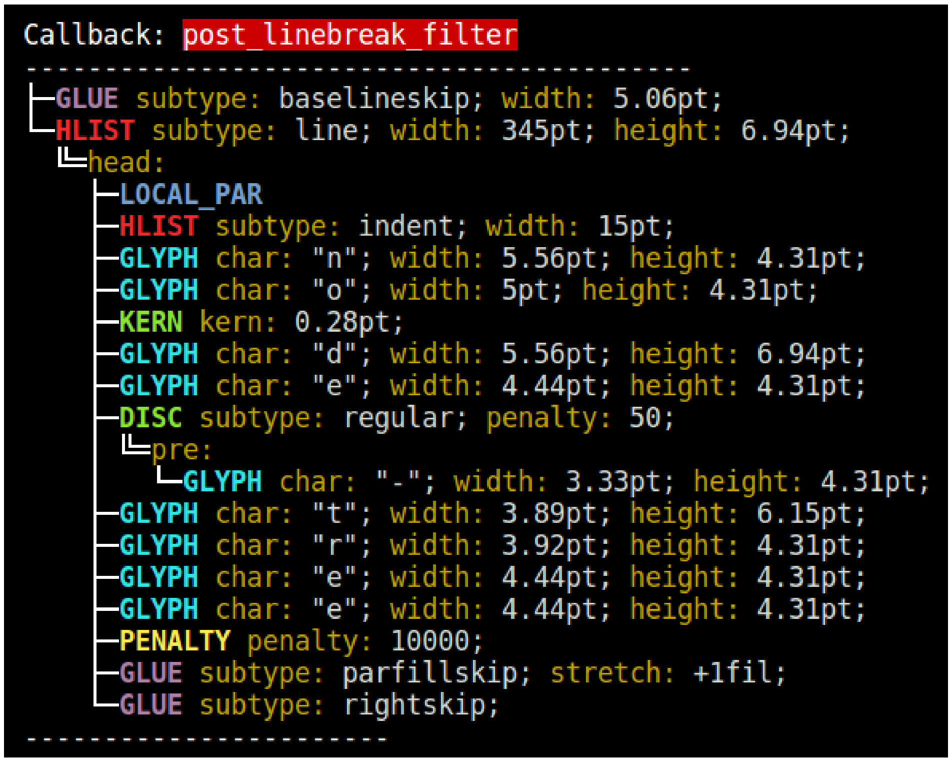
The nodetree package

Josef Friedrich
josef@friedrich.rocks
github.com/Josef-Friedrich/nodetree

v1.2 from 2016/07/18



```
Callback: post_linebreak_filter
------------------------------------------
├─GLUE subtype: baselineskip; width: 5.06pt;
└─HLIST subtype: line; width: 345pt; height: 6.94pt;
  ╚═head:
    ├─LOCAL_PAR
    ├─HLIST subtype: indent; width: 15pt;
    ├─GLYPH char: "n"; width: 5.56pt; height: 4.31pt;
    ├─GLYPH char: "o"; width: 5pt; height: 4.31pt;
    ├─KERN kern: 0.28pt;
    ├─GLYPH char: "d"; width: 5.56pt; height: 6.94pt;
    ├─GLYPH char: "e"; width: 4.44pt; height: 4.31pt;
    ├─DISC subtype: regular; penalty: 50;
    │ ╚═pre:
    │   └─GLYPH char: "-"; width: 3.33pt; height: 4.31pt;
    ├─GLYPH char: "t"; width: 3.89pt; height: 6.15pt;
    ├─GLYPH char: "r"; width: 3.92pt; height: 4.31pt;
    ├─GLYPH char: "e"; width: 4.44pt; height: 4.31pt;
    ├─GLYPH char: "e"; width: 4.44pt; height: 4.31pt;
    ├─PENALTY penalty: 10000;
    ├─GLUE subtype: parfillskip; stretch: +1fil;
    └─GLUE subtype: rightskip;
-----------------------
```

Contents

1 Abstract

`nodetree` is a development package that visualizes the structure of node lists. `nodetree` shows its debug informations in the consoles' output when you compile a LuaTEX file. It uses a similar visual representation for node lists as the UNIX `tree` command uses for a folder structure.

Node lists are the main building blocks of each document generated by the TEX engine *LuaTEX*. The package `nodetree` doesn't change the rendered document. The tree view can only be seen when using a terminal to generate the document.

`nodetree` is inspired by a gist from Patrick Gundlach.

2 Usage

The package `nodetree` can be used both with LuaTEX and LuaLATEX. You have to use both engines in a text console. Run for example `luatex luatex-test.tex` to list the nodes using LuaTEX.

```
\input{nodetree.tex}
\nodetreeregister{postline}

Lorem ipsum dolor.
\bye
```

Or run `lualatex lualatex-test.tex` to show a node tree using LuaLATEX. In LuaLATEX you can omit `\nodetreeregister{postline}`. `\usepackage{nodetree}` registers automatically the `post_linebreak_filter`. If you don't want debug the `post_linebreak_filter` use `\nodetreeunregister{postline}`.

```
\documentclass{article}
\usepackage{nodetree}

\begin{document}
Lorem ipsum dolor.
\end{document}
```

2.1 Debug nodes inside Lua code

Use the Lua function `nodetree.analyze(head)` to debug nodes inside your Lua code. The following code snippet demonstrates the usage in LuaTEX. `head` is the current node.

```
\input{nodetree.tex}

\directlua{
  local test = function (head)
    nodetree.analyze(head)
  end
  callback.register('post_linebreak_filter', test)
}
```

```
Lorem ipsum dolor.
\bye
```

This example illustrates how the function has to be applied in LuaLaTeX.

```
\documentclass{article}
\usepackage{nodetree}

\begin{document}

\directlua{
  local test = function (head)
    nodetree.analyze(head)
  end
  luatexbase.add_to_callback('post_linebreak_filter', test, 'test')
}

Lorem ipsum dolor.
\end{document}
```

3 Macros

3.1 `\nodetreeregister`

`\nodetreeregister` `\nodetreeregister{`⟨*callbacks*⟩`}`: The argument `{`⟨*callbacks*⟩`}` takes a comma separated list of callback aliases as described in (→ 4.1).

3.2 `\nodetreeunregister`

`\nodetreeunregister` `\nodetreeunregister{`⟨*callbacks*⟩`}`: The argument `{`⟨*callbacks*⟩`}` takes a comma separated list of callback aliases as described in (→ 4.1).

