is replaced with the full path of the specified filename, but without extension. It the attribute contains a '?', it is replaced with the filename without path and extension.

If '-' is returned, it denotes standard output.

ly.cli.main.**die**(*message*)
Exit with message to STDERR.

ly.cli.main.**load**(*filename*, *encoding*, *mode*)
Load a file, returning a ly.document.Document

ly.cli.main.**main**()

ly.cli.main.**parse_command**(*arg*)
Parse the command string, returning a list of command.command instances.

Exits when a command is invalid.

ly.cli.main.**parse_command_line**()
Return a three-tuple(options, commands, files).

options is an Options instance with all the command-line options commands is a list of command.command instances files is the list of filename arguments

Also performs error handling and may exit on certain circumstances.

ly.cli.main.**usage**()
Print usage info.

ly.cli.main.**usage_short**()
Print short usage info.

ly.cli.main.**version**()
Print version info.

ly.server package

Module contents

A package implementing an HTTP server to process LilyPond input code.

Submodules

ly.server.command module

The commands that are available to the command line.

class ly.server.command.**abs2rel**
Bases: ly.server.command._edit_command

convert absolute music to relative

edit(*opts*, *cursor*)

class ly.server.command.**highlight**
Bases: ly.server.command._export_command

convert source to syntax colored HTML.

export(*opts*, *cursor*, *exports*)

class ly.server.command.**indent**
Bases: ly.server.command._edit_command

run the indenter

edit(*opts*, *cursor*)

indenter(*opts*)
Get a ly.indent.Indenter initialized with our options.

class ly.server.command.**language**
Bases: ly.server.command._info_command

retrieve language from document

get_info(*info*)

class ly.server.command.**mode**
Bases: ly.server.command._info_command

retrieve mode from document

get_info(*info*)

class ly.server.command.**musicxml**
Bases: ly.server.command._export_command

convert source to MusicXML

export(*opts*, *cursor*, *exports*)

class ly.server.command.**reformat**
Bases: *ly.server.command.indent*

reformat the document

edit(*opts*, *cursor*)

class ly.server.command.**rel2abs**
Bases: ly.server.command._edit_command

convert relative music to absolute

edit(*opts*, *cursor*)

class ly.server.command.**set_variable**(*arg*)
Bases: ly.server.command._command

set a configuration variable to a value

run(*opts*, *data*)

class ly.server.command.**translate**(*language*)
Bases: ly.server.command._edit_command

translate pitch names

edit(*opts*, *cursor*)

class ly.server.command.**transpose**(*arg*)
Bases: ly.server.command._edit_command

transpose music

edit(*opts*, *cursor*)

class ly.server.command.**version**
Bases: ly.server.command._info_command

retrieve version from document

get_info(*info*)

ly.server.handler module

HTTP request handler

class ly.server.handler.**RequestHandler**(*request*, *client_address*, *server*)
Bases: BaseHTTPServer.BaseHTTPRequestHandler

create_command(*cmd*)
Parse one command from the JSON data, plus optionally some commands to set variables. Returns an array with command._command instances. Raises exceptions upon faulty data.

do_POST()
A POST request is expected to contain the task to be executed as a JSON object in the request body. The POST handler (currently) ignores the URL.

process_json_request(*request*)
Configure the action(s) to be taken, based on the JSON object. Raise errors when the JSON object can't be properly understood. Run the commands and return a string (from cursor.text()).

process_options(*opts*)
Instantiate a copy of the default options and update with the given opts

read_json_request()
Returns the message body parsed to a dictionary from JSON data. Raises - RuntimeWarning when no JSON data is present - ValueError when JSON parsing fails

ly.server.main module

The entry point for the 'ly-server' command.

ly.server.main.**die**(*message*)
Exit with message to STDERR. In ly-server this should only be called upon startup, not while processing requests. Then the error should be sent back to the client.

ly.server.main.**main**()

ly.server.main.**parse_command_line**()
Returns a two-tuple(server_opts, cmd_opts)

server_opts is a ServerOptions instance configuring the server behaviour cmd_opts is an Options instance with default options for future command executions triggered by HTTP requests.

Also performs error handling and may exit on certain circumstances.

ly.server.main.**usage**()
Print usage info.

ly.server.main.**usage_short**()
Print short usage info.

ly.server.main.**version**()
Print version info.

ly.server.options module

Options configuring the commands' behaviour

class ly.server.options.**Options**
Bases: object

Store all the startup options and their defaults.

set_variable(*name*, *value*)

class ly.server.options.**ServerOptions**
Bases: object

Options configuring the server itself, not the individual commands

Submodules

ly.slexer module

slexer – Stateful Lexer

parses text, searching for tokens represented by a regular expression.

Only depends on the standard Python re module.

You need to create at least one subclass of Parser, and a subclass of Token for every type of text to search for. Then you list the token class names in the 'items' tuple of the Parser subclass definition.

Different contexts can be parsed by creating multiple Parser subclasses. A Parser searches for tokens using the list of Token classes. (Token is simply a subclass of str in Python 3 and of unicode in Python 2). Every Token subclass has the regular expression part to search for in its 'rx' class attribute.

You start parsing text by instantiating a State (you don't need to subclass that) with the Parser subclass you want to parse the text with. Then you iterate over the generated tokens using the tokens(text) method of the State instance. You can use the tokens just as strings (e.g. if token == 'text'...) but you can also test for the type of the token (e.g. if isinstance(token, Number)...). The tokens also carry a 'pos' and an 'end' attribute, specifying their position in the parsed text string.

A token may cause a different Parser to be entered, of the current Parser to be left, etc. This is done by implementing the update_state() method of the Token subclass. This method is called automatically when the Token is instantiated.

The State maintains the parsing state (the list of active Parser instances). A State can be frozen to be thawed later to resume parsing text starting in a particular context. A Fridge can be used to store and recover a state under a simple integer number.

How to use slexer:

```
from slexer import Token, Parser, State

# create token classes:
class Word(Token):
    rx = r'\w+'

class Number(Token):
    rx = r'\d+'

class String(Token):
    pass

class StringStart(String):
    rx = '"'
    def update_state(self, state):
        state.enter(PString())

class StringEnd(String):
    rx = '"'
    def update_state(self, state):
        state.leave()

# create parsers:
class PTest(Parser):
    '''Looks for numbers, words and the double quote.'''
    items = (
```

```
        Number,
        Word,
        StringStart,
    )

class PString(Parser):
    '''Returns String by default, quits at double quote.'''
    default = String
    items = (
        StringEnd,
    )

s = State(PTest)
for t in s.tokens(
    'een tekst met 7 woorden, '
    'een "tekst met 2 aanhalingstekens" '
    'en 2 of 3 nummers'):
    print(t.__class__, t)
```

Running the above code, the result is:

```
<class '__main__.Word'> een
<class '__main__.Word'> tekst
<class '__main__.Word'> met
<class '__main__.Number'> 7
<class '__main__.Word'> woorden
<class '__main__.Word'> een
<class '__main__.StringStart'> "
<class '__main__.String'> tekst met 2 aanhalingstekens
<class '__main__.StringEnd'> "
<class '__main__.Word'> en
<class '__main__.Number'> 2
<class '__main__.Word'> of
<class '__main__.Number'> 3
<class '__main__.Word'> nummers
```

class ly.slexer.**Token**

Bases: unicode

Represents a parsed piece of text.

The subclass determines the type.

You should put the regular expression string in the rx class attribute. In the rx string, you may not use named groups starting with "g_".

To add token types to a Parser class, list the token class in the items attribute of the Parser class.

end

pos

rx = None

classmethod test_match(*match*)

Should return True if the match should indeed instantiate this class.

