

# The nodetree package

Josef Friedrich

[josef@friedrich.rocks](mailto:josef@friedrich.rocks)

[github.com/Josef-Friedrich/nodetree](https://github.com/Josef-Friedrich/nodetree)

Package from 2016/07/13

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

<b>1</b>	<b>Abstract</b>	<b>3</b>
<b>2</b>	<b>Usage</b>	<b>3</b>
2.1	Debug nodes inside Lua code . . . . .	3
<b>3</b>	<b>Macros</b>	<b>4</b>
3.1	<code>\nodetreeregister</code> . . . . .	4
3.2	<code>\nodetreeunregister</code> . . . . .	4
3.3	<code>\nodetreeoption</code> . . . . .	4
3.4	<code>\nodetreeset</code> . . . . .	4
<b>4</b>	<b>Options</b>	<b>5</b>
4.1	Option <code>callback</code> . . . . .	5
4.2	Option <code>verbosity</code> . . . . .	5
4.3	Option <code>color</code> . . . . .	6
4.4	Option <code>unit</code> . . . . .	6
4.5	Option <code>decimalplaces</code> . . . . .	6
<b>5</b>	<b>Visual tree structure</b>	<b>7</b>
5.1	Two different connections . . . . .	7
5.2	Unicode characters to show the tree view . . . . .	7
<b>6</b>	<b>Examples</b>	<b>8</b>
6.1	The node list of the package name . . . . .	8
6.2	The node list of a mathematical formula . . . . .	9
6.3	The node list of the word <i>Office</i> . . . . .	10
<b>7</b>	<b>Implementation</b>	<b>11</b>
7.1	The file <code>nodetree.tex</code> . . . . .	11
7.2	The file <code>nodetree.sty</code> . . . . .	11
7.3	The file <code>nodetree.lua</code> . . . . .	12
7.3.1	<code>nodex</code> — Extend the node library . . . . .	13
7.3.2	<code>tpl</code> — Template function . . . . .	18
7.3.3	<code>tree</code> — Build the node tree . . . . .	23
7.3.4	<code>callbacks</code> — Callback wrapper . . . . .	26
7.3.5	<code>base</code> — Exported base functions . . . . .	29

## 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 Lua<sub>La</sub>T<sub>E</sub>X.

```

\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 `\nodetreereset`

`\nodetreereset` `\nodetreereset{<kv-options>}`: This macro can only be used in Lua<sub>La</sub>T<sub>E</sub>X. `{<kv-options>}` are key value pairs.

```

\nodetreereset{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 multiple 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 distinguishes 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.



**field** connections are visualized by *Double* characters.



## 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;
            - FENCE 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: "0"; 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

```
26 \directlua{
27   nodetree = require('nodetree')
28   nodetree.set_option('engine', 'luatex')
29   nodetree.set_default_options()
30 }
```

`\nodetreeoption`

```
31 \def\nodetreeoption[#1]#2{
32   \directlua{
33     nodetree.set_option('#1', '#2')
34   }
35 }
```

`\nodetreeregister`

```
36 \def\nodetreeregister#1{
37   \directlua{
38     nodetree.set_option('callback', '#1')
39     nodetree.register_callbacks()
40   }
41 }
```

`\nodetreeunregister`

```
42 \def\nodetreeunregister#1{
43   \directlua{
44     nodetree.set_option('callback', '#1')
45     nodetree.unregister_callbacks()
46   }
47 }
```

### 7.2 The file nodetree.sty

```
26 \input{nodetree}
27 \directlua{
28   nodetree.set_option('engine', 'lualatex')
29 }

30 \RequirePackage{kvoptions}

31 \SetupKeyvalOptions{
32   family=NT,
33   prefix=NT@
34 }
```

```

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

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

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

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

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

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

47 \ProcessKeyvalOptions*
48 \directlua{
49   nodetree.set_default_options()
50   nodetree.register_callbacks()
51 }

```

\nodetreeset

```

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

```

### 7.3 The file nodetree.lua

```

1 local nodex = {}

2 local tpl = {}

3 local tree = {}

```

Nodes in LuaTeX are connected. The nodetree view distinguishes 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
```

```
4 tree.state = {}

5 local callbacks = {}

6 local base = {}

7 local options = {}
```

### 7.3.1 nodex — Extend the node library

Get the node id form, e. g.:

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

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

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

#### hlist (0)