3.3 `\nodetreeoption`

`\nodetreeoption` `\nodetreeoption[`⟨*option*⟩`]{`⟨*value*⟩`}`: (→ 4) This macro sets the option `[`⟨*option*⟩`]` to the value `{`⟨*value*⟩`}`.

3.4 `\nodetreeset`

`\nodetreeset` `\nodetreeset{`⟨*kv-options*⟩`}`: This macro can only be used in LuaLaTeX. `{`⟨*kv-options*⟩`}` are key value pairs.

```
\nodetreeset{color=no,callbacks={hpack,vpack},verbosity=2}
```

Alias (short)	**Alias (longer)**	**Callback**
`contribute`	`contributefilter`	`contribute_filter`
`buildpage`	`buildpagefilter`	`buildpage_filter`
`preline`	`prelinebreakfilter`	`pre_linebreak_filter`
`line`	`linebreakfilter`	`linebreak_filter`
`append`	`appendtovlistfilter`	`append_to_vlist_filter`
`postline`	`postlinebreakfilter`	`post_linebreak_filter`
`hpack`	`hpackfilter`	`hpack_filter`
`vpack`	`vpackfilter`	`vpack_filter`
`hpackq`	`hpackquality`	`hpack_quality`
`vpackq`	`vpackquality`	`vpack_quality`
`process`	`processrule`	`process_rule`
`preout`	`preoutputfilter`	`pre_output_filter`
`hyph`	`hyphenate`	`hyphenate`
`liga`	`ligaturing`	`ligaturing`
`kern`	`kerning`	`kerning`
`insert`	`insertlocalpar`	`insert_local_par`
`mhlist`	`mlisttohlist`	`mlist_to_hlist`

Figure 1: The callback aliases

4 Options

4.1 Option `callback`

The option `callback` is the most important setting of the package. You have to specify one alias to select the `callback`. Because of the underscores the callback name contains it can not set by its technical name (→ Figure 1).

This macros process callback options: `\nodetreeregister{`⟨*callbacks*⟩`}`, `\nodetreeunregister{`⟨*callbacks*⟩`}`, `\nodetreeset{`⟨*callback=<callbacks>*⟩`}` and `\usepackage[`⟨*callback=<callbacks>*⟩`]{`⟨*nodetree*⟩`}`.

Use commas to specify mulitple callbacks. Avoid using whitespaces:

```
\nodetreeregister{preline,line,postline}
```

Wrap your callback aliases in curly braces for the macro `\nodetreeset`:

```
\nodetreeset{callback={preline,line,postline}}
```

The same applies for the macro `\usepackage`:

```
\usepackage{callback={preline,line,postline}}
```

4.2 Option `verbosity`

Higher integer values result in a more verbose output. The default value for this options is `1`. At the moment only verbosity level `2` is implemented.

Unit	**Description**
pt	Point 1/72.27 inch. The conversion to metric units, to two decimal places, is 1 point = 2.85 mm = 28.45 cm.
pc	Pica, 12 pt
in	Inch, 72.27 pt
bp	Big point, 1/72 inch. This length is the definition of a point in PostScript and many desktop publishing systems.
cm	Centimeter
mm	Millimeter
dd	Didot point, 1.07 pt
cc	Cicero, 12 dd
sp	Scaled point, 1/65536 pt

Figure 2: Fixed units

Unit	**Description**
ex	x-height of the current font
em	Width of the capital letter M

Figure 3: Relative units

4.3 Option `color`

The default option for `color` is `colored`. Use any other string (for example `none` or `no`) to disable the colored terminal output of the package.

```
\usepackage[color=no]{nodetree}
```

4.4 Option `unit`

The option `unit` sets the length unit to display all length values of the nodes. The default option for `unit` is `pt`. See figure 2 and 3 for possible values.

4.5 Option `decimalplaces`

The options `decimalplaces` sets the number of decimal places for some node fields.

```
\nodetreeoption[decimalplaces]{4}
```

gets

```
├─GLYPH char: "a"; width: 5pt; height: 4.3055pt;
```

If `decimalplaces` is set to `0` only integer values are shown.

```
├─GLYPH char: "a"; width: 5pt; height: 4pt;
```

5 Visual tree structure

5.1 Two different connections

Nodes in LuaTeX are connected. The `nodetree` package distinguishs between the `list` and `field` connections.

- `list`: Nodes, which are double connected by `next` and `previous` fields.
- `field`: Connections to nodes by other fields than `next` and `previous` fields, e. g. `head`, `pre`.

5.2 Unicode characters to show the tree view

The package `nodetree` uses the unicode box drawing symbols. Your default terminal font should contain this characters to obtain the tree view. Eight box drawing characters are necessary.

Code	**Character**	**Name**
U+2500	─	BOX DRAWINGS LIGHT HORIZONTAL
U+2502	│	BOX DRAWINGS LIGHT VERTICAL
U+2514	└	BOX DRAWINGS LIGHT UP AND RIGHT
U+251C	├	BOX DRAWINGS LIGHT VERTICAL AND RIGHT
U+2550	═	BOX DRAWINGS DOUBLE HORIZONTAL
U+2551	║	BOX DRAWINGS DOUBLE VERTICAL
U+255A	╚	BOX DRAWINGS DOUBLE UP AND RIGHT
U+2560	╠	BOX DRAWINGS DOUBLE VERTICAL AND RIGHT

For `list` connections *light* characters are shown.

```
│ │
│ ├─list1
│ └─list2
└─list3
```

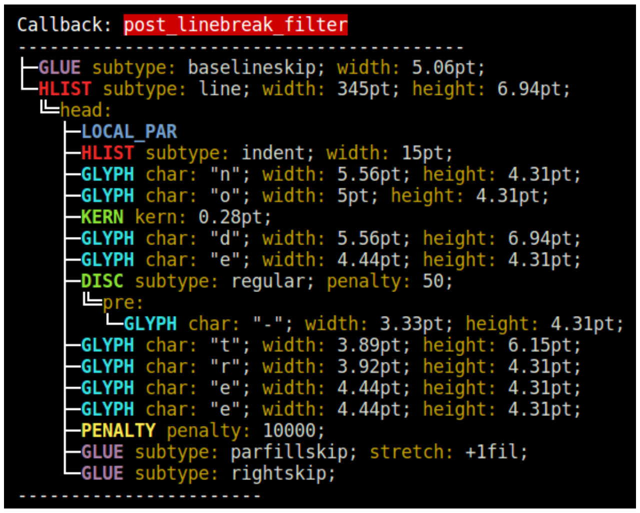
`field` connections are visialized by *Double* characters.

```
║ ║
║ ╠═field1
║ ╚═field2
╚═field3
```