This class method is only called if multiple Token classes in the Parser's items list have the same rx attribute. This method is then called for every matching Token subclass until one returns True. That token is then instantiated. (The method is not called for the last class in the list that have the same rx attribute, and also not if there is only one class with that rx attribute.)

The default implementation always returns True.

update_state(*state*)
Lets the token update the state, e.g. enter a different parser.

This method is called by the State upon instantiation of the tokens.

Don't use it later on to have a State follow already instantiated Tokens because the FallthroughParser type can also change the state without generating a Token. Use State.follow() to have a State follow instantiated Tokens.

The default implementation lets the Parser decide on state change.

class ly.slexer.**Parser**
Bases: object

Abstract base class for Parsers.

When creating Parser subclasses, you should set the 'items' attribute to a tuple of Token subclasses. On class construction, a large regular expression pattern is built by combining the expressions from the 'rx' attributes of the Token subclasses.

Additionally, you may implement the update_state() method which is called by the default implementation of update_state() in Token.

default = None

fallthrough(*state*)
Called when no match is returned by parse().

If this function returns True, the tokenizer stops parsing, assuming all text has been consumed. If this function returns False or None, it should alter the state (switch parsers) and parsing continues using the new Parser.

The default implementation simply returns True.

freeze()
Return our instance values as a hashable tuple.

items = ()

parse(*text*, *pos*)
Parse text from position pos and returns a Match Object or None.

re_flags = 0

classmethod thaw(*attrs*)

token(*match*)
Return a Token instance of the correct class.

The match object is returned by the parse() method.

update_state(*state*, *token*)
Called by the default implementation of Token.update_state().

Does nothing by default.

class ly.slexer.**FallthroughParser**
Bases: *ly.slexer.Parser*

Base class for parsers that 'match' instead of 'search' for a pattern.

You can also implement the fallthrough() method to do something with the state if there is no match. The default is to leave the current parser. See Parser().

fallthrough(*state*)
Called when no match is returned by parse().

This implementation leaves the current parser and returns None (causing the State to continue parsing).

parse(*text*, *pos*)
Match text at position pos and returns a Match Object or None.

class ly.slexer.**State**(*initialParserClass*)
Bases: object

Maintains state while parsing text.

You instantiate a State object with an initial parser class. Then you use tokens(text) to start parsing for tokens.

The state is basically a list of Parser instances; the last one is the active one. The enter() and leave() methods respectively enter a new parser or leave the current parser.

You can't leave() the initial parser instance.

depth()
Return the number of parsers currently active (1 or more).

You can use this e.g. to keep parsing until some context ends:

```
tokens = state.tokens(text) # iterator
depth = state.depth()
for token in tokens:
    if state.depth() < depth:
        break
    # do something
```

enter(*parser*)
Enter a new parser.

Note: 'parser' is an instantiated Parser subclass. Most times this method will be called from with the update_state() method of a Token subclass (or from a Parser subclass, which is also possible: the default implementation of Token.update_state() calls Parser.update_state(), which does nothing by default).

E.g. in the Token subclass:

```
def update_state(self, state):
    state.enter(SomeDifferentParser())
```

follow(*token*)
Act as if the token has been instantiated with the current state.

You need this when you already have the parsed tokens, (e.g. cached or saved somehow) but want to know which parser created them.

This method changes state according to the token. Basically it calls the update_state() method of the token instance, but it does some more work behind the scenes to ensure that the FallthroughParser type (see below) also is handled correctly.

freeze()
Return the current state as a tuple (hashable object).

leave()
Leave the current parser and pop back to the previous.

The first parser (specified on instantiation) will never be left.

parser()
Return the currently active Parser instance.

parsers()
Return all active parsers, the most current one first.

replace(*parser*)
Replace the current parser with a new one.

Somewhat equivalent to:

```
state.leave()
state.enter(SomeDifferentParser)
```

But using this method you can also replace the first parser.

classmethod thaw(*frozen*)
Reproduce a State object from the frozen state argument.

tokens(*text*, *pos=0*)
Parse a text string using our state info.

Yields Token instances. All tokens are a subclass of str (or unicode in Python 2.x) and have a pos and an end attribute, describing their position in the original string. If the current parser defines a 'default' class attribute, it is the Token subclass to use for pieces of text that would otherwise be skipped.

class ly.slexer.**Fridge**(*stateClass=<class 'ly.slexer.State'>*)
Bases: object

Stores frozen States under an integer number.

count()
Returns the number of stored frozen states.

freeze(*state*)
Stores a state and return an identifying integer.

thaw(*num*)
Returns the state stored under the specified number.

ly.document module

DocumentBase and Document

Represents a LilyPond source document (the text contents).

The Document implementation keeps the document in a (unicode) text string, but you can inherit from the DocumentBase class to support other representations of the text content.

Modifying is preferably done inside a context (the with statement), e.g.:

```
d = Document('some string')
with d:
    d[5:5] = 'different '
d.plaintext()  --> 'some different string'
```

Changes are applied when the context is exited, also the modified part of the document is re-tokenized. Changes may not overlap.

You may modify the document outside a context, in which case the document is re-tokenized immediately. This is much slower however when performing multiple changes after each other.

The tokens(block) method returns a tuple of tokens for the specified block. Depending on the implementation, a block describes a line in the LilyPond source document. It is not expected to have any methods, except that the '==' operator is supported between two blocks, and returns True if both refer to the same line of text in the source document.

Cursor

Defines a range or position in a Document.

Runner

A Runner allows iterating back and forth over the tokens of a document.

Source

Iterate over tokens in a (part of a) Document, with or without state.

class `ly.document.`**`Cursor`**(*doc*, *start=0*, *end=None*)

Bases: `object`

Defines a certain range (selection) in a Document.

You may change the start and end attributes yourself. Both must be an integer, end may also be None, denoting the end of the document.

As long as you keep a reference to the Cursor, its positions are updated when the document changes. When text is inserted at the start position, it remains the same. But when text is inserted at the end of a cursor, the end position moves along with the new text. E.g.:

```
d = Document('hi there, folks!')
c = Cursor(d, 8, 8)
with d:
    d[8:8] = 'new text'
c.start, c.end --> (8, 16)
```

Many tools in the ly module use this object to describe (part of) a document.

`blocks`()

Iterate over the selected blocks.

If there are multiple blocks and the cursor ends on the first position of the last selected block, that block is not included.

`document`

`end_block`()

Return the block the end attribute points at.

`has_selection`()

Return True when there is some text selected.

`lstrip`(*chars=None*)

Move start to the right, like Python's lstrip() string method.

`rstrip`(*chars=None*)

Move end to the left, like Python's lstrip() string method.

`select_all`()

Select all text.

select_end_of_block()
Move end to the end of the block.

select_start_of_block()
Move start to the start of the block.

start_block()
Return the block the start attribute points at.

strip(*chars=None*)
Strip chars from the selection, like Python's strip() method.

text()
Convenience method to return the selected text.

text_after()
Return text after the cursor in it's end block.

text_before()
Return text before the cursor in it's start block.

class ly.document.**Document**(*text=u''*, *mode=None*)
Bases: *ly.document.DocumentBase*

A plain text LilyPond source document that auto-updates the tokens.

The modified attribute is set to True as soon as the document is changed, but the setplaintext() method sets it to False.

apply_changes()

block(*position*)
Return the text block at the specified character position.

copy()
Return a full copy of the document.

index(*block*)
Return the linenumber of the block (starting with 0).

initial_state()
Return the state at the beginning of the document.

isvalid(*block*)
Return True if the block is a valid block.

classmethod load(*filename*, *encoding=u'utf-8'*, *mode=None*)
Load the document from a file, using the specified encoding and mode.

mode()
Return the mode (lilypond, html, etc). None means automatic mode.

modified = False

position(*block*)
Return the position of the specified block.

setmode(*mode*)
Sets the mode to one of the ly.lex modes.