```
14   hlist = {
15     [0] = 'unknown',
16     [1] = 'line',
17     [2] = 'box',
18     [3] = 'indent',
19     [4] = 'alignment',
20     [5] = 'cell',
21     [6] = 'equation',
22     [7] = 'equationnumber',
23   },
```

#### vlist (1)

```
24   vlist = {
25     [0] = 'unknown',
26     [4] = 'alignment',
27     [5] = 'cell',
28   },
```

## **rule (2)**

```
29     rule = {
30         [0] = 'unknown',
31         [1] = 'box',
32         [2] = 'image',
33         [3] = 'empty',
34         [4] = 'user',
35     },
```

Nodes without subtypes:

- ins (3)
- mark (4)

## **adjust (5)**

```
36     adjust = {
37         [0] = 'normal',
38         [1] = 'pre',
39     },
```

## **boundary (6)**

```
40     boundary = {
41         [0] = 'cancel',
42         [1] = 'user',
43         [2] = 'protrusion',
44         [3] = 'word',
45     },
```

## **disc (7)**

```
46     disc = {
47         [0] = 'discretionary',
48         [1] = 'explicit',
49         [2] = 'automatic',
50         [3] = 'regular',
51         [4] = 'first',
52         [5] = 'second',
53     },
```

Nodes without subtypes:

- whatsit (8)
- local\_par (9)
- dir (10)

### **math (11)**

```
54     math = {
55         [0] = 'beginmath',
56         [1] = 'endmath',
57     },
```

### **glue (12)**

```
58     glue = {
59         [0] = 'userskip',
60         [1] = 'lineskip',
61         [2] = 'baselineskip',
62         [3] = 'parskip',
63         [4] = 'abovedisplayskip',
64         [5] = 'belowdisplayskip',
65         [6] = 'abovedisplayshortskip',
66         [7] = 'belowdisplayshortskip',
67         [8] = 'leftskip',
68         [9] = 'rightskip',
69         [10] = 'topskip',
70         [11] = 'splittopskip',
71         [12] = 'tabskip',
72         [13] = 'spaceskip',
73         [14] = 'xspaceskip',
74         [15] = 'parfillskip',
75         [16] = 'mathskip',
76         [17] = 'thinmuskip',
77         [18] = 'medmuskip',
78         [19] = 'thickmuskip',
79         [98] = 'conditionalmathskip',
80         [99] = 'muglue',
81         [100] = 'leaders',
82         [101] = 'cleaders',
83         [102] = 'xleaders',
84         [103] = 'gleaders',
85     },
```

### **kern (13)**

```
86     kern = {
87         [0] = 'fontkern',
88         [1] = 'userkern',
89         [2] = 'accentkern',
90         [3] = 'italiccorrection',
91     },
```

Nodes without subtypes:

- **penalty (14)**

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

#### **noad (18)**

```

92     noad = {
93         [0] = 'ord',
94         [1] = 'opdisplaylimits',
95         [2] = 'oplimits',
96         [3] = 'opnolimits',
97         [4] = 'bin',
98         [5] = 'rel',
99         [6] = 'open',
100        [7] = 'close',
101        [8] = 'punct',
102        [9] = 'inner',
103        [10] = 'under',
104        [11] = 'over',
105        [12] = 'vcenter',
106    },

```

#### **radical (19)**

```

107    radical = {
108        [0] = 'radical',
109        [1] = 'uradical',
110        [2] = 'uroot',
111        [3] = 'uunderdelimater',
112        [4] = 'uoverdelimater',
113        [5] = 'udelimiterunder',
114        [6] = 'udelimiterover',
115    },

```

Nodes without subtypes:

- fraction (20)

#### **accent (21)**

```

116    accent = {
117        [0] = 'bothflexible',
118        [1] = 'fixedtop',
119        [2] = 'fixedbottom',
120        [3] = 'fixedboth',
121    },

```

#### **fence (22)**

```

122     fence = {
123         [0] = 'unset',
124         [1] = 'left',
125         [2] = 'middle',
126         [3] = 'right',
127     },

```

Nodes without subtypes:

- math\_char (23)
- sub\_box (24)
- sub\_mlist (25)
- math\_text\_char (26)
- delim (27)
- margin\_kern (28)

### **glyph (29)**

```

128     glyph = {
129         [0] = 'character',
130         [1] = 'ligature',
131         [2] = 'ghost',
132         [3] = 'left',
133         [4] = 'right',
134     },

```

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)

