

forest-ext

Clea F. Rees*

2026/01/17

Abstract

forest-ext consists of various libraries for Sašo Živanović’s package `forest` (2017). The aim of the libraries is to provide bug fixes or extensions currently unavailable in `forest` itself. I hope that this package — or at least many of its constituents — will eventually be rendered unnecessary by an updated `forest` and disappear.

Contents

1	Basic usage	3
2	Tagging	3
2.0.1	Customisation	5
2.0.2	Custom plugs	6
2.0.3	Complete control	7
2.1	Workflow	8
2.2	Example	10
3	Multiple parents	12
3.1	Creating multiple parents	12
3.2	Connecting multiple parents	15
4	Linguistics extensions	17
5	Utilities	18
5.1	Alignment	18
5.2	Outer labels	19
5.3	‘Tagging’ keylists	20
6	Implementation	21
	ext.ling	22
	ext.multi	24
	ext.tagging	31

*Bug tracker: codeberg.org/cfr/prooftrees/issues | Code: codeberg.org/cfr/prooftrees | Mirror: github.com/cfr42/prooftrees

ext.utils	42
7 Toks etc.	42
8 ‘Tagging keylists’	43
9 Styles	46

1 Basic usage

This package currently provides the following libraries:

- `ext.ling` (*lib.*) Experimental elementary support for trees involving multi-dominance, based on `ext.multi`. See section 4.
- `ext.multi` (*lib.*) Experimental elementary support for nodes with multiple parents. See section 3.
- `ext.tagging` (*lib.*) Experimental automatic tagging of forest trees. See section 2.

Although this relies only on documented public interfaces provided by `forest` — no `forest` internals are patched or redefined — the library does change the same PGF internals as the tagging support in `latex-lab-tikz-testphase` (L^AT_EX Project 2025b).

- `ext.utils` (*lib.*) Bits 'n bobs. See section 5.

For debugging, the following alternative libraries are provided:

- `ext.ling-debug` (*lib.*) `ext.ling` plus debugging. See section 4.
- `ext.multi-debug` (*lib.*) `ext.multi` plus debugging. See section 3.
- `ext.tagging-debug` (*lib.*) `ext.tagging` plus debugging. See section 2.
- `ext.utils-debug` (*lib.*) `ext.utils` plus debugging. See section 5.

Load the libraries in the same way as standard libraries:

```
\usepackage[<comma-separated-list of libraries>]{forest}
```

or

```
\usepackage{forest}
\useforestlibrary{<comma-separated-list of libraries>}
```

For example, the following line would load `forest-lib-ext.multi` and apply any defaults globally.

```
\usepackage[ext.multi]{forest}
```

The following lines would load the same library, but without applying any defaults.

```
\usepackage{forest}
\useforestlibrary{ext.multi}
```

Any default settings can then be applied locally using `\forestapplylibrarydefaults{<list of libraries>}`, if desired.

2 Tagging¹

Note that this library requires `ext.utils`, described in section 5.

- `ext.tagging` (*lib.*) Experimental semi-automatic tagging of `forest` trees.
- `ext.tagging-debug` (*lib.*) `ext.tagging` plus debugging.

`forest-lib-ext.tagging` (and `forest-lib-ext.tagging-debug`) are based on the ‘first-aid’ in `latex-lab-tikz-testphase` by Ulrike Fischer (L^AT_EX Project 2025b). Those patches do not work with `forest` because

¹For an introduction to support for tagged PDF in L^AT_EX 2_ε, see Fischer (2025). For gorier details see, for example, International Organization for Standardization (2025) and PDF Association (2024a,b) and related publications.

a `forest` tree includes many `tikzpicture` environments, some of which may never be typeset and all of which are used only indirectly *via* low-level T_EX boxes. Moreover, the `latex-lab` code depends on PGF’s ‘remember picture’ feature, which is not compatible with `forest` with or without tagging.

In addition to making it possible to tag `forest` environments in tagged documents, the library produces an alternative text describing the tree semi-automatically. This is important because trees are unlike some other images, where relatively short summaries provide a reasonable alternative to the picture. To provide high quality access to the information contained in a typical tree, it is necessary to describe it in detail. Both the content of the nodes and their structural relationships must be described, together with any labels and annotations.

The current implementation does not do all of the work: it does not include information from regular labels or the content of annotations added using regular TikZ or PGF techniques. However, it does describe the main tree’s structure, together with the content of its nodes and edge labels, though you may need to override the generated content for content which includes special characters, in a quite broad sense of ‘special’.

The support for tagging adds the following `forest` *stages* which are executed in order, sandwiched between `compute xy stage` and `before drawing tree`.

If you redefine (or load code which redefines) the default implementation of stages, you must include or replace the additions from this library. For an example of how to do this, see `prooftrees` (Rees 2026), which includes, redefines, supplements or replaces these additions.

`before tagging nodes` Empty by default. Analogous to `before typesetting nodes`, `before packing` etc.

(*keylist*)
`tag nodes (tag. keylist)` Executes code to assign tagging code to each node in the tree.

Note this is a *tagging* keylist. See section 5.3.

`before collating tags` Empty by default. Analogous to `before typesetting nodes`, `before packing` etc.

(*keylist*)
`collate tags (tag. keylist)` Walks the tree to collate the tags into a single alternative text for the tree.