Use None to auto-determine the mode.

setplaintext(*text*)
Set the text of the document, sets modified to False.

state_end(*block*)
Return the state at the end of the specified block.

text(*block*)
Return the text of the specified block.

tokens(*block*)
Return the tuple of tokens of the specified block.

class ly.document.**DocumentBase**
Bases: object

Abstract base class for Document instances.

You should inherit the following methods:

setplaintext __len__ __getitem__ block index position text tokens isvalid initial_state state_end apply_changes

You may inherit (e.g. to get speed improvements):

plaintext next_block previous_block blocks_forward blocks_backward state

You may use the following attributes:

filename (None) # can represent the filename of the document on disk encoding (None) # can represent the encoding of the document when reading/writing to disk

apply_changes()
Apply the changes and update the tokens.

block(*position*)
Return the text block at the specified character position.

The text block itself has no methods, but it can be used as an argument to other methods of this class.

(Blocks do have to support the '==' operator.)

blocks_backward(*block*)
Iter backwards starting with the specified block.

blocks_forward(*block*)
Iter forward starting with the specified block.

check_changes()
Debugging method that checks for overlapping edits.

encoding = **None**

filename = **None**

index(*block*)
Return the linenumber of the block (starting with 0).

initial_state()
Return the state at the beginning of the document.

isblank(*block*)
Return True if the block is empty or blank.

isvalid(*block*)
Return True if the block is a valid block.

next_block(*block*)
Return the next block, which may be invalid.

plaintext()
The document contents as a plain text string.

position(*block*)
Return the position of the specified block.

previous_block(*block*)
Return the previous block, which may be invalid.

setplaintext(*text*)
Sets the document contents to the text string.

size()
Return the number of characters in the document.

state(*block*)
Return the state at the start of the specified block.

state_end(*block*)
Return the state at the end of the specified block.

text(*block*)
Return the text of the specified block.

tokens(*block*)
Return the tuple of tokens of the specified block.

The pos and end attributes of every token point to the position of the token in the block.

tokens_with_position(*block*)
Return a tuple of tokens of the specified block.

The pos and end attributes of every token point to the position in the Document, instead of to the position in the current block.

This makes it easier to iterate over tokens and change the document.

update_cursors()
Updates the position of the registered Cursor instances.

class ly.document.**Runner**(*doc*, *tokens_with_position=False*)
Bases: object

Iterates back and forth over tokens.

A Runner can stop anywhere and remembers its current token.

classmethod at(*cursor*, *after_token=False*, *tokens_with_position=False*)
Create and init from a Cursor.

The Runner is positioned so that yielding forward starts with the first complete token after the cursor's start position.

Set after_token to True if you want to position the cursor after the token, so that it gets yielded when you go backward.

If tokens_with_position is True, uses the tokens_with_position() method to get the tokens, else (by default), the tokens() method is used.

backward()
Yields tokens in backward direction across blocks.

backward_line()
Yields tokens in backward direction in the current block.

copy()
Return a new Runner at the current position.

document
Return our Document.

forward()
Yields tokens in forward direction across blocks.

forward_line()
Yields tokens in forward direction in the current block.

move_to_block(*block*, *at_end=False*)
Positions the Runner at the start of the given text block.

If at_end == True, the iterator is positioned past the end of the block.

next_block(*at_end=False*)
Go to the next block, positioning the cursor at the start by default.

Returns False if there was no next block, else True.

position()
Returns the position of the current token.

previous_block(*at_end=True*)
Go to the previous block, positioning the cursor at the end by default.

Returns False if there was no previous block, else True.

set_position(*position*, *after_token=False*)
Positions the Runner at the specified position.

Set after_token to True if you want to position the cursor after the token, so that it gets yielded when you go backward.

token()
Re-returns the last yielded token.

class ly.document.**Source**(*cursor*, *state=None*, *partial=1*, *tokens_with_position=False*)
Bases: object

Helper iterator.

Iterates over the (block, tokens) tuples from a Document (or a part thereof). Stores the current block in the block attribute and the tokens (which also is a generator) in the tokens attribute.

Iterating over the source object itself just yields the tokens, while the block attribute contains the current block.

You can also iterate over the tokens attribute, which will yield the remaining tokens of the current block and then stop.

If you specify a state, the tokens will update the state. If you specify state = True, the state will be taken from the document.

consume(*iterable*, *position*)
Consumes iterable (supposed to be reading from us) until position.

Returns the last token if that overlaps position.

document
Return our Document.

next()

position(*token*)
Returns the position of the token in the current block.

If the iterator was instantiated with tokens_with_position == True, this position is the same as the token.pos attribute, and the current block does not matter. (In that case you'll probably not use this method.)

pushback(*pushback=True*)
Yields the last yielded token again on the next request.

This can be called multiple times, but only the last token will be yielded again. You can also undo a call to pushback() using pushback(False).

token()
Re-returns the last yielded token.

until_parser_end()
Yield the tokens until the current parser is quit.

You can only use this method if you have a State enabled.

ly.docinfo module

Harvest information from a ly.document.DocumentBase instance.

class ly.docinfo.**DocInfo**(*doc*)
Bases: object

Harvest information from a ly.document.DocumentBase instance.

All tokens are saved in the tokens attribute as a tuple. Newline tokens are added between all lines. All corresponding classes are in the classes attribute as a tuple. This makes quick search and access possible.

The tokens are requested from the document using the tokens_with_position() method, so you can always locate them back in the original document using their pos attribute.

DocInfo does not update when the document changes, you should just instantiate a new one.

complete()
Return whether the document is probably complete and could be compilable.

count_tokens(*cls*)
Return the number of tokens that are (a subclass) of the specified class.

If you only want the number of instances of the exact class (not a subclass of) you can use info.classes.count(cls), where info is a DocInfo instance.

counted_tokens()
Return a dictionary mapping classes to the number of instances of that class.

definitions()
The list of LilyPond identifiers the document defines.

document

find(*token=None*, *cls=None*, *pos=0*, *endpos=-1*)
Return the index of the first specified token and/or class after pos.

If token is None, the cls should be specified. If cls is given, the token should be an instance of the specified class. If endpos is given, never searches beyond endpos. Returns -1 if the token is not found.

find_all(*token=None*, *cls=None*, *pos=0*, *endpos=-1*)
Yield all indices of the first specified token and/or class after pos.

If token is None, the cls should be specified. If cls is given, the token should be an instance of the specified class. If endpos is given, never searches beyond endpos. Returns -1 if the token is not found.

global_staff_size()
The global-staff-size, if set, else None.

has_output()
Return True when the document probably generates output.

I.e. has notes, rests, markup or other output-generating commands.

include_args()
The list of include command arguments.

language()
The pitch language, None if not set in the document.

markup_definitions()
The list of markup command definitions in the document.

mode()
Return the mode, e.g. "lilypond".

output_args()
The list of arguments of constructs defining the name of output documents.

This looks at the bookOutputName, bookOutputSuffix and define output-suffix commands.

Every argument is a two tuple(type, argument) where type is either "suffix" or "name".

range(*start=0*, *end=None*)
Return a new instance of the DocInfo class for the selected range.

Only the tokens completely contained within the range start..end are added to the new instance. This can be used to perform fast searches on a subset of a document.

scheme_load_args()
The list of scheme (load) command arguments.

token_hash()
Return an integer hash for all non-whitespace and non-comment tokens.

This hash does not change when only comments or whitespace are changed.

version()
Return the version_string() as a tuple of ints, e.g. (2, 16, 2).

version_string()
Return the version as a string, e.g. "2.19.8".