6 Examples

6.1 The node list of the package name

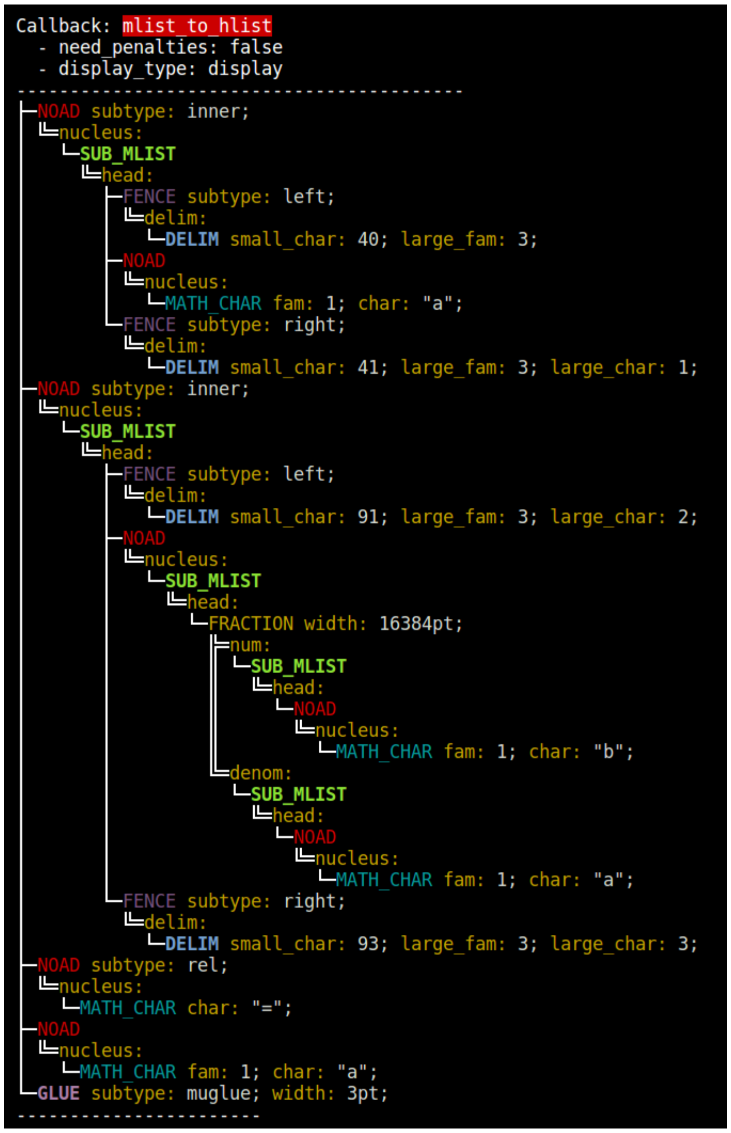
```
\documentclass{article}
\usepackage{nodetree}
\begin{document}
nodetree
\end{document}
```



```
Callback: post_linebreak_filter
------------------------------------------
├─GLUE subtype: baselineskip; width: 5.06pt;
└─HLIST subtype: line; width: 345pt; height: 6.94pt;
  ╚═head:
    ├─LOCAL_PAR
    ├─HLIST subtype: indent; width: 15pt;
    ├─GLYPH char: "n"; width: 5.56pt; height: 4.31pt;
    ├─GLYPH char: "o"; width: 5pt; height: 4.31pt;
    ├─KERN kern: 0.28pt;
    ├─GLYPH char: "d"; width: 5.56pt; height: 6.94pt;
    ├─GLYPH char: "e"; width: 4.44pt; height: 4.31pt;
    ├─DISC subtype: regular; penalty: 50;
    │ ╚═pre:
    │   └─GLYPH char: "-"; width: 3.33pt; height: 4.31pt;
    ├─GLYPH char: "t"; width: 3.89pt; height: 6.15pt;
    ├─GLYPH char: "r"; width: 3.92pt; height: 4.31pt;
    ├─GLYPH char: "e"; width: 4.44pt; height: 4.31pt;
    ├─GLYPH char: "e"; width: 4.44pt; height: 4.31pt;
    ├─PENALTY penalty: 10000;
    ├─GLUE subtype: parfillskip; stretch: +1fil;
    └─GLUE subtype: rightskip;
-----------------------
```

6.2 The node list of a mathematical formula

```
\documentclass{article}
\usepackage[callback={mhlist}]{nodetree}
\begin{document}
\[\left(a\right)\left[\frac{b}{a}\right]=a\,\]
\end{document}
```