Note this is a *tagging* keylist. See section 5.3.

`before tagging tree (keylist)` Empty by default. Analogous to `before typesetting nodes`, `before packing` etc.

`tag tree stage (stage)` Calculates an approximate bounding box for the tree and inserts the collated tagging data into the document’s tagging structure using `tagpdf`.

The code inserts a tagged structure analogous to (and heavily derived from) the `alt` plug provided by `latex-lab-tikz-testphase`. However, unlike the `latex-lab` plug, the library generates the `alt` text automatically by default. The result can be configured using a small number of keys. The keys’ scope is the entire tree, except that the scope of `alt text` is *the current node*.

`alt text (auto. toks) = <tokens>`

Override the automatic generation of alternative text for the current node.

Internally, the code uses the further key `node@ttoks`. In essence, if `alt text` is empty, `node@ttoks` is constructed from the node’s `content`, `edge label` and any applicable structural descriptors, as specified by `is root`, `is branch` and so on. If `alt text` is not empty, it is used as-is. The reason for this indirect assignment — first constructing `node@ttoks` and only then assigning it to `alt text` — is that the value of `node@ttoks` is constructed incrementally (i.e. partially by `delayed` keys) and keeping `alt text` as-is makes it easy to test during every cycle.

`node@ttoks` is intended for purely internal use and should **NOT** be used outside the library code. `alt text` is the public face of this key.

Note that tagging content is always attached to nodes². Labels, edge labels and structures

²I’m not altogether happy with this implementation, so this may change, but I want to keep things relatively simple for now.

are not (currently?) tagged independently. So, if you specify `alt text`, you replace not only the `content` of the node in the corresponding tag, but the content of any `edge label` and any relevant structural information. So if you want, say, a branch number prepended or an indication that the node is a ‘`child`’ or ‘`leaf`’, say, or that the tree forks from this node, you must include that information into the `<tokens>` when specifying `alt text`.

`is root (auto. toks reg.) = <tokens>`

Specify text to insert when describing the root. Default is `root`.

`is child (auto. toks reg.) = <tokens>`

Specify text to insert when describing a child. Default is `child`.

`is leaf (auto. toks reg.) = <tokens>`

Specify text to insert when describing a leaf node. Default is `end branch`.

`is edge label (auto. toks reg.) = <tokens>`

Specify text to insert when describing an edge label. Default is `edge label`.

`has branches (auto. toks reg.) = <tokens>`

Specify text to insert when describing a parent’s branches. Default is `branches`. A number is inserted before to indicate the number of branches.

`is branch (auto. toks reg.) = <tokens>`

Specify text to insert when describing node’s (and, hence, this subtree’s) position in the tree. Default is `branch`. A number is appended to indicate which branch.

2.0.1 Customisation

Most users will not need the options explained in this section.

`tagging (bool. reg.)`

`tagging` may be used to make code conditional on the activation status of tagging. For this reason, it has a public name. However, it should **NOT** be changed.

More generally, you should not suspend, resume, enable or disable tagging inside a `forest` environment unless you understand what you are doing with respect to *both* the tagging code *and* `forest`³.

`tag nodes uses (choice) = none|alt text`

Configures the keylist `tag nodes`. `alt text` installs the default auto-generation code which constructs a value if `alt text` is unspecified for a (non-phantom) node.

The order in which nodes are tagged may be set using `tag nodes processing order`. The default is `unique=tree`.

`collate tags uses (choice) = none|alt text`

Configures the keylist `collate tags`. `alt text` installs code to collate the values of the autowrapped `toks` option `alt text`.

The order of collation may be set using `collate tags processing order`. The default is `unique=tree depth first`.

`tag tree uses (choice) = none|alt text`

Configures the style `tag tree`. `alt text` installs the default keys used to calculate approximate dimensions for the bounding box and to pass the collated tags to the *plug* responsible for tagging the tree.

This style is used by the default implementation of `tag tree stage`:

³Possibly nobody currently meets both of these requirements.

```
tag tree stage/.style={for root'=tag tree},
```

2.0.2 Custom plugs

By default, everything is `noop`. If the user does nothing and tagging is active, the `alt` plug is used. If this is not desired, it is sufficient to use `,` which will make everything (remain) `noop` or `,` which will allow the `latex-lab` patches to mix explosively with your `forest` trees. This is not recommended unless you plan to prevent such encounters yourself. In the worst cases, the combination will result in fatal compilation errors. In the best cases, the document will compile, but tagging will almost certainly be broken.

However, it is possible to strike a middle course and use the infrastructure provided by this library as the basis for custom tagging. Some approaches were explained in section 2.0.1. If those are not sufficient, you may define custom plugs. This section explains the minimal requirements for such plugs to be used by this library i.e. without using `custom tagging`.

Requirements Let `Percy` be the name of your custom plug. Then `ext.tagging` requires:

1. a plug named `Percy` for socket `tagsupport/forest/setup`;
2. a plug named `Percy` for socket `tagsupport/forest/tag`.

If both conditions are satisfied, writing

```
\forestset{%
  plug=Percy,
}
```

will not result in an immediate error.