Looks for the version LilyPond command. The string is returned without quotes. Returns None if there was no version command found.

ly.barcheck module

Add, check or remove bar checks in selected music.

class ly.barcheck.**event**
Bases: object

A limited event type at a certain time.

append(*node*)

ly.barcheck.**insert**(*cursor*, *music=None*)
Insert bar checks within the selected range.

ly.barcheck.**remove**(*cursor*)
Remove bar checks from the selected music.

ly.colorize module

Classes and functions to colorize (syntax-highlight) parsed source.

Highlighting is based on CSS properties and their values, although the Mapping object can map a token's class to any object or value.

The Mapping object normally maps a token's class basically to a CSS class and possibly a base CSS class. This way you can define base styles (e.g. string, comment, etc) and have specific classes (e.g. LilyPond string, Scheme comment) inherit from that base style. This CSS class is described by the css_class named tuple, with its three fields: mode, name, base. E.g. ('lilypond', 'articulation', 'keyword'). The base field may be None.

The css classes are mapped to dictionaries of css properties, like {'font-weight': 'bold', 'color': '#4800ff'}, etc.

A scheme (a collection of styles) is simply a dictionary mapping the mode to a dictionary of CSS dictionaries. The base styles are in the [None] item of the scheme dictionary.

class ly.colorize.**HtmlWriter**
Bases: object

A do-it-all object to create syntax highlighted HTML.

You can set the instance attributes to configure the behaviour in all details. Then call the html(cursor) method to get the HTML.

bgcolor = None

css_mapper = None

css_scheme = {u'mup': {}, None: {u'function': {u'color': u'#0000c0', u'font-weight': u'bold'}, u'comment': {u'color':

document_id = u'document'

encoding = u'UTF-8'

fgcolor = None

full_html = True

html(*cursor*)
Return the output HTML.

inline_style = False

linenumbers_bgcolor = u'#eeeeee'

linenumbers_fgcolor = None

linenumbers_id = u'linenumbers'

number_lines = False

set_wrapper_attribute(*attr*)
Choose attribute name for wrapper tag

set_wrapper_tag(*tag*)
Define the tag to be used for wrapping the content

stylesheet_ref = None

title = u''

wrapper_attribute = u'id'

wrapper_tag = u'pre'

class ly.colorize.**Mapper**
Bases: dict

Maps token classes to arbitrary values, which can be highlighting styles.

Mapper behaves like a dict, you set items with a token class as key to an arbitrary value.

But getting items can be done using a token. The token class's method resolution order is walked up and the value for the first available class found in the keys is returned. The class is also cached to speed up requests for other tokens.

ly.colorize.**add_line_numbers**(*cursor*, *html*, *linenum_attrs=None*, *document_attrs=None*)
Combines the html (returned by html()) with the line numbers in a HTML table.

The linenum_attrs are put in the <td> tag for the line numbers. The default value is: {"style": "background: #eeeeee;"}. The document_attrs are put in the <td> tag for the document. The default is empty.

By default, the id for the linenumbers <td> is set to "linenumbers", and the id for the document <td> is set to "document".

ly.colorize.**css_attr**(*d*)
Return a dictionary with a 'style' key.

The value is the style items in d formatted with css_item() joined with spaces. If d is empty, an empty dictionary is returned.

class ly.colorize.**css_class**(*mode*, *name*, *base*)
Bases: tuple

base
Alias for field number 2

mode
Alias for field number 0

name
Alias for field number 1

ly.colorize.**css_dict**(*css_style*, *scheme={u'mup': {}, None: {u'function': {u'color': u'#0000c0', u'font-weight': u'bold'}, u'comment': {u'color': u'#808080', u'font-style': u'italic'}, u'string': {u'color': u'#c00000'}, u'keyword': {u'font-weight': u'bold'}, u'escape': {u'color': u'#008080'}, u'variable': {u'color': u'#0000ff'}, u'error': {u'color': u'#ff0000', u'text-decoration-color': u'#ff0000', u'text-decoration': u'underline'}, u'value': {u'color': u'#808000'}}, u'html': {}, u'scheme': {}, u'texinfo': {}, u'lilypond': {u'lyrictext': {u'color': u'#006000'}, u'articulation': {u'color': u'#ff8000', u'font-weight': u'bold'}, u'grob': {u'color': u'#c000c0'}, u'dynamic': {u'color': u'#ff8000', u'font-weight': u'bold'}, u'fingering': {u'color': u'#ff8000'}, u'duration': {u'color': u'#008080'}, u'lyricmode': {u'color': u'#006000'}, u'stringnumber': {u'color': u'#ff8000'}, u'markup': {u'color': u'#008000', u'font-weight': u'normal'}, u'slur': {u'font-weight': u'bold'}, u'context': {u'font-weight': u'bold'}}}*)

Return the css properties dict for the style, taken from the scheme.

This can be used for inline style attributes.

ly.colorize.**css_group**(*selector*, *d*)
Return a "selector { items...}" part of a CSS stylesheet.

ly.colorize.**css_item**(*i*)
Return "name: value;" where i = (name, value).

ly.colorize.**css_mapper**(*mapping=None*)
Return a Mapper dict, mapping token classes to two CSS classes.

By default the mapping returned by default_mapping() is used.

class ly.colorize.**css_style_attribute_formatter**(*scheme={u'mup': {}, None: {u'function': {u'color': u'#0000c0', u'font-weight': u'bold'}, u'comment': {u'color': u'#808080', u'font-style': u'italic'}, u'string': {u'color': u'#c00000'}, u'keyword': {u'font-weight': u'bold'}, u'escape': {u'color': u'#008080'}, u'variable': {u'color': u'#0000ff'}, u'error': {u'color': u'#ff0000', u'text-decoration-color': u'#ff0000', u'text-decoration': u'underline'}, u'value': {u'color': u'#808000'}}, u'html': {}, u'scheme': {}, u'texinfo': {}, u'lilypond': {u'lyrictext': {u'color': u'#006000'}, u'articulation': {u'color': u'#ff8000', u'font-weight': u'bold'}, u'grob': {u'color': u'#c000c0'}, u'dynamic': {u'color': u'#ff8000', u'font-weight': u'bold'}, u'fingering': {u'color': u'#ff8000'}, u'duration': {u'color': u'#008080'}, u'lyricmode': {u'color': u'#006000'}, u'stringnumber': {u'color': u'#ff8000'}, u'markup': {u'color': u'#008000', u'font-weight': u'normal'}, u'slur': {u'font-weight': u'bold'}, u'context': {u'font-weight': u'bold'}}}*)

Bases: object

Return the inline style attribute for a specified style.

ly.colorize.**default_mapping**()
Return a good default mapping from token class(es) to style and default style, per group.

ly.colorize.**format_css_span_class**(*css_style*)
Return a string like 'class="mode-style base"' for the specified style.

ly.colorize.**format_html_document**(*body*, *title=u''*, *stylesheet=None*, *stylesheet_ref=None*, *encoding=u'UTF-8'*)
Return a complete HTML document.

The body is put inside body tags unchanged. The title is html-escaped. If stylesheet_ref is given, it is put as a <link> reference in the HTML; if stylesheet is given, it is put verbatim in a <style> section in the HTML. The encoding is set in the meta http-equiv field, but the returned HTML is in normal Python unicode (python2) or str (python3) format, you should encode it yourself in the same encoding (by default utf-8) when writing it to a file.

ly.colorize.**format_stylesheet**(*scheme={u'mup': {}, None: {u'function': {u'color': u'#0000c0', u'font-weight': u'bold'}, u'comment': {u'color': u'#808080', u'font-style': u'italic'}, u'string': {u'color': u'#c00000'}, u'keyword': {u'font-weight': u'bold'}, u'escape': {u'color': u'#008080'}, u'variable': {u'color': u'#0000ff'}, u'error': {u'color': u'#ff0000', u'text-decoration-color': u'#ff0000', u'text-decoration': u'underline'}, u'value': {u'color': u'#808000'}}, u'html': {}, u'scheme': {}, u'texinfo': {}, u'lilypond': {u'lyrictext': {u'color': u'#006000'}, u'articulation': {u'color': u'#ff8000', u'font-weight': u'bold'}, u'grob': {u'color': u'#c000c0'}, u'dynamic': {u'color': u'#ff8000', u'font-weight': u'bold'}, u'fingering': {u'color': u'#ff8000'}, u'duration': {u'color': u'#008080'}, u'lyricmode': {u'color': u'#006000'}, u'stringnumber': {u'color': u'#ff8000'}, u'markup': {u'color': u'#008000', u'font-weight': u'normal'}, u'slur': {u'font-weight': u'bold'}, u'context': {u'font-weight': u'bold'}}}*)

Return a formatted stylesheet for the stylesheet scheme dictionary.

ly.colorize.**get_tokens**(*cursor*)

Return the list of tokens for the cursor.