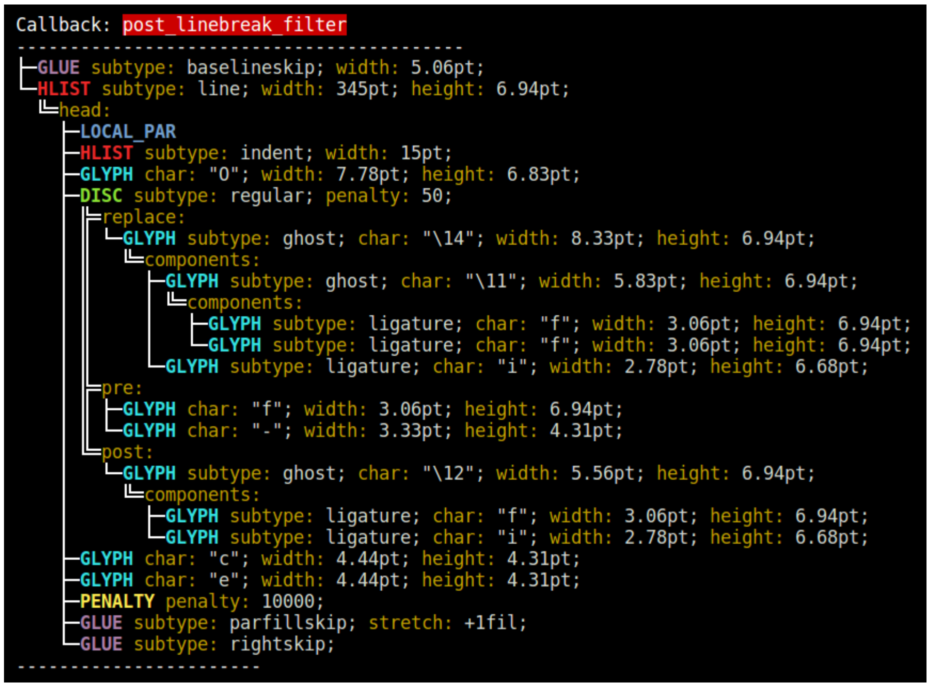


```
Callback: mlist_to_hlist
  - need_penalties: false
  - display_type: display
------------------------------------------
├─NOAD subtype: inner;
│ ╚═nucleus:
│   └─SUB_MLIST
│     ╚═head:
│       ├─FENCE subtype: left;
│       │ ╚═delim:
│       │   └─DELIM small_char: 40; large_fam: 3;
│       ├─NOAD
│       │ ╚═nucleus:
│       │   └─MATH_CHAR fam: 1; char: "a";
│       └─FENCE subtype: right;
│         ╚═delim:
│           └─DELIM small_char: 41; large_fam: 3; large_char: 1;
├─NOAD subtype: inner;
│ ╚═nucleus:
│   └─SUB_MLIST
│     ╚═head:
│       ├─FENCE subtype: left;
│       │ ╚═delim:
│       │   └─DELIM small_char: 91; large_fam: 3; large_char: 2;
│       ├─NOAD
│       │ ╚═nucleus:
│       │   └─SUB_MLIST
│       │     ╚═head:
│       │       └─FRACTION width: 16384pt;
│       │         ╠═num:
│       │         ║ └─SUB_MLIST
│       │         ║   ╚═head:
│       │         ║     └─NOAD
│       │         ║       ╚═nucleus:
│       │         ║         └─MATH_CHAR fam: 1; char: "b";
│       │         ╚═denom:
│       │           └─SUB_MLIST
│       │             ╚═head:
│       │               └─NOAD
│       │                 ╚═nucleus:
│       │                   └─MATH_CHAR fam: 1; char: "a";
│       └─FENCE subtype: right;
│         ╚═delim:
│           └─DELIM small_char: 93; large_fam: 3; large_char: 3;
├─NOAD subtype: rel;
│ ╚═nucleus:
│   └─MATH_CHAR char: "=";
├─NOAD
│ ╚═nucleus:
│   └─MATH_CHAR fam: 1; char: "a";
└─GLUE subtype: muglue; width: 3pt;
----------------------
```

6.3 The node list of the word *Office*

The characters *ffi* are deeply nested in a discretionary node.

```
\documentclass{article}
\usepackage{nodetree}
\begin{document}
Office
\end{document}
```



```
Callback: post_linebreak_filter
------------------------------------------
├─GLUE subtype: baselineskip; width: 5.06pt;
└─HLIST subtype: line; width: 345pt; height: 6.94pt;
  ╚═head:
    ├─LOCAL_PAR
    ├─HLIST subtype: indent; width: 15pt;
    ├─GLYPH char: "O"; width: 7.78pt; height: 6.83pt;
    ├─DISC subtype: regular; penalty: 50;
    │ ╠═replace:
    │ ║ └─GLYPH subtype: ghost; char: "\14"; width: 8.33pt; height: 6.94pt;
    │ ║   ╚═components:
    │ ║     ├─GLYPH subtype: ghost; char: "\11"; width: 5.83pt; height: 6.94pt;
    │ ║     │ ╚═components:
    │ ║     │   ├─GLYPH subtype: ligature; char: "f"; width: 3.06pt; height: 6.94pt;
    │ ║     │   └─GLYPH subtype: ligature; char: "f"; width: 3.06pt; height: 6.94pt;
    │ ║     └─GLYPH subtype: ligature; char: "i"; width: 2.78pt; height: 6.68pt;
    │ ╠═pre:
    │ ║ ├─GLYPH char: "f"; width: 3.06pt; height: 6.94pt;
    │ ║ └─GLYPH char: "-"; width: 3.33pt; height: 4.31pt;
    │ ╚═post:
    │   └─GLYPH subtype: ghost; char: "\12"; width: 5.56pt; height: 6.94pt;
    │     ╚═components:
    │       ├─GLYPH subtype: ligature; char: "f"; width: 3.06pt; height: 6.94pt;
    │       └─GLYPH subtype: ligature; char: "i"; width: 2.78pt; height: 6.68pt;
    ├─GLYPH char: "c"; width: 4.44pt; height: 4.31pt;
    ├─GLYPH char: "e"; width: 4.44pt; height: 4.31pt;
    ├─PENALTY penalty: 10000;
    ├─GLUE subtype: parfillskip; stretch: +1fil;
    └─GLUE subtype: rightskip;
-----------------------
```

7 Implementation

7.1 The file nodetree.tex

```
\directlua{
  nodetree = require('nodetree')
  nodetree.set_option('engine', 'luatex')
  nodetree.set_default_options()
}
```

\nodetreeoption

```
\def\nodetreeoption[#1]#2{
  \directlua{
    nodetree.set_option('#1', '#2')
  }
}
```

\nodetreeregister

```
\def\nodetreeregister#1{
  \directlua{
    nodetree.set_option('callback', '#1')
    nodetree.register_callbacks()
  }
}
```

\nodetreeunregister

```
\def\nodetreeunregister#1{
  \directlua{
    nodetree.set_option('callback', '#1')
    nodetree.unregister_callbacks()
  }
}
```

7.2 The file nodetree.sty

```
\input{nodetree}
\directlua{
  nodetree.set_option('engine', 'lualatex')
}
```

```
\RequirePackage{kvoptions}
```

```
\SetupKeyvalOptions{
  family=NT,
  prefix=NT@
}
```

```
\DeclareStringOption[term]{channel}
\define@key{NT}{channel}[]{\nodetreeoption[channel]{#1}}

\DeclareStringOption[postlinebreak]{callback}
\define@key{NT}{callback}[]{\nodetreeoption[callback]{#1}}

\DeclareStringOption[1]{verbosity}
\define@key{NT}{verbosity}[]{\nodetreeoption[verbosity]{#1}}

\DeclareStringOption[colored]{color}
\define@key{NT}{color}[]{\nodetreeoption[color]{#1}}

\DeclareStringOption[1]{unit}
\define@key{NT}{unit}[]{\nodetreeoption[unit]{#1}}

\DeclareStringOption[1]{decimalplaces}
\define@key{NT}{decimalplaces}[]{\nodetreeoption[decimalplaces]{#1}}

\ProcessKeyvalOptions*
\directlua{
  nodetree.set_default_options()
  nodetree.register_callbacks()
}
```

\nodetreeset

```
\newcommand{\nodetreeset}[1]{\setkeys{nodetree}{#1}}
```

7.3 The file nodetree.lua

```
local nodex = {}

local tpl = {}

local tree = {}
```

Nodes in LuaTEX are connected. The nodetree view distinguishs between the `list` and `field` connections.