In order to do something useful, of course, `Percy` must do rather more than this, so let's see what `alt` is used. `tagsupport/forest/setup` is used right at the start of the tree. This happens before any parenthetical argument is processed, before any star is used, before the default preamble and well before any tree-specific preamble⁴. In particular, the default values of *tagging keylists* may still be manipulated at this point, since the socket is used before they are transformed into regular keylist options. The `alt` plug exploits this using the following code:

```
\socket_new_plugin {tagsupport/forest/setup}{alt}
{
  \forestset{
    plug=alt,
    tag nodes uses=alt text,
    collate tags uses=alt text,
    tag tree uses=alt,
  }
}
```

Note that it is good practice to set `plug` here, even if the code is already plug-specific, since the value is used later when calling the `tagsupport/forest/tag` socket. The content of the `alt tagsupport/forest/tag` plug is very similar to the `latex-lab` patch for `.`

So let's assume that `Percy` should use the same code as the `alt` plug for the `tagsupport/forest/tag` socket, but something different for `tagsupport/forest/setup`.

⁴It uses a generic hook to inject code before an internal macro. This ensures it works for both the environment and command forms without adding an additional `TEX` group, but is clearly not ideal.

As noted above, `tag nodes uses`, `collate tags uses` and `tag tree uses` are choice keys. Given the way `pgfkeys` implements such keys, Percy might do something like this:

```
\NewSocketPlug {tagsupport/forest/setup}{percy}
{%
  \forestset{%
    plug=percy,
    tag nodes uses=percy,
    collate tags uses=percy,
    tag tree uses=alt,
  }%
}
\forestset{%
  declare autowrapped toks={percy text}{},
  tag nodes uses/percy/.style={%
    redeclare tagging keylist={tag nodes}{%
      if percy text={}{%
        percy text/.option=content,
        +percy text={Percy: },
      }{%
        percy text+={: },
        percy text+/.option=content,
      },
    },
  },
  collate tags uses/percy/.style={%
    redeclare tagging keylist={collate tags}{%
      collate tag/.option=percy text,
    },
  },
}
\NewSocketPlug {tagsupport/forest/tag}{percy}
{%
  \AssignSocketPlug {tagsupport/forest/tag}{alt}%
  \UseSocket {tagsupport/forest/tag}%
}
```

This would result in each node in the tree contributing both its content and a prefix specified by option `percy text` to the alternative text provided in the tagging structure of the PDF. No structural information is added here i.e. there are no descriptions of branching or of the relationships between nodes⁵.

2.0.3 Complete control

`custom tagging (code key) = true|false`

not custom tagging (code key)

If true, do not tag following trees in the current $\text{T}_\text{E}\text{X}$ group.

This key must be used BEFORE `\begin{forest}` or `\Forest`.

If you do not want to use the library's tagging code, you can easily avoid it by simply not using it. However, you might want to use it for only some trees or you might wish to use the pre-defined stages as a basis for a custom configuration. In such cases, `custom tagging` may be used to tell the library that it should not tag trees in the local $\text{T}_\text{E}\text{X}$ group even if tagging is active. In this

⁵For a more realistic implementation, see section 6 for the code used for the `alt` plug. For a more elaborate example of customisation, see Rees (2026).

case, the user (or another package) is entirely responsible for tagging. The custom tagging code may nonetheless test `tagging` and use the additional stages, if desired. For example, it could redefine the stages which generate and concatenate the tags or it could install alternative plugs into appropriate sockets.

Note that `latex-lab`'s code is still active in this scenario, so you are responsible for dealing with the patches it applies for `tikzpicture` environments. Note also that `custom tagging` is *not* a boolean register or option — it is simply designed to emulate one. It in fact uses the `.code` handler to set an `expl3` boolean variable.

The default `alt` plug is implemented in modular fashion, so it is possible, with care, to take a pick-'n-mix approach.

2.1 Workflow

`ext.tagging` redefines `forest`'s `stages`. If you just wish to use the library to tag ordinary trees, you can ignore the details of this definition. However, should you wish to use the library with a custom definition of `stages`, the details below should enable you to do so. As with `forest`'s own definitions, the various steps may be redefined, replaced, removed or extended as required. The library also follows the `forest` package's convention in providing `before` keylists reserved for user use i.e. all such keylists are empty by default.

Tagging is initialised and finalised by code added to the hooks `env/forest/begin` and `env/forest/end`.

```
stages/.style={
  for root'={
    process keylist register=default preamble,
    process keylist register=preamble
  },
  process keylist=given options,
  process keylist=before typesetting nodes,
  typeset nodes stage,
  process keylist=before packing,
  pack stage,
  process keylist=before computing xy,
  compute xy stage,
  process keylist=before tagging nodes,
  process keylist=tag nodes,
  process keylist=before collating tags,
  process keylist=collate tags,
  process keylist=before tagging tree,
  tag tree stage,
  process keylist=before drawing tree,
  draw tree stage
},
```

This describes the default implementation with `setup plug=alt` and `tag plug=alt`⁶.

1. `default preamble` (see Živanović 2017)
2. `preamble` (see Živanović 2017)

⁶Strictly speaking, the non-trivial claims in items 10 to 15 are almost entirely false as stated. For example, `tag nodes` could construct an entirely new branch and put all the tagging information there, `collate tags` could then collect that information and write it to an external file and `tag tree stage` could embed or attach that file. But that is not very useful to know. The proof of this is simple: if such radical divergence features in your tagging plans, you do not need this package, while, if you do, it shouldn't. QED. It follows that you should skip this footnote.