Tokens that are partially inside the cursor's selection are re-created so that they fall exactly within the selection.

This can be used to convert a highlighted part of a document to e.g. HTML.

ly.colorize.**html**(*cursor*, *mapper*, *span=<function format_css_span_class>*)

Return a HTML string with the tokens wrapped in <span class=> elements.

The span argument is a function returning an attribute for the tag for the specified style. By default the format_css_span_class() function is used, that returns a 'class="group style base"' string. You'll want to wrap the HTML inside <pre> tokens and add a CSS stylesheet.

ly.colorize.**html_escape**(*text*)

Escape &, < and >.

ly.colorize.**html_escape_attr**(*text*)

Escape &, ", < and >.

ly.colorize.**html_format_attrs**(*d*)

Format the attributes dict as a string.

The attributes are escaped correctly. A space is prepended for every assignment.

ly.colorize.**map_tokens**(*cursor*, *mapper*)

Yield a two-tuple(token, style) for every token.

The style is what mapper[token] returns. Style may be None, which also happens with unparsed (not-tokenized) text.

ly.colorize.**melt_mapped_tokens**(*mapped_tokens*)

Melt adjacent tokens with the same mapping together.

class ly.colorize.**style**(*name*, *base*, *classes*)

Bases: tuple

base

Alias for field number 1

classes

Alias for field number 2

name
Alias for field number 0

ly.cursortools module

Routines manipulating ly.document.Cursor instances.

ly.cursortools.**find_indent**(*iterable*)
Yield (token, is_indent, nest) for every occurring indent/dedent token.

The tokens are yielded from the specified iterable.

ly.cursortools.**select_block**(*cursor*)
Try to select a meaningful block.

Searches backwards for an indenting token, then selects up to the corresponding dedenting token. If needed searches an extra level back to always extend the selection. Returns True if the cursor's selection has changed.

ly.dom module

This module is deprecated. When *ly.music* is able to generate LilyPond code from scratch, this module will be removed. LilyPond DOM

(c) 2008-2011 Wilbert Berendsen License: GPL.

A simple Document Object Model for LilyPond documents.

The purpose is to easily build a LilyPond document with good syntax, not to fully understand all features LilyPond supports. (This DOM does not enforce a legal LilyPond file.)

All elements of a LilyPond document inherit Node.

Note: elements keep a weak reference to their parent.

class ly.dom.**AddDuration**
Bases: object

Mixin to add a duration (as child).

ly(*printer*)

class ly.dom.**AddLyrics**(*parent=None*)
Bases: *ly.dom.InputLyrics*

after = 1

before = 1

may_remove_brackets = False

name = u'addlyrics'

class ly.dom.**Assignment**(*name=None*, *parent=None*, *valueObj=None*)
Bases: *ly.dom.Container*

A varname = value construct with it's value as its first child The name can be a string or a Reference object: so that everywhere where this varname is referenced, the name is the same.

after = 1

before = 1

ly(*printer*)

setValue(*obj*)

value()

class ly.dom.**BlankLine**(*parent=None*)
Bases: *ly.dom.Newline*

A blank line.

before = 1

class ly.dom.**Block**(*parent=None*)
Bases: *ly.dom.Container*

A vertical container type that puts everything on a new line.

after = 1

before = 1

defaultSpace = u'\n'

class ly.dom.**BlockComment**(*text=u''*, *parent=None*)
Bases: *ly.dom.Comment*

A block comment between %{ and %}

after

before

ly(*printer*)

class ly.dom.**Book**(*parent=None*)
Bases: *ly.dom.Section*

name = u'book'

class ly.dom.**BookPart**(*parent=None*)
Bases: *ly.dom.Section*

name = u'bookpart'

class ly.dom.**ChoirStaff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Chord**(*parent=None*)
Bases: *ly.dom.Container*

A chord containing one of more Pitches and optionally one Duration. This is a bit of a hack, awaiting real music object support.

ly(*printer*)

class ly.dom.**ChordMode**(*parent=None*)
Bases: *ly.dom.InputMode*

name = u'chordmode'

class ly.dom.**ChordNames**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Clef**(*clef*, *parent=None*)
Bases: *ly.dom.Leaf*

A clef.

ly(*printer*)

class ly.dom.**Command**(*name*, *parent=None*)
Bases: *ly.dom.Statement*

Use this to create a LilyPond command supplying the name (or a Reference) when instantiating.

class ly.dom.**CommandEnclosed**(*name*, *parent=None*)
Bases: *ly.dom.StatementEnclosed*

Use this to print LilyPond commands that have a single bracket-enclosed list of arguments. The command name is supplied to the constructor.

class ly.dom.**Comment**(*text=u''*, *parent=None*)
Bases: *ly.dom.Text*

A LilyPond comment at the end of a line

after = 1

ly(*printer*)

class ly.dom.**Container**(*parent=None*)
Bases: *ly.dom.LyNode*

A node that concatenates its children on output

after

before

defaultSpace = u' '

ly(*printer*)

class ly.dom.**Context**(*contextName=u''*, *parent=None*)
Bases: *ly.dom.HandleVars*, *ly.dom.Section*

A context section for use inside Layout or Midi sections.

name = u'context'

class ly.dom.**ContextName**(*text=u''*, *parent=None*)
Bases: *ly.dom.Text*

Used to print a context name, like Score.

ly(*printer*)

class ly.dom.**ContextProperty**(*prop*, *context=None*, *parent=None*)
Bases: *ly.dom.Leaf*

A Context.property or Context.layoutObject construct. Call e.g. ContextProperty('aDueText', 'Staff') to get 'Staff.aDueText'.

ly(*printer*)

class ly.dom.**ContextType**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.Container*

new or context Staff = 'bla' with { } << music >>

A with (With) element is added automatically as the first child as soon as you use our convenience methods that manipulate the variables in with. If the with element is empty, it does not print anything. You should add one other music object to this.

addInstrumentNameEngraverIfNecessary()
Adds the Instrument_name_engraver to the node if it would need it to print instrument names.

after = 1

before = 1

ctype = None

getWith()
Gets the attached with clause. Creates it if not there.

isAtom = True

ly(*printer*)

class ly.dom.**CueVoice**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Devnull**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Document**(*parent=None*)
Bases: *ly.dom.Container*

A container type that puts everything on a new line. To be used as a full LilyPond document.

after = 1

defaultSpace = u'\n'

class ly.dom.**DrumMode**(*parent=None*)
Bases: *ly.dom.InputMode*

name = u'drummode'

class ly.dom.**DrumStaff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**DrumVoice**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Duration**(*dur*, *dots=0*, *factor=1*, *parent=None*)
Bases: *ly.dom.Leaf*

A duration with duration (in logarithmic form): (-2 ... 8), where -2 = longa, -1 = breve, 0 = 1, 1 = 2, 2 = 4, 3 = 8, 4 = 16, etc, dots (number of dots), factor (Fraction giving the scaling of the duration).