- `list`: Nodes, which are double connected by `next` and `previous` fields.
- `field`: Connections to nodes by other fields than `next` and `previous` fields, e. g. `head`, `pre`.

The lua table named `tree.state` holds state values for the current tree item.

```
%  tree.state:
%    - 1:
%      - list: continue
%      - field: stop
%    - 2:
```

```
%      - list: continue
%      - field: stop
```

```
tree.state = {}
```

```
local callbacks = {}
```

```
local base = {}
```

```
local options = {}
```

7.3.1 nodex — Extend the node library

Get the node id form, e. g.:

```
% <node    nil <    172 >    nil : hlist 2>
```

```
function nodex.node_id(n)
  return string.gsub(tostring(n), '^<node%s+%S+%s+<%s+(%d+).*', '%1')
end
```

```
function nodex.subtype(n)
  local typ = node.type(n.id)
  local subtypes = {
```

hlist (0)

```
    hlist = {
      [0] = 'unknown',
      [1] = 'line',
      [2] = 'box',
      [3] = 'indent',
      [4] = 'alignment',
      [5] = 'cell',
      [6] = 'equation',
      [7] = 'equationnumber',
    },
```

vlist (1)

```
    vlist = {
      [0] = 'unknown',
      [4] = 'alignment',
      [5] = 'cell',
    },
```

rule (2)

```
  rule = {
    [0] = 'unknown',
    [1] = 'box',
    [2] = 'image',
    [3] = 'empty',
    [4] = 'user',
  },
```

Nodes without subtypes:

- ins (3)
- mark (4)

adjust (5)

```
  adjust = {
    [0] = 'normal',
    [1] = 'pre',
  },
```

boundary (6)

```
  boundary = {
    [0] = 'cancel',
    [1] = 'user',
    [2] = 'protrusion',
    [3] = 'word',
  },
```

disc (7)

```
  disc  = {
    [0] = 'discretionary',
    [1] = 'explicit',
    [2] = 'automatic',
    [3] = 'regular',
    [4] = 'first',
    [5] = 'second',
  },
```

Nodes without subtypes:

- whatsit (8)
- local_par (9)
- dir (10)

math (11)

```
  math = {
    [0] = 'beginmath',
    [1] = 'endmath',
  },
```

glue (12)

```
  glue = {
    [0]   = 'userskip',
    [1]   = 'lineskip',
    [2]   = 'baselineskip',
    [3]   = 'parskip',
    [4]   = 'abovedisplayskip',
    [5]   = 'belowdisplayskip',
    [6]   = 'abovedisplayshortskip',
    [7]   = 'belowdisplayshortskip',
    [8]   = 'leftskip',
    [9]   = 'rightskip',
    [10]  = 'topskip',
    [11]  = 'splittopskip',
    [12]  = 'tabskip',
    [13]  = 'spaceskip',
    [14]  = 'xspaceskip',
    [15]  = 'parfillskip',
    [16]  = 'mathskip',
    [17]  = 'thinmuskip',
    [18]  = 'medmuskip',
    [19]  = 'thickmuskip',
    [98]  = 'conditionalmathskip',
    [99]  = 'muglue',
    [100] = 'leaders',
    [101] = 'cleaders',
    [102] = 'xleaders',
    [103] = 'gleaders',
  },
```

kern (13)

```
  kern = {
    [0] = 'fontkern',
    [1] = 'userkern',
    [2] = 'accentkern',
    [3] = 'italiccorrection',
  },
```

Nodes without subtypes:

- penalty (14)

- unset (15)
- style (16)
- choice (17)

noad (18)

```
noad = {
  [0] = 'ord',
  [1] = 'opdisplaylimits',
  [2] = 'oplimits',
  [3] = 'opnolimits',
  [4] = 'bin',
  [5] = 'rel',
  [6] = 'open',
  [7] = 'close',
  [8] = 'punct',
  [9] = 'inner',
  [10] = 'under',
  [11] = 'over',
  [12] = 'vcenter',
},
```

radical (19)

```
radical = {
  [0] = 'radical',
  [1] = 'uradical',
  [2] = 'uroot',
  [3] = 'uunderdelimiter',
  [4] = 'uoverdelimiter',
  [5] = 'udelimiterunder',
  [6] = 'udelimiterover',
},
```

Nodes without subtypes:
- fraction (20)

accent (21)

```
accent = {
  [0] = 'bothflexible',
  [1] = 'fixedtop',
  [2] = 'fixedbottom',
  [3] = 'fixedboth',
},
```

fence (22)

```
  fence = {
    [0] = 'unset',
    [1] = 'left',
    [2] = 'middle',
    [3] = 'right',
  },
```

Nodes without subtypes:

- math_char (23)
- sub_box (24)
- sub_mlist (25)
- math_text_char (26)
- delim (27)
- margin_kern (28)

glyph (29)

```
  glyph = {
    [0] = 'character',
    [1] = 'ligature',
    [2] = 'ghost',
    [3] = 'left',
    [4] = 'right',
  },
```

Nodes without subtypes:

- align_record (30)
- pseudo_file (31)
- pseudo_line (32)
- page_insert (33)
- split_insert (34)
- expr_stack (35)
- nested_list (36)
- span (37)
- attribute (38)
- glue_spec (39)
- attribute_list (40)
- temp (41)
- align_stack (42)
- movement_stack (43)
- if_stack (44)
- unhyphenated (45)
- hyphenated (46)
- delta (47)
- passive (48)
- shape (49)

```
  }
  subtypes.whatsit = node.whatsits()
  local out = ''
  if subtypes[typ] and subtypes[typ][n.subtype] then
    out = subtypes[typ][n.subtype]
    if options.verbosity > 1 then
      out = out .. tpl.type_id(n.subtype)
    end
    return out
  else
    return tostring(n.subtype)
  end
  assert(false)
end
```