3. `given options` (see Živanović 2017)
4. `before typesetting nodes` (see Živanović 2017)
5. `typeset nodes stage` (see Živanović 2017)
6. `before packing` (see Živanović 2017)
7. `pack stage` (see Živanović 2017)
8. `before computing xy` (see Živanović 2017)
9. `compute xy stage` (see Živanović 2017)
10. `before tagging nodes` Empty by default. Use in the same way as forest's `before` keylists.
11. `tag nodes` A *tagging keylist* which should, when processed, ensure that each node which requires tagging is correctly tagged in whichever way the installed `tag plug` and associated code requires e.g. for the default `alt` configuration, `alt text`.
12. `before collating tags` Empty by default. Use in the same way as forest's `before` keylists.
13. `collate tags` A *tagging keylist* which should, when processed, result in the collation of all tags for the tree in the form expected by `tag tree stage`.
14. `before tagging tree` Empty by default. Use in the same way as forest's `before` keylists.
15. `tag tree stage` Executes code to actually tag the tree using the data finalised in `collate tags` (possibly modified by `before tagging tree`).
16. `before drawing tree` (see Živanović 2017)
17. `draw tree stage` (see Živanović 2017)

2.2 Example

Here is a complete example⁷:

```
\DocumentMetadata{
  tagging=on,
  lang=en-GB,
  pdfversion=2.0,
  pdfstandard=ua-2,
}
\tagpdfsetup{
  math/mathml/structelem,
}
\documentclass{article}
\usepackage[ext.tagging]{forest}
\ifcsname directlua\endcsname
  \usepackage{unicode-math}
\else
  \usepackage[T1]{fontenc}
\fi
\title{This Test Needs No Title}
\begin{document}
  ABC apple banana pear
  \begin{forest}
    % This example is from Jasper Habicht.
    [VP
      [DP[John]]
      [V', alt text=V prime,
        [V[sent]]
        [DP[Mary]]
        [DP[D[a]] [NP[letter]]]]
      ]
    ]
  \end{forest}
  ABC apple banana pear
}
\end{document}
```

Note the use of `alt text` to avoid problems due to the use of `'` with PDF_TE_X. If the (L^AT_EX Project recommended) engine Lua_LA_TE_X is used, you need not be quite so careful, but you should always check the content of the `alt text` for unpleasant surprises.

If compiled with pdf_LA_TE_X, the above example yields the following structure:

```
<PDF>
<StructTreeRoot>
  <Document xmlns="http://iso.org/pdf2/ssn"
    id="ID.02"
  >
  <text-unit xmlns="https://www.latex-project.org/ns/dflt"
    id="ID.05"
```

⁷Note that the recommended syntax for invoking and using tagging support in L^AT_EX 2_ε changes very frequently. In particular, the recommended options for `\DocumentMetadata` and `\tagpdfsetup`, including whether to use the latter at all, are not at all stable. You should therefore check and use the recommended options at the time your document is written — there is nothing in the code before `\documentclass` which is in any way particular to using the libraries provided by this package. That is, deviations from documented best practice in the use of `\DocumentMetadata` and `\tagpdfsetup` are either due to mistakes on my part or the result of updates following the publication of this document. In either case, you should avoid replicating the deviations in your own code.

```

    rolemaps-to="Part"
  >
<text xmlns="https://www.latex-project.org/ns/dflt"
  id="ID.06"
  xmlns:Layout="http://iso.org/pdf/ssn/Layout"
  Layout:TextAlign="Justify"
  rolemaps-to="P"
  >
  <?MarkedContent page="1" ?>ABC apple banana pear
<Figure xmlns="http://iso.org/pdf2/ssn"
  id="ID.07"
  alt="root VP 2 branches   branch 1 DP   child John end branch   V prime V   child
    ↪ sent end branch   DP   child Mary end branch   DP 2 branches   branch 1 D   child
    ↪ a end branch   branch 2 NP   child letter end branch   "
  xmlns:Layout="http://iso.org/pdf/ssn/Layout"
  Layout:BBox="{ 259.4641, 542.90266, 407.35223, 667.19801 }"
  >
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  <?MarkedContent page="1" ?>
  </Figure>
  <?MarkedContent page="1" ?> ABC apple banana pear
</text>
</text-unit>
</Document>
</StructTreeRoot>
</PDF>

```

A similar result is obtained with Lua \LaTeX , but the output is a bit longer as it includes many empty MarkedContents.

3 Multiple parents

This library provides some basic facilities for formatting trees which are not technically trees in `forest`'s sense. In the (one) strict sense of 'tree', every node but one has exactly one parent, while the one has none.

However, in a different/looser sense of 'tree', every node but one has at least one parent, while the one has none. This library makes it a bit easier to draw such trees with `forest`.

The library began in response to a query from Alan Munn on T_EX SE and initially focused entirely on *multi-dominance* structures in linguistics. Support for those structures is available in the `ext.ling` library, which now uses the more general `ext.multi`.

The styles in section 3.1 support drawing connections from a child to additional parents not currently in the tree, while those in section 3.2 support adding connections to additional extant parents.

Note that

- styles are always specified for the child node;
- the child must have exactly one 'natural' parent i.e. it must be part of the existing tree structure when the style is used.