ly(*printer*)

class ly.dom.**Dynamics**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Enclosed**(*parent=None*)
Bases: *ly.dom.Container*

Encloses all children between brackets: { ... } If may_remove_brackets is True in subclasses, the brackets are removed if there is only one child and that child is an atom (i.e. a single LilyPond expression.

after = 0

before = 0

isAtom = True

ly(*printer*)

may_remove_brackets = False

post = u'}'

pre = u'{'

class ly.dom.**FigureMode**(*parent=None*)
Bases: *ly.dom.InputMode*

name = u'figuremode'

class ly.dom.**FiguredBass**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**FretBoards**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Global**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**GrandStaff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**GregorianTranscriptionStaff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**GregorianTranscriptionVoice**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**HandleVars**
Bases: object

A powerful mixin class to facilitate handling unique variable assignments inside a Container more. E.g.: >>> h = Header() >>> h['composer'] = "Johann Sebastian Bach" creates a subnode (by default Assignment) with the name 'composer', and that node again gets an autogenerated subnode of type QuotedString (if the argument wasn't already a Node).

childClass
alias of *Assignment*

ifbasestring(*func*)
Ensure that the method is only called for basestring objects. Otherwise the same method from the super class is called.

importNode(*obj*)
Try to interpret the object and transform it into a Node object of the right species.

class ly.dom.**Header**(*parent=None*)
Bases: *ly.dom.HandleVars*, *ly.dom.Section*

name = u'header'

class ly.dom.**Identifier**(*name=None*, *parent=None*)
Bases: *ly.dom.Leaf*

An identifier, prints as name. Name may be a string or a Reference object.

isAtom = True

ly(*printer*)

class ly.dom.**Include**(*text=u''*, *parent=None*)
Bases: *ly.dom.Line*

a LilyPond include statement

ly(*printer*)

class ly.dom.**InnerChoirStaff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**InnerStaffGroup**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**InputChords**(*parent=None*)
Bases: *ly.dom.ChordMode*

name = u'chords'

class ly.dom.**InputDrums**(*parent=None*)
Bases: *ly.dom.DrumMode*

name = u'drums'

class ly.dom.**InputFigures**(*parent=None*)
Bases: *ly.dom.FigureMode*

name = u'figures'

class ly.dom.**InputLyrics**(*parent=None*)
Bases: *ly.dom.LyricMode*

name = u'lyrics'

class ly.dom.**InputMode**(*parent=None*)
Bases: *ly.dom.StatementEnclosed*

The abstract base class for input modes such as lyricmode/lyrics, chordmode/chords etc.

class ly.dom.**InputNotes**(*parent=None*)
Bases: *ly.dom.NoteMode*

name = u'notes'

class ly.dom.**KeySignature**(*note=0*, *alter=0*, *mode=u'major'*, *parent=None*)
Bases: *ly.dom.Leaf*

A key signature expression, like:

key c major The pitch should be given in the arguments note and alter and is written out in the document's language.

ly(*printer*)

class ly.dom.**Layout**(*parent=None*)
Bases: *ly.dom.HandleVars*, *ly.dom.Section*

name = u'layout'

class ly.dom.**Leaf**(*parent=None*)
Bases: *ly.dom.LyNode*

A leaf node without children

class ly.dom.**Line**(*text=u''*, *parent=None*)
Bases: *ly.dom.Text*

A text node that claims its own line.

after = 1

before = 1

class ly.dom.**LineComment**(*text=u''*, *parent=None*)
Bases: *ly.dom.Comment*

A LilyPond comment that takes a full line

before = 1

class ly.dom.**LyNode**(*parent=None*)
Bases: *ly.node.WeakNode*

Base class for LilyPond objects, based on Node, which takes care of the tree structure.

after = 0

before = 0

concat(*other*)
Returns a string with newlines to concat this node to another one. If zero newlines are requested, an empty string is returned.

isAtom = False

ly(*printer*)
Returns printable output for this object. Can ask printer for certain settings, e.g. pitch language etc.

class ly.dom.**LyricMode**(*parent=None*)
Bases: *ly.dom.InputMode*

name = u'lyricmode'

class ly.dom.**Lyrics**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**LyricsTo**(*cid*, *parent=None*)
Bases: *ly.dom.LyricMode*

ly(*printer*)

name = u'lyricsto'

class ly.dom.**Mark**(*parent=None*)
Bases: *ly.dom.Statement*

The mark command.

name = u'mark'

class ly.dom.**Markup**(*parent=None*)
Bases: *ly.dom.StatementEnclosed*

The markup command. You can add many children, in that case Markup automatically prints { and } around them.

name = u'markup'

class ly.dom.**MarkupCommand**(*name*, *parent=None*)
Bases: *ly.dom.Command*

A markup command with more or no arguments, that does not auto-enclose its arguments. Useful for commands like note-by-number or hspace.

You must supply the name. Its arguments are its children. If one argument can be a markup list, use a Enclosed() construct for that.

class ly.dom.**MarkupEnclosed**(*name*, *parent=None*)
Bases: *ly.dom.CommandEnclosed*

A markup that auto-encloses all its arguments, like 'italic', 'bold' etc. You must supply the name.

class ly.dom.**MensuralStaff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**MensuralVoice**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Midi**(*parent=None*)
Bases: *ly.dom.HandleVars*, *ly.dom.Section*

name = u'midi'

class ly.dom.**Named**
Bases: object

Mixin to print a name before the contents of the container. format() is called on the self.name attribute, so it may also be a Reference.

ly(*printer*)

name = u'‘

class ly.dom.**Newline**(*parent=None*)
Bases: *ly.dom.LyNode*

A newline.

after = 1

class ly.dom.**NoteMode**(*parent=None*)
Bases: *ly.dom.InputMode*

name = u'notemode'

class ly.dom.**NoteNames**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Paper**(*parent=None*)
Bases: *ly.dom.HandleVars*, *ly.dom.Section*

name = u'paper'

class ly.dom.**Partial**(*dur*, *dots=0*, *factor=1*, *parent=None*)
Bases: *ly.dom.Named*, *ly.dom.Duration*

partial <duration> You should add a Duration element

after = 1

before = 1

name = u'partial'

class ly.dom.**PianoStaff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Pitch**(*octave=0*, *note=0*, *alter=0*, *parent=None*)
Bases: *ly.dom.Leaf*

A pitch with octave, note, alter. octave is specified by an integer, zero for the octave containing middle C. note is a number from 0 to 6, with 0 corresponding to pitch C and 6 corresponding to pitch B. alter is the number of whole tones for alteration (can be int or Fraction)

ly(*printer*)
Print the pitch in the preferred language.

class ly.dom.**Printer**
Bases: object

Performs certain operations on behalf of a LyNode tree, like quoting strings or translating pitch names, etc.

indent(*node*)
Return a formatted printout of node (and its children)

indentGen(*node*, *startIndent=0*)
A generator that walks on the output of the given node, and returns properly indented LilyPond code.

primary_quote_left = u'\u2018'

primary_quote_right = u'\u2019'

quoteString(*text*)

secondary_quote_left = u'\u201c'

secondary_quote_right = u'\u201d'

class ly.dom.**QuotedString**(*text=u''*, *parent=None*)
Bases: *ly.dom.Text*

A string that is output inside double quotes.

isAtom = True

ly(*printer*)

class ly.dom.**Reference**(*name=u''*)
Bases: object

A simple object that keeps a name, to use as a (context) identifier. Set the name attribute to the name you want to display, and on all places in the document the name will show up.

class ly.dom.**Relative**(*parent=None*)
Bases: *ly.dom.Statement*

relative <pitch> music

You should add a Pitch (optionally) and another music object, e.g. Sim or Seq, etc.

name = u'relative'

class ly.dom.**RhythmicStaff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Scheme**(*text=u''*, *parent=None*)
Bases: *ly.dom.Text*

A Scheme expression, without the extra # prepended

isAtom = True

ly(*printer*)

class ly.dom.**SchemeLily**(*parent=None*)
Bases: *ly.dom.Enclosed*

A LilyPond expression between #{ #} (inside scheme)

post = u'#}'

pre = u'#{'

class ly.dom.**SchemeList**(*parent=None*)
Bases: *ly.dom.Enclosed*

A list of items enclosed in parentheses

ly(*printer*)

post = u')'

pre = u'('

class ly.dom.**Score**(*parent=None*)
Bases: *ly.dom.Section*

name = u'score'

class ly.dom.**ScoreContext**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

Represents the Score context in LilyPond, but the name would collide with the Score class that represents score { } constructs.