7.3.2 tpl — Template function

```
function tpl.round(number)
  local mult = 10^(options.decimalplaces or 0)
  return math.floor(number * mult + 0.5) / mult
end

function tpl.length(input)
  input = tonumber(input)
  input = input / tex.sp('1' .. options.unit)
  return string.format('%g%s', tpl.round(input), options.unit)
end

function tpl.fill(number, order, field)
  if order ~= nil and order ~= 0 then
    if field == 'stretch' then
      out = '+'
    else
      out = '-'
    end
    return out .. string.format(
      '%gfi%s', number / 2^16,
      string.rep('l', order - 1)
    )
  else
    return tpl.length(number)
  end
end

tpl.node_colors = {
  hlist = {'red', 'bright'},
  vlist = {'green', 'bright'},
  rule = {'blue', 'bright'},
  ins = {'blue'},
  mark = {'magenta'},
  adjust = {'cyan'},
  boundary = {'red', 'bright'},
  disc = {'green', 'bright'},
  whatsit = {'yellow', 'bright'},
  local_par = {'blue', 'bright'},
  dir = {'magenta', 'bright'},
  math = {'cyan', 'bright'},
  glue = {'magenta', 'bright'},
  kern = {'green', 'bright'},
  penalty = {'yellow', 'bright'},
  unset = {'blue'},
  style = {'magenta'},
  choice = {'cyan'},
  noad = {'red'},
  radical = {'green'},
  fraction = {'yellow'},
  accent = {'blue'},
  fence = {'magenta'},
  math_char = {'cyan'},
  sub_box = {'red', 'bright'},
  sub_mlist = {'green', 'bright'},
  math_text_char = {'yellow', 'bright'},
  delim = {'blue', 'bright'},
  margin_kern = {'magenta', 'bright'},
  glyph = {'cyan', 'bright'},
  align_record = {'red'},
  pseudo_file = {'green'},
  pseudo_line = {'yellow'},
  page_insert = {'blue'},
  split_insert = {'magenta'},
  expr_stack = {'cyan'},
  nested_list = {'red'},
  span = {'green'},
  attribute = {'yellow'},
  glue_spec = {'magenta'},
  attribute_list = {'cyan'},
  temp = {'magenta'},
  align_stack = {'red', 'bright'},
  movement_stack = {'green', 'bright'},
  if_stack = {'yellow', 'bright'},
  unhyphenated = {'magenta', 'bright'},
  hyphenated = {'cyan', 'bright'},
  delta = {'red'},
  passive = {'green'},
  shape = {'yellow'},
}

function tpl.color_code(code)
  return string.char(27) .. '[' .. tostring(code) .. 'm'
end
```

```
% local colors = {
%     -- attributes
%     reset = 0,
%     clear = 0,
%     bright = 1,
%     dim = 2,
%     underscore = 4,
%     blink = 5,
%     reverse = 7,
%     hidden = 8,
%
%     -- foreground
%     black = 30,
%     red = 31,
%     green = 32,
%     yellow = 33,
%     blue = 34,
%     magenta = 35,
%     cyan = 36,
%     white = 37,
%
%     -- background
%     onblack = 40,
%     onred = 41,
%     ongreen = 42,
%     onyellow = 43,
%     onblue = 44,
%     onmagenta = 45,
%     oncyan = 46,
%     onwhite = 47,
% }
```

```
function tpl.color(color, mode, background)
  if options.color ~= 'colored' then
    return ''
  end

  local out = ''
  local code = ''

  if mode == 'bright' then
    out = tpl.color_code(1)
  elseif mode == 'dim' then
    out = tpl.color_code(2)
  end

  if not background then
    if color == 'reset' then code = 0
    elseif color == 'red' then code = 31
    elseif color == 'green' then code = 32
    elseif color == 'yellow' then code = 33
    elseif color == 'blue' then code = 34
    elseif color == 'magenta' then code = 35
    elseif color == 'cyan' then code = 36
    else code = 37 end
  else
    if color == 'black' then code = 40
    elseif color == 'red' then code = 41
    elseif color == 'green' then code = 42
    elseif color == 'yellow' then code = 43
    elseif color == 'blue' then code = 44
    elseif color == 'magenta' then code = 45
    elseif color == 'cyan' then code = 46
    elseif color == 'white' then code = 47
    else code = 40 end
  end
  return out .. tpl.color_code(code)
end

function tpl.key_value(key, value)
  local out = tpl.color('yellow') .. key .. ': '
  if value then
    out = out .. tpl.color('white') .. value .. '; '
  end
  return out .. tpl.color('reset')
end

function tpl.char(input)
  return string.format('%q', unicode.utf8.char(input))
end

function tpl.type(type, id)
  local out = tpl.color(
    tpl.node_colors[type][1],
    tpl.node_colors[type][2]
    )
    .. string.upper(type)
  if options.verbosity > 1 then
    out = out .. tpl.type_id(id)
  end
  return out .. tpl.color('reset')  .. ' '
end

function tpl.callback_variable(variable_name, variable)
  if variable ~= nil and variable ~= '' then
    tpl.print(variable_name .. ': ' .. tostring(variable))
  end
end

function tpl.line(length)
  if length == 'long' then
    return '------------------------------------------'
  else
    return '------------------------'
  end
end

function tpl.callback(callback_name, variables)
  tpl.print('\n\n')
  tpl.print('Callback: ' .. tpl.color('red', '', true) ..
    callback_name .. tpl.color('reset')
  )
  if variables then
    for name, value in pairs(variables) do
      if value ~= nil and value ~= '' then
        tpl.print('  - ' .. name .. ': ' .. tostring(value))
      end
    end
  end
  tpl.print(tpl.line('long'))
end

function tpl.type_id(id)
  return '[' .. tostring(id) .. ']'
end

function tpl.branch(connection_type, connection_state, last)
  local c = connection_type
  local s = connection_state
  local l = last
  if c == 'list' and s == 'stop' and l == false then
    return '  '
  elseif c == 'field' and s == 'stop' and l == false then
    return '  '
  elseif c == 'list' and s == 'continue' and l == false then
    return '│ '
  elseif c == 'field' and s == 'continue' and l == false then
    return '║ '
  elseif c == 'list' and s == 'continue' and l == true then
    return '├─'
  elseif c == 'field' and s == 'continue' and l == true then
    return '╟═'
  elseif c == 'list' and s == 'stop' and l == true then
    return '└─'
  elseif c == 'field' and s == 'stop' and l == true then
    return '╚═'
  end
end

function tpl.branches(level, connection_type)
  local out = ''
  for i = 1, level - 1  do
```

```
    out = out .. tpl.branch('list', tree.state[i]['list'], false)
    out = out .. tpl.branch('field', tree.state[i]['field'], false)
  end
```