```

135 }
136 subtypes.whatsit = node.whatsits()
137 local out = ''
138 if subtypes[typ] and subtypes[typ][n.subtype] then
139     out = subtypes[typ][n.subtype]
140     if options.verbosity > 1 then
141         out = out .. tpl.type_id(n.subtype)
142     end
143     return out
144 else
145     return tostring(n.subtype)
146 end
147 assert(false)
148 end

```

### 7.3.2 tpl — Template function

```

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

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

158 function tpl.fill(number, order, field)
159     if order ~= nil and order ~= 0 then
160         if field == 'stretch' then
161             out = '+'
162         else
163             out = '-'
164         end
165         return out .. string.format(
166             '%gfi%s', number / 2^16,
167             string.rep('l', order - 1)
168         )
169     else
170         return tpl.length(number)
171     end
172 end

173 tpl.node_colors = {
174     hlist = {'red', 'bright'},
175     vlist = {'green', 'bright'},
176     rule = {'blue', 'bright'},
177     ins = {'blue'},
178     mark = {'magenta'},

```

```

179  adjust = {'cyan'},
180  boundary = {'red', 'bright'},
181  disc = {'green', 'bright'},
182  whatsit = {'yellow', 'bright'},
183  local_par = {'blue', 'bright'},
184  dir = {'magenta', 'bright'},
185  math = {'cyan', 'bright'},
186  glue = {'magenta', 'bright'},
187  kern = {'green', 'bright'},
188  penalty = {'yellow', 'bright'},
189  unset = {'blue'},
190  style = {'magenta'},
191  choice = {'cyan'},
192  noad = {'red'},
193  radical = {'green'},
194  fraction = {'yellow'},
195  accent = {'blue'},
196  fence = {'magenta'},
197  math_char = {'cyan'},
198  sub_box = {'red', 'bright'},
199  sub_mlist = {'green', 'bright'},
200  math_text_char = {'yellow', 'bright'},
201  delim = {'blue', 'bright'},
202  margin_kern = {'magenta', 'bright'},
203  glyph = {'cyan', 'bright'},
204  align_record = {'red'},
205  pseudo_file = {'green'},
206  pseudo_line = {'yellow'},
207  page_insert = {'blue'},
208  split_insert = {'magenta'},
209  expr_stack = {'cyan'},
210  nested_list = {'red'},
211  span = {'green'},
212  attribute = {'yellow'},
213  glue_spec = {'magenta'},
214  attribute_list = {'cyan'},
215  temp = {'magenta'},
216  align_stack = {'red', 'bright'},
217  movement_stack = {'green', 'bright'},
218  if_stack = {'yellow', 'bright'},
219  unhyphenated = {'magenta', 'bright'},
220  hyphenated = {'cyan', 'bright'},
221  delta = {'red'},
222  passive = {'green'},
223  shape = {'yellow'},
224 }

225 function tpl.color_code(code)
226   return string.char(27) .. '[' .. tostring(code) .. 'm'
227 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,
% }

```

```

228 function tpl.color(color, mode, background)
229   if options.color ~= 'colored' then
230     return ''
231   end

232   local out = ''
233   local code = ''

234   if mode == 'bright' then
235     out = tpl.color_code(1)
236   elseif mode == 'dim' then
237     out = tpl.color_code(2)
238   end

239   if not background then
240     if color == 'reset' then code = 0
241     elseif color == 'red' then code = 31
242     elseif color == 'green' then code = 32
243     elseif color == 'yellow' then code = 33
244     elseif color == 'blue' then code = 34

```

```

245     elseif color == 'magenta' then code = 35
246     elseif color == 'cyan' then code = 36
247     else code = 37 end
248 else
249     if color == 'black' then code = 40
250     elseif color == 'red' then code = 41
251     elseif color == 'green' then code = 42
252     elseif color == 'yellow' then code = 43
253     elseif color == 'blue' then code = 44
254     elseif color == 'magenta' then code = 45
255     elseif color == 'cyan' then code = 46
256     elseif color == 'white' then code = 47
257     else code = 40 end
258 end
259 return out .. tpl.color_code(code)
260 end

261 function tpl.key_value(key, value)
262     local out = tpl.color('yellow') .. key .. ': '
263     if value then
264         out = out .. tpl.color('white') .. value .. '; '
265     end
266     return out .. tpl.color('reset')
267 end