Because the latter is used more often, use ScoreContext to get new Score etc.

ctype = u'Score'

class ly.dom.**Section**(*parent=None*)
Bases: *ly.dom.StatementEnclosed*

Very much like a Statement. Use as base class for book { }, score { } etc. By default never removes the brackets and always starts on a new line.

after = 1

before = 1

may_remove_brackets = False

class ly.dom.**Seq**(*parent=None*)
Bases: *ly.dom.Enclosed*

An SequentialMusic expression between { }

post = u'}'

pre = u'{'

class ly.dom.**Seqr**(*parent=None*)
Bases: *ly.dom.Seq*

may_remove_brackets = True

class ly.dom.**Sim**(*parent=None*)
Bases: *ly.dom.Enclosed*

An SimultaneousMusic expression between << >>

post = u'>>'

pre = u'<<'

class ly.dom.**Simr**(*parent=None*)
Bases: *ly.dom.Sim*

may_remove_brackets = True

class ly.dom.**Staff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**StaffGroup**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Statement**(*parent=None*)
Bases: *ly.dom.Named*, *ly.dom.Container*

Base class for statements with arguments. The statement is read in the name attribute, the arguments are the children.

before = 0

isAtom = True

class ly.dom.**StatementEnclosed**(*parent=None*)
Bases: *ly.dom.Named*, *ly.dom.Enclosed*

Base class for LilyPond commands that have a single bracket-enclosed list of arguments.

may_remove_brackets = True

class ly.dom.**TabStaff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**TabVoice**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Tempo**(*duration*, *value*, *parent=None*)
Bases: *ly.dom.Container*

A tempo setting, like: tempo 4 = 100 May have a child markup or quoted string.

after = 1

before = 1

ly(*printer*)

class ly.dom.**Text**(*text=u''*, *parent=None*)
Bases: *ly.dom.Leaf*

A leaf node with arbitrary text

ly(*printer*)

class ly.dom.**TextDur**(*text=u''*, *parent=None*)
Bases: *ly.dom.AddDuration*, *ly.dom.Text*

A text note with an optional duration as child.

class ly.dom.**TimeSignature**(*num*, *beat*, *parent=None*)
Bases: *ly.dom.Leaf*

A time signature, like: time 4/4

ly(*printer*)

class ly.dom.**Transposition**(*parent=None*)
Bases: *ly.dom.Statement*

transposition <pitch> You should add a Pitch.

name = u'transposition'

class ly.dom.**UserContext**(*ctype*, *cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

Represents a context the user creates. e.g. new MyStaff = cid << music >>

class ly.dom.**VaticanaStaff**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**VaticanaVoice**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**Version**(*text=u''*, *parent=None*)
Bases: *ly.dom.Line*

a LilyPond version instruction

ly(*printer*)

class ly.dom.**Voice**(*cid=None*, *new=True*, *parent=None*)
Bases: *ly.dom.ContextType*

class ly.dom.**VoiceSeparator**(*parent=None*)
Bases: *ly.dom.Leaf*

A Voice Separator: \

ly(*printer*)

class ly.dom.**With**(*parent=None*)
Bases: *ly.dom.HandleVars*, *ly.dom.Section*

If this item has no children, it prints nothing.

after = 0

before = 0

ly(*printer*)

name = u'with'

ly.duration module

LilyPond information and logic concerning durations

ly.duration.**base_scaling**(*tokens*)
Return (base, scaling) as two Fractions for the list of tokens.

ly.duration.**base_scaling_string**(*duration*)
Return (base, scaling) as two Fractions for the specified string.

ly.duration.**format_fraction**(*value*)
Format the value as "5/1" etc.

ly.duration.**fraction**(*tokens*)
Return the duration of the Duration tokens as a Fraction.

ly.duration.**fraction_string**(*duration*)
Return the duration of the specified string as a Fraction.

ly.duration.**tostring**(*dur*, *dots=0*, *factor=1*)
Returns the LilyPond string representation of a given logarithmic duration.

Supports values from -3 up to and including 11. -2 = 'longa', 0 = '1' (whole note), etc.

Adds the number of dots (defaults to 0) and the fraction factor if given.

ly.etreeutil module

Utility functions for use with xml.etree.ElementTree.

ly.etreeutil.**indent**(*elem*, *indent_string=u' '*, *level=0*)
Indent the XML in element.

Text content that is already non-whitespace is not changed.

ly.etreeutil.**isblank**(*s*)
Return True if s is empty or whitespace only.

ly.indent module

Indent and auto-indent.

class ly.indent.**Indenter**
Bases: object

compute_indent(*document*, *block*)
Return the indent the specified block should have.

Returns False if the block is not indentable, e.g. when it is part of a multiline string.

This method is used to determine the indent of one line, and just looks to previous lines, copying the indent of the line where the current indent depth starts, and/or adding a level of indent or alignment space.

Use this method only for one line or the first of a group you're indenting.

decrease_indent(*cursor*)
Manually remove one level of indent from all lines of cursor.

get_indent(*document*, *block*)
Return the indent the block currently has.

Returns False if the block is not indentable, e.g. when it is part of a multiline string.

increase_indent(*cursor*)
Manually add indent to all lines of cursor.

indent(*cursor*, *indent_blank_lines=False*)
Indent all lines in the cursor's range.

If indent_blank_lines is True, the indent of blank lines is made larger if necessary. If False (the default), the indent of blank lines if not changed if it is shorter than it should be.

indent_tabs = False

indent_width = 2

class ly.indent.**Line**(*document*, *block*)
Bases: object

Brings together all relevant information about a line (block).

is_alignable_scheme_keyword(*token*)
Return True if token is an alignable Scheme word like "if", etc.

ly.node module

The Node class.

(c) 2008-2011 Wilbert Berendsen License: GPL.

This module contains the Node class that can be used as a simple DOM (Document Object Model) for building a tree structure.

A Node has children with list-like access methods and keeps also a reference to its parent. A Node can have one parent; appending a Node to another Node causes it to be removed from its parent node (if any).

To iterate over the children of a Node:

```
for n in node:
    do_something(n)
```

To get the list of children of a Node:

```
children = list(node)
```

Of course you can get the children directly using:

```
child = node[3]
```

You should inherit from Node to make meaningful tree node types, e.g. to add custom attributes or multiple sub-types.

A WeakNode class is provided as well, which uses a weak reference to the parent, so that no cyclic references are created which might improve garbage collection.

class ly.node.**Node**(*parent=None*)
Bases: object

A list-like class to build tree structures with.

ancestors()
Climb the tree up over the parents.

append(*node*)
Append a node to the current node.