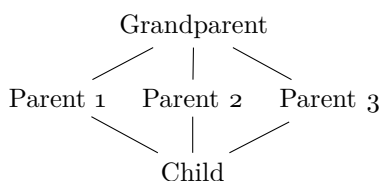
Load `ext.multi` or `ext.multi-debug` as described in section 1.

3.1 Creating multiple parents

Note:

- the child should be created as the *child* of its ultimate grandparent;
- the child's parents will all be children of the child's grandparent.

For example, consider the tree,



This structure can be conveniently created using `multi`, but to translate it into the bracket notation `forest` uses, all of `Child`'s parents should first be omitted and `Child` should instead be specified as the child of `Grandparent`.

```
\begin{forest}
  [Grandparent [Child]]
\end{forest}
```

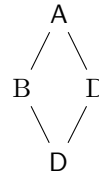
Parents 1, 2 and 3 should be specified as an option to `Child`:

```
\begin{forest}
  [Grandparent [Child, multi={Parent 1,Parent 2,Parent 3}]]
\end{forest}
```

`multi (style) = {\langle content of parent 1, \dots, content of parent n \rangle}` where $n \in \mathbb{N}, n > 1$

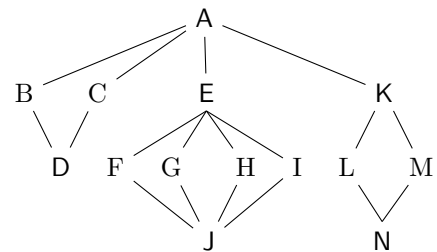
For every $i \in \mathbb{N}$ such that $0 < i \leq n$, create a new child of the current node's parent with content $\langle content of parent i \rangle$. Then detach the current node from its parent and attach it as the child of its n parents.

```
\begin{forest}
[A
[D,
multi={B,D}
]
]
\end{forest}
```



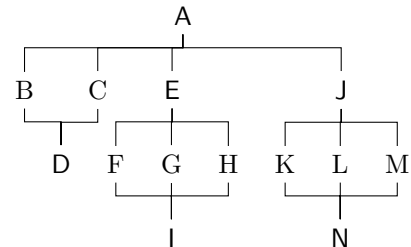
If `parent anchor` and/or `child anchor` are set, edges are drawn to/from these points as one would expect.

```
\begin{forest}
[A
[D, multi={B,C}, ]
[E, parent anchor=children,
[J, multi={F,G,H,I}, ]
]
[K
[N, multi={L,M}, child
↪ anchor=parent, ]
]
]
\end{forest}
```



If the `edges` library is loaded, the `multi` library loads the `TikZ` library, `ext.paths.ortho` and tries to emulate `forked edge` appropriately⁸.

```
\begin{forest}
forked edges,
[A
[D, multi={B,C} ]
[E
[I, multi={F,G,H} ]
]
[J
[N, multi={K,L,M} ]
]
]
\end{forest}
```

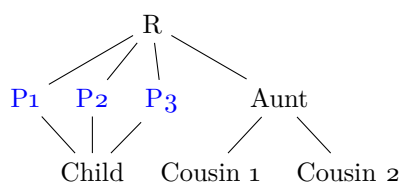


If we apply `forked edges` to only part of a tree, we can produce the rather ugly, but hopefully informative, structure below.

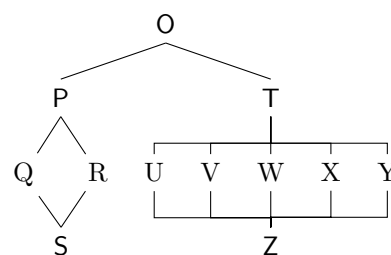
⁸The alignment seems to me to be close, but not always quite perfect, though I do not know why at the moment.

Box 3.5

```
\begin{forest}
  [R [Child, multi={P1,P2,P3},every parent=blue,]
    ↳ [Aunt [Cousin 1] [Cousin 2]]]
\end{forest}
```



```
\begin{forest}
  for tree={%
    child anchor=parent,
    parent anchor=children,
    fork sep'=1em,
  },
  [O
    [P
      [S, multi={Q,R} ]
      [T, forked edges=descendants,
        [Z,multi={U,V,W,X,Y} ]
      ]
    ]
  ]
\end{forest}
```



Note that the change to `fork sep` for the tree in `forest`'s preamble affects the edges drawn from and to the nodes inserted by `multi`. This is because the library forwards values given to `fork sep` and applications of `forked edge` so that `forest` keys work in (hopefully) reasonably intuitive ways.

Should you *not* want such keys forwarded, either load the library without defaults (see section 1) or override the behaviour for the current \TeX group with, say,

```
\forestset{%
  unautoforward=fork sep,
  null/.style={},
  forked edge'/.forward to=/forest/null,
}
```

The `phantom style` is needed because, unlike `forest`'s provision for its own forwarding facilities, `pgfkeys` provides no easy way to undo the effects of the `.forward to` handler.

Since the library is currently experimental and implementation is complicated if one wants to avoid using `forest` internals, configuration options are currently limited.

`every parent (keylist) = {⟨key-value list⟩}`

Apply `⟨key-value list⟩` to all the current node's parents. If `multi` is used, these are the parents created as a result; otherwise, it is the current node's singular parent or none, if the node has no parent.