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

271 function tpl.type(type, id)
272     local out = tpl.color(
273         tpl.node_colors[type][1],
274         tpl.node_colors[type][2]
275     )
276     .. string.upper(type)
277     if options.verbosity > 1 then
278         out = out .. tpl.type_id(id)
279     end
280     return out .. tpl.color('reset') .. ' '
281 end

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

287 function tpl.line(length)
288     if length == 'long' then
289         return '-----'

```

```

290 else
291     return '-----'
292 end
293 end

294 function tpl.callback(callback_name, variables)
295     tpl.print('\n\n')
296     tpl.print('Callback: ' .. tpl.color('red', '', true) ..
297         callback_name .. tpl.color('reset')
298     )
299     if variables then
300         for name, value in pairs(variables) do
301             if value ~= nil and value ~= '' then
302                 tpl.print(' - ' .. name .. ': ' .. tostring(value))
303             end
304         end
305     end
306     tpl.print(tpl.line('long'))
307 end

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

311 function tpl.branch(connection_type, connection_state, last)
312     local c = connection_type
313     local s = connection_state
314     local l = last
315     if c == 'list' and s == 'stop' and l == false then
316         return ' '
317     elseif c == 'field' and s == 'stop' and l == false then
318         return ' '
319     elseif c == 'list' and s == 'continue' and l == false then
320         return '|'
321     elseif c == 'field' and s == 'continue' and l == false then
322         return '||'
323     elseif c == 'list' and s == 'continue' and l == true then
324         return '└─'
325     elseif c == 'field' and s == 'continue' and l == true then
326         return '┌─'
327     elseif c == 'list' and s == 'stop' and l == true then
328         return '└─'
329     elseif c == 'field' and s == 'stop' and l == true then
330         return '┌─'
331     end
332 end

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

```

```

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

```

Format the last branches

```

339 if connection_type == 'list' then
340     out = out .. tpl.branch('list', tree.state[level]['list'], true)
341 else
342     out = out .. tpl.branch('list', tree.state[level]['list'], false)
343     out = out .. tpl.branch('field', tree.state[level]['field'], true)
344 end
345 return out
346 end

```

```

347 function tpl.print(text)
348
349 if options.channel == 'log' then
350     if not log then
351         log = io.open(tex.jobname .. '_nodetree.log', 'a')
352     end
353     log:write(text, '\n')
354 else
355     print(' ' .. text)
356 end
357 end

```

### 7.3.3 tree — Build the node tree

```

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

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

363     if options.verbosity < 2 and
364         -- glyph
365         field == 'font' or
366         field == 'left' or
367         field == 'right' or
368         field == 'uchyph' or
369         -- hlist
370         field == 'dir' or
371         field == 'glue_order' or
372         field == 'glue_sign' or
373         field == 'glue_set' or
374         -- glue
375         field == 'stretch_order' then
376         return ''

```

```

377 elseif options.verbosity < 3 and
378     field == 'prev' or
379     field == 'next' or
380     field == 'id'
381 then
382     return ''
383 end

384 if field == 'prev' or field == 'next' then
385     out = nodex.node_id(head[field])
386 elseif field == 'subtype' then
387     out = nodex.subtype(head)
388 elseif
389     field == 'width' or
390     field == 'height' or
391     field == 'depth' or
392     field == 'kern' or
393     field == 'shift' then
394     out = tpl.length(head[field])
395 elseif field == 'char' then
396     out = tpl.char(head[field])
397 elseif field == 'glue_set' then
398     out = tpl.round(head[field])
399 elseif field == 'stretch' or field == 'shrink' then
400     out = tpl.fill(head[field], head[field .. '_order'], field)
401 else
402     out = tostring(head[field])
403 end

404 return tpl.key_value(field, out)
405 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.
406 function tree.set_state(level, connection_type, connection_state)
407     if not tree.state[level] then
408         tree.state[level] = {}
409     end
410     tree.state[level][connection_type] = connection_state
411 end

412 function tree.analyze_fields(fields, level)
413     local max = 0
414     local connection_state = ''
415     for _ in pairs(fields) do
416         max = max + 1
417     end
418     local count = 0
419     for field_name, recursion_node in pairs(fields) do

```

```

420     count = count + 1
421     if count == max then
422         connection_state = 'stop'
423     else
424         connection_state = 'continue'
425     end
426     tree.set_state(level, 'field', connection_state)
427     tpl.print(tpl.branches(level, 'field') .. tpl.key_value(field_name))
428     tree.analyze_list(recursion_node, level + 1)
429 end
430 end