Format the last branches

```
  if connection_type == 'list' then
    out = out .. tpl.branch('list', tree.state[level]['list'], true)
  else
    out = out .. tpl.branch('list', tree.state[level]['list'], false)
    out = out .. tpl.branch('field', tree.state[level]['field'], true)
  end
  return out
end

function tpl.print(text)

  if options.channel == 'log' then
    if not log then
      log = io.open(tex.jobname .. '_nodetree.log', 'a')
    end
    log:write(text, '\n')
  else
    print('  ' .. text)
  end
end
```

7.3.3 tree — Build the node tree

```
function tree.format_field(head, field)
  local out = ''

  if not head[field] or head[field] == 0 then
    return ''
  end

  if options.verbosity < 2 and
    -- glyph
    field == 'font' or
    field == 'left' or
    field == 'right' or
    field == 'uchyph' or
    -- hlist
    field == 'dir' or
    field == 'glue_order' or
    field == 'glue_sign' or
    field == 'glue_set' or
    -- glue
    field == 'stretch_order' then
    return ''
  elseif options.verbosity < 3 and
    field == 'prev' or
    field == 'next' or
    field == 'id'
  then
    return ''
  end

  if field == 'prev' or field == 'next' then
    out = nodex.node_id(head[field])
  elseif field == 'subtype' then
    out = nodex.subtype(head)
  elseif
    field == 'width' or
    field == 'height' or
    field == 'depth' or
    field == 'kern' or
    field == 'shift' then
    out = tpl.length(head[field])
  elseif field == 'char' then
    out = tpl.char(head[field])
  elseif field == 'glue_set' then
    out = tpl.round(head[field])
  elseif field == 'stretch' or field == 'shrink' then
    out = tpl.fill(head[field], head[field .. '_order'], field)
  else
    out = tostring(head[field])
  end

  return tpl.key_value(field, out)
end
```

`level` is a integer beginning with 1. The variable `connection_type` is a string, which can be either `list` or `field`. The variable `connection_state` is a string, which can be either `continue` or `stop`.

```
function tree.set_state(level, connection_type, connection_state)
  if not tree.state[level] then
    tree.state[level] = {}
  end
  tree.state[level][connection_type] = connection_state
end

function tree.analyze_fields(fields, level)
  local max = 0
  local connection_state = ''
  for _ in pairs(fields) do
    max = max + 1
  end
  local count = 0
  for field_name, recursion_node in pairs(fields) do
```

```
    count = count + 1
    if count == max then
      connection_state = 'stop'
    else
      connection_state = 'continue'
    end
    tree.set_state(level, 'field', connection_state)
    tpl.print(tpl.branches(level, 'field') .. tpl.key_value(field_name))
    tree.analyze_list(recursion_node, level + 1)
  end
end

function tree.analyze_node(head, level)
  local connection_state
  local out = ''
  if head.next then
    connection_state = 'continue'
  else
    connection_state = 'stop'
  end
  tree.set_state(level, 'list', connection_state)
  out = tpl.branches(level, 'list')
    .. tpl.type(node.type(head.id), head.id)
  if options.verbosity > 1 then
    out = out .. tpl.key_value('no', nodex.node_id(head))
  end

  local fields = {}
  for field_id, field_name in pairs(node.fields(head.id, head.sub-
type)) do
    if field_name ~= 'next' and
      field_name ~= 'prev' and
      node.is_node(head[field_name]) then
      fields[field_name] = head[field_name]
    else
      out = out .. tree.format_field(head, field_name)
    end
  end

  tpl.print(out)
  tree.analyze_fields(fields, level)
end

function tree.analyze_list(head, level)
  while head do
    tree.analyze_node(head, level)
    head = head.next
  end
end
```

```
function tree.analyze_callback(head)
  tree.analyze_list(head, 1)
  tpl.print(tpl.line('short') .. '\n')
end
```

7.3.4 callbacks — Callback wrapper

```
function callbacks.contribute_filter(extrainfo)
  tpl.callback('contribute_filter', {extrainfo = extrainfo})
  return true
end

function callbacks.buildpage_filter(extrainfo)
  tpl.callback('buildpage_filter', {extrainfo = extrainfo})
  return true
end

function callbacks.pre_linebreak_filter(head, groupcode)
  tpl.callback('pre_linebreak_filter', {groupcode = groupcode})
  tree.analyze_callback(head)
  return true
end

function callbacks.linebreak_filter(head, is_display)
  tpl.callback('linebreak_filter', {is_display = is_display})
  tree.analyze_callback(head)
  return true
end
```