It will be reparented, that means it will be removed from it's former parent if it had one.

backward()
Iterate (backwards) over the preceding siblings.

clear()
Remove all children.

copy()
Return a deep copy of the node and its children

descendants(*depth=-1*)
Yield all the descendants, in tree order. Same as iter_depth().

dump()
Return a string representation of the tree.

extend(*iterable*)
Append every Node from iterable.

find(*cls*, *depth=-1*)
Yield all descendants if they are an instance of cls.

cls may also be a tuple of classes. This method uses iter_depth().

find_child(*cls*, *depth=-1*)
Return the first descendant that's an instance of cls.

cls may also be a tuple of classes. This method uses iter_rings().

find_children(*cls*, *depth=-1*)
Yield all descendants if they are an instance of cls.

cls may also be a tuple of classes. This method uses iter_rings().

find_parent(*cls*)
Find an ancestor that's an instance of the given class.

cls may also be a tuple of classes.

forward()
Iterate over the following siblings.

index(*node*)
Return the index of the given child node.

insert(*index*, *node*)
Insert a node at the specified index.

insert_before(*other*, *node*)
Insert a node before the other node.

is_descendant_of(*parent*)
Return True if self is a descendant of parent, else False.

iter_depth(*depth=-1*)
Iterate over all the children, and their children, etc.

Set depth to restrict the search to a certain depth, -1 is unrestricted.

iter_rings(*depth=-1*)
Iterate over the children in rings, depth last.

This method returns the closest descendants first. Set depth to restrict the search to a certain depth, -1 is unrestricted.

next_sibling()
Return the sibling object just after us in our parents list.

Returns None if this is the last child, or if we have no parent.

parent()
The parent, or None if the node has no parent.

previous_sibling()
Return the sibling object just before us in our parents list.

Returns None if this is the first child, or if we have no parent.

remove(*node*)
Remove the given child node.

replace(*old*, *new*)
Replace a child node with another node.

sort(*key=None*, *reverse=False*)
Sorts the children, optionally using the key function.

Using a key function is recommended, or you must add comparison methods to your Node subclass.

toplevel()
Return the toplevel parent Node of this node.

unlink()
Remove all children and unlink() them as well.

class ly.node.**WeakNode**(*parent=None*)
Bases: *ly.node.Node*

A Node type using a weak reference to the parent.

parent()
The parent, or None if the node has no parent.

ly.pkginfo module

Meta-information about the LY package.

This information is used by the install script, and also for the command `ly --version`.

ly.pkginfo.**name = u'python-ly'**
name of the package

ly.pkginfo.**version = u'0.9.5'**
the current version

ly.pkginfo.**description = u'Tool and library for manipulating LilyPond files'**
short description

ly.pkginfo.**long_description = u'The python-ly package provides a Python library and a commandline tool that can b**
long description

ly.pkginfo.**maintainer = u'Wilbert Berendsen'**
maintainer name

ly.pkginfo.**maintainer_email = u'info@frescobaldi.org'**
maintainer email

ly.pkginfo.**url = u'https://github.com/wbsoft/python-ly'**
homepage

ly.pkginfo.**license = u'GPL'**
license

ly.reformat module

Formatting tools to improve the readability of a ly.document.Document without changing the semantic meaning of the LilyPond source.

Basically the tools only change whitespace to make the source-code more readable.

See also ly.indent.

ly.reformat.**break_indenters**(*cursor*)
Add newlines around indent and dedent tokens where needed.

If there is stuff after a { or << (that's not closed on the same line) it is put on a new line, and if there if stuff before a } or >>, the } or >> is put on a new line.

It is necessary to run the indenter again over the same part of the document, as it will look garbled with the added newlines.

ly.reformat.**move_long_comments**(*cursor*)
Move line comments with more than 2 comment characters to column 0.

ly.reformat.**reformat**(*cursor*, *indenter*)
A do-it-all function improving the LilyPond source formatting.

ly.reformat.**remove_trailing_whitespace**(*cursor*)
Removes whitespace from all lines in the cursor's range.

ly.rhythm module

Implementation of the tools to edit durations of selected music.

Durations are represented simply by lists of ly.lex.lilypond.Duration tokens.

All functions expect a ly.document.Cursor with the selected range.

class ly.rhythm.**music_item**(*tokens*, *dur_tokens*, *may_remove*, *insert_pos*, *pos*, *end*)
Bases: tuple

dur_tokens
Alias for field number 1

end
Alias for field number 5

insert_pos
Alias for field number 3

may_remove
Alias for field number 2

pos
Alias for field number 4

tokens
Alias for field number 0

ly.rhythm.**music_items**(*cursor*, *command=False*, *chord=False*, *partial=1*)
Yield music_item instances describing rests, skips or pitches.

cursor is a ly.document.Cursor instance.

The following keyword arguments can be used:

- command: whether to allow pitches in \relative, \transpose, etc.
- chord: whether to allow pitches inside chords.
- partial: ly.document.INSIDE (default), PARTIAL or OUTSIDE. See the documentation of ly.document.Source.__init__().

ly.rhythm.**music_tokens**(*source*, *command=False*, *chord=False*)
DEPRECATED. Yield lists of tokens describing rests, skips or pitches.

source is a ly.document.Source instance following the state.

The following keyword arguments can be used:

•command: whether to allow pitches in \relative, \transpose, etc.

•chord: whether to allow pitches inside chords.

This function is deprecated and will be removed. You should use music_items() instead.

ly.rhythm.**preceding_duration**(*cursor*)
Return a preceding duration before the cursor, or an empty list.

ly.rhythm.**remove_dups**(*iterable*)
Change reoccurring strings to '' in iterable.

ly.rhythm.**rhythm_dot**(*cursor*)
Add a dot to all durations.

ly.rhythm.**rhythm_double**(*cursor*)
Doubles all duration values.

ly.rhythm.**rhythm_explicit**(*cursor*)
Make all durations explicit.

ly.rhythm.**rhythm_extract**(*cursor*)
Return a list of the durations from the cursor's range.

ly.rhythm.**rhythm_halve**(*cursor*)
Halves all duration values.

ly.rhythm.**rhythm_implicit**(*cursor*)
Remove reoccurring durations.

ly.rhythm.**rhythm_implicit_per_line**(*cursor*)
Remove reoccurring durations, but always write one on a new line.

ly.rhythm.**rhythm_overwrite**(*cursor*, *durations*)
Apply a list of durations to the cursor's range.

The durations list looks like ["4", "8", "", "16.",] etc.

ly.rhythm.**rhythm_remove**(*cursor*)
Remove all durations.

ly.rhythm.**rhythm_remove_fraction_scaling**(*cursor*)
Remove the scaling containing fractions (like `*1/3`) from all durations.

ly.rhythm.**rhythm_remove_scaling**(*cursor*)
Remove the scaling (like `*3`, `*1/3`) from all durations.

ly.rhythm.**rhythm_undot**(*cursor*)
Remove one dot from all durations.

ly.util module

Utility functions.

ly.util.**int2letter**(*number*, *chars='ABCDEFGHIJKLMNOPQRSTUVWXYZ'*)
Converts an integer to one or more letters.

E.g. 1 -> A, 2 -> B, ... 26 -> Z, 27 -> AA, etc. Zero returns the empty string.

chars is the string to pick characters from, defaulting to string.ascii_uppercase.

ly.util.**int2roman**(*n*)
Convert an integer value to a roman number string.

E.g. 1 -> "I", 12 -> "XII", 2015 -> "MMXV"

n has to be > 1.

ly.util.**int2text**(*number*)
Converts an integer to the English language name of that integer.

E.g. converts 1 to "One". Supports numbers 0 to 999999. This can be used in LilyPond identifiers (that do not support digits).

ly.util.**mkid**(**args*)
Makes a lower-camel-case identifier of the strings in args.

All strings are concatenated with the first character of every string uppercased, except for the first character, which is lowercased.

Examples:

```
mkid("Violin") ==> "violin"
mkid("soprano", "verse") ==> "sopranoVerse"
mkid("scoreOne", "choirII") ==> "scoreOneChoirII"
```

ly.words module

LilyPond reserved words for auto completion and highlighting.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W