Initial value: empty.

Box 3.5 illustrates usage with a simple example.

3.2 Connecting multiple parents

Sometimes one wants instead to give the current node an additional parent without removing the existing one and one does not wish to add the additional parent, but rather to specify some other extant node in the tree.

This kind of structure cannot be so easily automated, especially if one wants to avoid edges crossing each other or nodes. However, it is possible to provide some convenient styles to assist in manually specifying such structures.

```
also parent (style) = {⟨dynamic tree operation⟩}{⟨extant node⟩:⟨keylist⟩}
                  = {⟨dynamic tree operation⟩}{⟨extant node⟩}
+also parent (style) = {⟨extant node⟩:⟨keylist⟩}
                    = {⟨extant node⟩}
also parent+ (style) = {⟨extant node⟩:⟨keylist⟩}
                    = {⟨extant node⟩}
```

Adds $\langle extant node \rangle$ as an additional parent of the current node. $\langle keylist \rangle$ specifies a list of key-values for the connecting node (see below).

The current node becomes $\langle extant node \rangle$'s *fosterling*, while $\langle extant node \rangle$ becomes the current node's *foster parent*.

The styles work by creating a new child of $\langle extant node \rangle$. This node affects the structure of the tree and can be configured in the usual way, but it is not visible. One might say it is 'semi-phantom': it is not quite `phantom` because, for instance, it has visible edges which serve to connect the current node with the additional parent.

For an illustration, see the (rather odd-looking) family tree in box 3.6⁹.

`+also parent` prepends the new child to $\langle extant node \rangle$; `also parent+` appends it. These are just shorthand wrappers around `also parent` using the `prepend` and `append` dynamic tree operations.

Note that $\langle dynamic tree operation \rangle$ should create a new node, though this is not enforced.

```
fosterlings (step) Visit the current node's fosterlings.
foster parents (step) Visit the current node's foster parents.
every fosterling (step) = {⟨nodewalk⟩}
                        Visit every fosterling in ⟨nodewalk⟩.
every foster parent (step) = {⟨nodewalk⟩}
                        Visit every foster parent in ⟨nodewalk⟩.
c fosterling (step) Visit the fosterling which the current node connects to a foster parent.
c foster parent (step) Visit the foster parent which the current node connects to a fosterling.
```

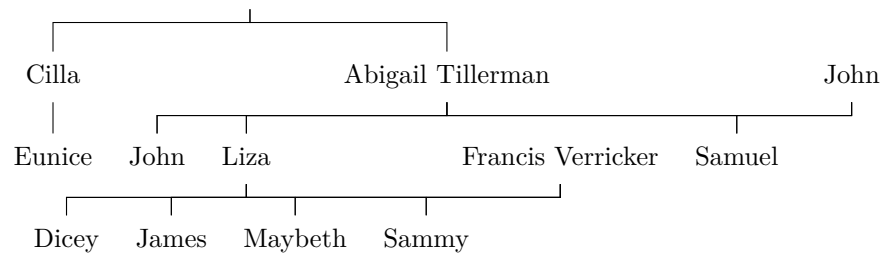
This last pair of steps are only really useful if you want to change `edge path`, since they are only accessible from a constructed, typically invisible node.

```
debug multi phantoms (bool. = true|false
                      reg.)
not debug multi phantoms
  (bool. reg.) Render the normally invisible nodes created by also parent etc. visible for debugging purposes.
                If the nodes have no content, their borders are drawn in red; otherwise, their contents are rendered
                in red. Visible rendering does not change the remainder of the tree e.g. it does not alter the
                spacing of nodes or the paths of edges. However, if the nodes occur near the tree's boundaries,
                the bounding box may expand to accommodate them10.
```

⁹The names are from the children's novels by Cynthia Voigt.

¹⁰It should not be hard to prevent this, but does not seem worth the trouble.

Box 3.6

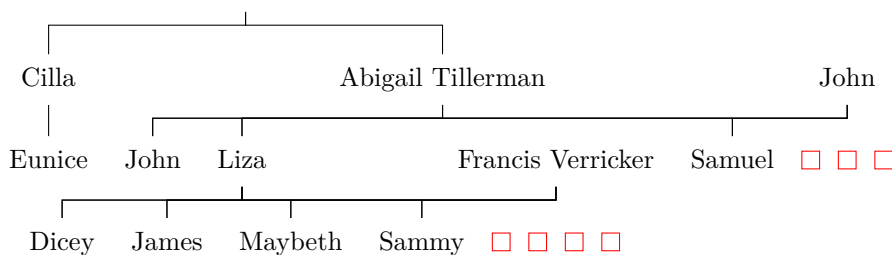


```

\begin{forest}
  forked edges,
  delay={%
    for tree={%
      +content=\strut,
    },
  },
  [,coordinate,calign primary child=1,calign secondary child=2,calign=midpoint,
    [Cilla
      [Eunice]
    ]
    [Abigail Tillerman
      [John,also parent={append}{j}]
      [Liza, also parent={append}{!r3}, for children={also parent={append}{!un}}
        [Dicey]
        [James]
        [Maybeth]
        [Sammy]
      ]
      [Francis Verricker,no edge]
      [Samuel, also parent+={!r3}]
    ]
    [John, name=j, no edge
  ]
\end{forest}

```


Box 3.7



```
{%
\forestset{debug multi phantoms}%
\begin{forest}
  forked edges,
  delay={%
    for tree={%
      +content=\strut,
    },
  },
[,coordinate,calign primary child=1,calign secondary child=2,calign=midpoint,
  [Cilla
    [Eunice]
  ]
  [Abigail Tillerman
    [John,also parent={append}{j}]
    [Liza, also parent={append}{!r3}, for children={also parent={append}{!un}}
      [Dicey]
      [James]
      [Maybeth]
      [Sammy]
    ]
    [Francis Verricker,no edge]
    [Samuel, also parent+={!r3}]
  ]
  [John, name=j, no edge]
]
\end{forest}%
}
```

Requires `ext.multi-debug`. If the debugging code is not loaded, use of these keys will do nothing but write a warning to the console and log.