431 function tree.analyze_node(head, level)
432     local connection_state
433     local out = ''
434     if head.next then
435         connection_state = 'continue'
436     else
437         connection_state = 'stop'
438     end
439     tree.set_state(level, 'list', connection_state)
440     out = tpl.branches(level, 'list')
441     .. tpl.type(node.type(head.id), head.id)
442     if options.verbosity > 1 then
443         out = out .. tpl.key_value('no', nodex.node_id(head))
444     end

445     local fields = {}
446     for field_id, field_name in pairs(node.fields(head.id, head.sub-
type)) do
447         if field_name ~= 'next' and
448             field_name ~= 'prev' and
449             node.is_node(head[field_name]) then
450             fields[field_name] = head[field_name]
451         else
452             out = out .. tree.format_field(head, field_name)
453         end
454     end

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

458 function tree.analyze_list(head, level)
459     while head do
460         tree.analyze_node(head, level)
461         head = head.next
462     end
463 end

```

```

464 function tree.analyze_callback(head)
465     tree.analyze_list(head, 1)
466     tpl.print(tpl.line('short') .. '\n')
467 end

```

#### 7.3.4 callbacks — Callback wrapper

```

468 function callbacks.contribute_filter(extrainfo)
469     tpl.callback('contribute_filter', {extrainfo = extrainfo})
470     return true
471 end

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

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

481 function callbacks.linebreak_filter(head, is_display)
482     tpl.callback('linebreak_filter', {is_display = is_display})
483     tree.analyze_callback(head)
484     return true
485 end

    TODO: Fix return values, page output
486 function callbacks.append_to_vlist_filter(head, locationcode, pre-
    vdepth, mirrored)
487     local variables = {
488         locationcode = locationcode,
489         prevdepth = prevdepth,
490         mirrored = mirrored,
491     }
492     tpl.callback('append_to_vlist_filter', variables)
493     tree.analyze_callback(head)
494     return true
495 end

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

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

```

```

502 local variables = {
503     groupcode = groupcode,
504     size = size,
505     packtype = packtype,
506     direction = direction,
507     attributelist = attributelist,
508 }
509 tpl.callback('hpack_filter', variables)
510 tree.analyze_callback(head)
511 return true
512 end

513 function callbacks.vpack_filter(head, groupcode, size, packtype, maxdepth, di-
    rection, attributelist)
514     local variables = {
515         groupcode = groupcode,
516         size = size,
517         packtype = packtype,
518         maxdepth = tpl.length(maxdepth),
519         direction = direction,
520         attributelist = attributelist,
521     }
522     tpl.callback('vpack_filter', variables)
523     tree.analyze_callback(head)
524     return true
525 end

526 function callbacks.hpack_quality(incident, detail, head, first, last)
527     local variables = {
528         incident = incident,
529         detail = detail,
530         first = first,
531         last = last,
532     }
533     tpl.callback('hpack_quality', variables)
534     tree.analyze_callback(head)
535 end

536 function callbacks.vpack_quality(incident, detail, head, first, last)
537     local variables = {
538         incident = incident,
539         detail = detail,
540         first = first,
541         last = last,
542     }
543     tpl.callback('vpack_quality', variables)
544     tree.analyze_callback(head)
545 end

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

```

```

547     local variables = {
548         width = width,
549         height = height,
550     }
551     tpl.callback('process_rule', variables)
552     tree.analyze_callback(head)
553     return true
554 end

555 function callbacks.pre_output_filter(head, groupcode, size, pack-
    type, maxdepth, direction)
556     local variables = {
557         groupcode = groupcode,
558         size = size,
559         packtype = packtype,
560         maxdepth = maxdepth,
561         direction = direction,
562     }
563     tpl.callback('pre_output_filter', variables)
564     tree.analyze_callback(head)
565     return true
566 end

567 function callbacks.hyphenate(head, tail)
568     tpl.callback('hyphenate')
569     tpl.print('head:')
570     tree.analyze_callback(head)
571     tpl.print('tail:')
572     tree.analyze_callback(tail)
573 end

574 function callbacks.ligaturing(head, tail)
575     tpl.callback('ligaturing')
576     tpl.print('head:')
577     tree.analyze_callback(head)
578     tpl.print('tail:')
579     tree.analyze_callback(tail)
580 end