TODO: Fix return values, page output

```
function callbacks.append_to_vlist_filter(head, locationcode, pre-
vdepth, mirrored)
  local variables = {
    locationcode = locationcode,
    prevdepth = prevdepth,
    mirrored = mirrored,
  }
  tpl.callback('append_to_vlist_filter', variables)
  tree.analyze_callback(head)
  return true
end

function callbacks.post_linebreak_filter(head, groupcode)
  tpl.callback('post_linebreak_filter', {groupcode = groupcode})
  tree.analyze_callback(head)
  return true
end

function callbacks.hpack_filter(head, groupcode, size, packtype, di-
rection, attributelist)
```

```
  local variables = {
    groupcode = groupcode,
    size = size,
    packtype = packtype,
    direction = direction,
    attributelist = attributelist,
  }
  tpl.callback('hpack_filter', variables)
  tree.analyze_callback(head)
  return true
end

function callbacks.vpack_filter(head, groupcode, size, packtype, maxdepth, di-
rection, attributelist)
  local variables = {
    groupcode = groupcode,
    size = size,
    packtype = packtype,
    maxdepth = tpl.length(maxdepth),
    direction = direction,
    attributelist = attributelist,
  }
  tpl.callback('vpack_filter', variables)
  tree.analyze_callback(head)
  return true
end

function callbacks.hpack_quality(incident, detail, head, first, last)
  local variables = {
    incident = incident,
    detail = detail,
    first = first,
    last = last,
  }
  tpl.callback('hpack_quality', variables)
  tree.analyze_callback(head)
end

function callbacks.vpack_quality(incident, detail, head, first, last)
  local variables = {
    incident = incident,
    detail = detail,
    first = first,
    last = last,
  }
  tpl.callback('vpack_quality', variables)
  tree.analyze_callback(head)
end

function callbacks.process_rule(head, width, height)
```

```
  local variables = {
    width = width,
    height = height,
  }
  tpl.callback('process_rule', variables)
  tree.analyze_callback(head)
  return true
end

function callbacks.pre_output_filter(head, groupcode, size, pack-
type, maxdepth, direction)
  local variables = {
    groupcode = groupcode,
    size = size,
    packtype = packtype,
    maxdepth = maxdepth,
    direction = direction,
  }
  tpl.callback('pre_output_filter', variables)
  tree.analyze_callback(head)
  return true
end

function callbacks.hyphenate(head, tail)
  tpl.callback('hyphenate')
  tpl.print('head:')
  tree.analyze_callback(head)
  tpl.print('tail:')
  tree.analyze_callback(tail)
end

function callbacks.ligaturing(head, tail)
  tpl.callback('ligaturing')
  tpl.print('head:')
  tree.analyze_callback(head)
  tpl.print('tail:')
  tree.analyze_callback(tail)
end

function callbacks.kerning(head, tail)
  tpl.callback('kerning')
  tpl.print('head:')
  tree.analyze_callback(head)
  tpl.print('tail:')
  tree.analyze_callback(tail)
end

function callbacks.insert_local_par(local_par, location)
  tpl.callback('insert_local_par', {location = location})
  tree.analyze_callback(local_par)
  return true
end

function callbacks.mlist_to_hlist(head, display_type, need_penalties)
  local variables = {
    display_type = display_type,
    need_penalties = need_penalties,
  }
  tpl.callback('mlist_to_hlist', variables)
  tree.analyze_callback(head)
  return node.mlist_to_hlist(head, display_type, need_penalties)
end
```

7.3.5 base — Exported base functions

```
function base.normalize_options()
  options.verbosity = tonumber(options.verbosity)
  options.decimalplaces = tonumber(options.decimalplaces)
end

function base.set_default_options()
  local defaults = {
    verbosity = 1,
    callback = 'postlinebreak',
    engine = 'luatex',
    color = 'colored',
    decimalplaces = 2,
    unit = 'pt',
    channel = 'term',
  }
  if not options then
    options = {}
  end
  for key, value in pairs(defaults) do
    if not options[key] then
      options[key] = value
    end
  end
  base.normalize_options()
end

function base.set_option(key, value)
  if not options then
    options = {}
  end
  options[key] = value
  base.normalize_options()
end
```

```
function base.get_option(key)
  if not options then
    options = {}
  end
  if options[key] then
    return options[key]
  end
end
```

```
function base.get_callback_name(alias)
  if alias == 'contribute' or alias == 'contributefilter' then
    return 'contribute_filter'

  elseif alias == 'buildpage' or alias == 'buildpagefilter' then
    return 'buildpage_filter'

  elseif alias == 'preline' or alias == 'prelinebreakfilter' then
    return 'pre_linebreak_filter'

  elseif alias == 'line' or alias == 'linebreakfilter' then
    return 'linebreak_filter'

  elseif alias == 'append' or alias == 'appendtovlistfilter' then
    return 'append_to_vlist_filter'

  elseif alias == 'postline' or alias == 'postlinebreakfilter' then
    return 'post_linebreak_filter'

  elseif alias == 'hpack' or alias == 'hpackfilter' then
    return 'hpack_filter'

  elseif alias == 'vpack' or alias == 'vpackfilter' then
    return 'vpack_filter'
```

TODO: Fix: Unable to register callback

```
  elseif alias == 'hpackq' or alias == 'hpackquality' then
    return 'hpack_quality'
```

TODO: Fix: Unable to register callback

```
  elseif alias == 'vpackq' or alias == 'vpackquality' then
    return 'vpack_quality'

  elseif alias == 'process' or alias == 'processrule' then
    return 'process_rule'

  elseif alias == 'preout' or alias == 'preoutputfilter' then
    return 'pre_output_filter'

  elseif alias == 'hyph' or alias == 'hyphenate' then
```

```
    return 'hyphenate'

  elseif alias == 'liga' or alias == 'ligaturing' then
    return 'ligaturing'

  elseif alias == 'kern' or alias == 'kerning' then
   return 'kerning'

  elseif alias == 'insert' or alias == 'insertlocalpar' then
    return 'insert_local_par'

  elseif alias == 'mhlist' or alias == 'mlisttohlist' then
    return 'mlist_to_hlist'

  else
    return 'post_linebreak_filter'
  end
end

function base.register(cb)
  if options.engine == 'lualatex' then
    luatexbase.add_to_callback(cb, callbacks[cb], 'nodetree')
  else
    id, error = callback.register(cb, callbacks[cb])
  end
end

function base.register_callbacks()
  for alias in string.gmatch(options.callback, '([^,]+)') do
    base.register(base.get_callback_name(alias))
  end
end

function base.unregister(cb)
  if options.engine == 'lualatex' then
    luatexbase.remove_from_callback(cb, 'nodetree')
  else
    id, error = callback.register(cb, nil)
  end
end

function base.unregister_callbacks()
  for alias in string.gmatch(options.callback, '([^,]+)') do
    base.unregister(base.get_callback_name(alias))
  end
end

function base.execute()
  local c = base.get_callback()
  if options.engine == 'lualatex' then
    luatexbase.add_to_callback(c, callbacks.post_linebreak_filter, 'nodetree')
  else
    id, error = callback.register(c, callbacks.post_linebreak_fil-
ter)
  end
end

function base.analyze(head)
  tpl.print('\n')
  tree.analyze_list(head, 1)
end

return base
```

Change History

v0.1

General: Converted to DTX file . 10

v1.0

General: Inital release 10

v1.1

General: Fix the registration of same callbacks 10

v1.2

General: Fix difference between README.md in the upload and that from nodetree.dtx . . 10

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.