For an example, see box 3.7. Note that the content of the `forest` environment is identical to that in box 3.6. The red squares are the effect of toggling `debug multi phantoms` beforehand.

4 Linguistics extensions

This library provides some elementary styles for formatting trees involving *multi-dominance*, together with a style for dealing with empty nodes resistant to the linguistics library's `nice empty nodes`. These former were developed in response to a query from Alan Munn on T_EX SE.

See also section 3, especially for straight connections to multiple parents and dynamic creation of multiple parents as children of a single grandparent.

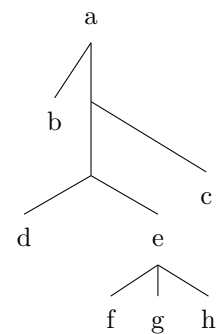
```
pretty nice empty nodes = {\keylist}
                        (style)
```

Make empty nodes prettier in cases where `nice empty nodes` cannot be used. `<keylist>` permits supplementing or overriding what is done for empty nodes.

Note that `nice empty nodes` is preferable, so should be used where possible. For details, see the documentation of `nice empty nodes` in Živanović (2017).

For example¹¹,

```
\begin{forest}
  for tree={
    calign angle=60,
    align middle child,
  },
  pretty nice empty nodes={
    for current and
    ↪ siblings={anchor=parent},
    parent anchor=children,
    calign with current edge,
  },
  [a
  [b
  [
  [
    [d]
    [e
    [f]
    [g]
    [h]
    ]
  ]
  [c]
  ]
  ]
\end{forest}
```



5 Utilities

This library provides *tagging keylists*, together with a few styles which do not really fit anywhere else.

5.1 Alignment

`align middle child (style) = <option>`

If the current node has an odd number of children, sets `calign child` to the middle child and sets `calign = <option>`. `<option>` should, therefore, be a valid value for `calign`.

If `<option>` is omitted, a default of `child edge` is applied.

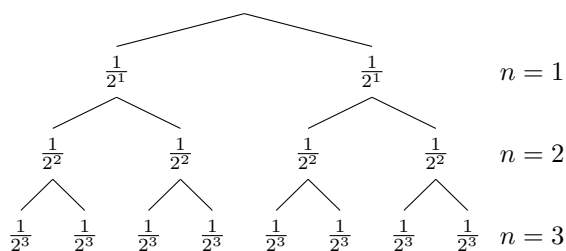
See box 4.1 for an example.

`align middle children (style) = <option>`

Sets `align middle child = <option>` for the tree.

¹¹Based on T_EX SE answer: [717677](#). Based on T_EX SE question [717592](#) by argo.

Box 5.1



```

\begin{forest}
  for tree={
    parent anchor=children,
    child anchor=parent,
  },
  delay={
    for descendants={
      content/.process={0w{level}}{\frac{1}{2^{#1}}},
    },
    for nodewalk={
      fake=root,
      while nodewalk valid={1}{1}%
    }{
      outer label/.process={0w{level}}{\{$n=#1$\}: {anchor=west}}}%
    },
  },
  [ [ [ ] ] [ ] ] [ [ ] ] [ [ ] ] ]
\end{forest}

```

5.2 Outer labels

Outer labels are nodes added after the tree is drawn, aligned with a boundary of the bounding box of the completed tree and nodes within the tree. The idea is to enable the addition of labels such as those shown in box 5.1.

`outer labels at` (*toks reg.*) = $\langle anchor \rangle$

Additional alignment point for any outer labels. $\langle anchor \rangle$ should be a valid anchor for the ‘current bounding box’ when the tree has been drawn, but additional code is not yet executed.

The default is `east`, which is probably what is wanted for most trees using the `forest` default value of `grow` etc.

Note that this is a *register*. You cannot use different values for different parts of a tree.

`outer labels` (*keylist reg.*) = $\{\langle keylist \rangle\}$

PGF/TikZ key-values applied to all nodes where `outer label` is set. Options passed to `outer label` are applied later, so may override defaults for the tree.

The default is `anchor=base west`.

Note that this is a *register*. You cannot use different values for different parts of a tree.

`outer label` (*style*) = $\{\langle content \rangle\}$

= $\{\langle content \rangle\}:\{\langle options \rangle\}$

Create a label aligned with the current node and the additional alignment point specified by `outer labels at` with content $\langle content \rangle$. If $\langle options \rangle$ are given, they are passed to the code responsible for creating the node.

5.3 ‘Tagging’ keylists

A ‘tagging keylist’ is very similar to a `forest keylist option`, but its default value can be changed and/or it can be redeclared¹². For motivation, see section 2.