581 function callbacks.kerning(head, tail)
582     tpl.callback('kerning')
583     tpl.print('head:')
584     tree.analyze_callback(head)
585     tpl.print('tail:')
586     tree.analyze_callback(tail)
587 end

588 function callbacks.insert_local_par(local_par, location)
589     tpl.callback('insert_local_par', {location = location})
590     tree.analyze_callback(local_par)

```

```

591 return true
592 end

593 function callbacks.mlist_to_hlist(head, display_type, need_penal-
ties)
594   local variables = {
595     display_type = display_type,
596     need_penalties = need_penalties,
597   }
598   tpl.callback('mlist_to_hlist', variables)
599   tree.analyze_callback(head)
600   return node.mlist_to_hlist(head, display_type, need_penalties)
601 end

```

### 7.3.5 base — Exported base functions

```

602 function base.normalize_options()
603   options.verbosity = tonumber(options.verbosity)
604   options.decimalplaces = tonumber(options.decimalplaces)
605 end

606 function base.set_default_options()
607   local defaults = {
608     verbosity = 1,
609     callback = 'postlinebreak',
610     engine = 'luatex',
611     color = 'colored',
612     decimalplaces = 2,
613     unit = 'pt',
614     channel = 'term',
615   }
616   if not options then
617     options = {}
618   end
619   for key, value in pairs(defaults) do
620     if not options[key] then
621       options[key] = value
622     end
623   end
624   base.normalize_options()
625 end

626 function base.set_option(key, value)
627   if not options then
628     options = {}
629   end
630   options[key] = value
631   base.normalize_options()
632 end

```

```

633 function base.get_option(key)
634   if not options then
635     options = {}
636   end
637   if options[key] then
638     return options[key]
639   end
640 end

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

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

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

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

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

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

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

656   elseif alias == 'vpack' or alias == 'vpackfilter' then
657     return 'vpack_filter'

  TODO: Fix: Unable to register callback
658   elseif alias == 'hpackq' or alias == 'hpackquality' then
659     return 'hpack_quality'

  TODO: Fix: Unable to register callback
660   elseif alias == 'vpackq' or alias == 'vpackquality' then
661     return 'vpack_quality'

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

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

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

```

```

667     return 'hyphenate'

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

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

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

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

676 else
677     return 'post_linebreak_filter'
678 end
679 end

680 function base.register(cb)
681     if options.engine == 'luatex' then
682         luatexbase.add_to_callback(cb, callbacks[cb], 'nodetree')
683     else
684         id, error = callback.register(cb, callbacks[cb])
685     end
686 end

687 function base.register_callbacks()
688     for alias in string.gmatch(options.callback, '([^\,]+)') do
689         base.register(base.get_callback_name(alias))
690     end
691 end

692 function base.unregister(cb)
693     if options.engine == 'luatex' then
694         luatexbase.remove_from_callback(cb, 'nodetree')
695     else
696         id, error = callback.register(cb, nil)
697     end
698 end

699 function base.unregister_callbacks()
700     for alias in string.gmatch(options.callback, '([^\,]+)') do
701         base.unregister(base.get_callback_name(alias))
702     end
703 end

704 function base.execute()
705     local c = base.get_callback()

```

```

706 if options.engine == 'luatex' then
707     luatexbase.add_to_callback(c, callbacks.post_linebreak_filter, 'nodetree')
708 else
709     id, error = callback.register(c, callbacks.post_linebreak_fil-
710     ter)
711 end

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

716 return base

```

## Change History

v0.1		v1.1
General: Converted to DTX file . . .	10	
v1.0		General: Fix the registration of
General: Inital release . . . . .	10	same callbacks . . . . . 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.

D	N	P
\DeclareStringOp- tion .... 35, 37, 39, 41, 43, 45	\n ... 295, 353, 466, 713	\ProcessKeyvalOp- tions ..... 47
\define@key ... 36, 38, 40, 42, 44, 46	\nodetreeoption . ..... <u>31</u> , 36, 38, 40, 42, 44, 46	<b>R</b>
	\nodetreeregister <u>36</u>	\RequirePackage .. 30
<b>I</b>	\nodetreeset ..... <u>52</u>	<b>S</b>
\input ..... 26	\nodetreeunregis- ter ..... <u>42</u>	\setkeys ..... 52
		\SetupKeyvalOp- tions ..... 31