More specifically, *inside* a `forest` environment, it behaves exactly like a regular `forest keylist option`¹³. However, *outside* a `forest` environment, its default value can be modified and/or replaced. Where this is not a requirement, you should use a regular `keylist` option since *tagging keylists* are subject to additional limitations and the implementation is significantly less efficient.

Important:

1. These keys are not really tagging-specific and do not require tagging to be active, despite the names, so may be useful in other contexts.
2. These keys are only available *outside* `forest` environments.
3. *Tagging keylists* cannot be declared as registers¹⁴. Each tagging keylist corresponds to a *keylist option*. The option is automatically declared just before every `forest` environment in the current `TEX` group.
4. An additional `TEX` group is added to all `forest` environments. This ensures that the option declaration is properly localised, which in turn allows any tagging keylists’ default values to be further manipulated after the current `forest` is finished.
5. *Outside* `forest` environments, unlike `forest` keylists, tagging keylists are *not* ordered and do *not* store more than one instance of any key. The underlying implementation uses `l3prop` property lists.
Inside `forest` environments, tagging keylists *are* ordered and behave as regular `forest` keylist options. `l3prop` property lists are *not* used inside `forest` environments.
6. *Outside the forest environment, they may be manipulated **only** using the keys defined by this library.*
7. *Inside the forest environment, they may be manipulated **only** using regular forest methods.*

Note that to actually influence a tree, any tagging keylist must be processed during the construction of that tree. Simply declaring a tagging keylist with some set of options will not, in itself, affect the typeset result in anyway. This is equally true of regular `forest` keylists. Please see Živanović (2017) for details.

```
declare tagging = {⟨keylist⟩}{⟨key-value list⟩}
keylist, redeclare tagging
keylist (code key)
```

Declares or redeclares a `forest keylist option`.

Available only outside forest environments.

Since keylists cannot actually be redeclared, what really happens is this:

- An internal property list is defined to hold *⟨default⟩*. This may then be manipulated using the various keys explained below.
- At the start of each `forest` environment (within the current `TEX` group), a `keylist` option is declared. The default value passed to `declare keylist` is *not* necessarily *⟨key-value list⟩*. It is, rather, a key-value list derived from the contents of the underlying property list at the time. Hence, the default may be further manipulated after the `keylist` option is declared.

¹²As far as I can tell, this is not possible for regular `forest` keylist options. Once declared, their default values are fixed.

¹³This is because it *is* a regular `keylist` option at this point.

¹⁴This is not a limitation since changing the default value of a *keylist register* is trivial.

Note that if you do not want the default be be manipulable after the keylist is declared, you should use the forest key `declare keylist={⟨keylist⟩}{⟨key-value list⟩}` instead, as this will be far more efficient.

`tagging keylist put (code = {⟨keylist⟩}{⟨key-value list⟩}`

key) Adds the contents of ⟨key-value list⟩ to a ⟨keylist⟩ declared with `declare tagging keylist`.

Note that if ⟨key-value list⟩ includes an occurrence of a key already in the list, the key will be replaced, even if the value differs.

`tagging keylist remove key = {⟨keylist⟩}{⟨key⟩}`

(*code key*) Removes ⟨key⟩ from ⟨keylist⟩, where ⟨keylist⟩ was previously declared with `declare tagging keylist`.

Available only outside forest environments.

Note this removes the ⟨key⟩ regardless of its current value (if any).

`tagging keylist remove (code = {⟨keylist⟩}{⟨key-value list⟩}`

key) For each ⟨key⟩ or ⟨key⟩=⟨value⟩ pair in ⟨key-value list⟩, removes ⟨key⟩ from ⟨keylist⟩ iff it has the specified ⟨value⟩ (if given) or no value (otherwise), where ⟨keylist⟩ was previously declared with `declare tagging keylist`.

Available only outside forest environments.

Note that a valueless key is distinct from one with an empty value. To remove ⟨key⟩ iff it has no value, use ⟨*key*⟩. To remove ⟨key⟩ iff it's value is empty, use ⟨*key*⟩= or ⟨*key*⟩=.

6 Implementation

A double underscore (__) or an 'at' (@) indicates an internal macro or key. These are liable to change without notice and should not be used elsewhere.

ext.ling

Clea F. Rees*

2026/01/17

<*sty>

```
1 \NeedsTeXFormat{LaTeX2e}
2 %% $Id: forest-ext-ling.dtx 11498 2026-01-17 00:35:22Z cfrees $}
3 \!debug \ProvidesForestLibrary{ext.ling}[2025-12-05 v0.1]
4 \debug \ProvidesForestLibrary{ext.ling-debug}[2025-12-05 v0.1]
5 %
6 \!debug \disable@package@load {forest-lib-ext.ling-debug}
7 \debug \disable@package@load {forest-lib-ext.ling}
8 {%
9 \!debug \PackageWarning {ext.ling (forest library)}
10 \debug \PackageWarning {ext.ling-debug (forest library)}
11 {Only one of ext.ling and ext.ling-debug should be loaded.
12 Since the
13 \!debug ext.ling
14 \debug ext.ling-debug
15 library has already been loaded, I will ignore your request for
16 \!debug ext.ling-debug.%
17 \debug ext.ling.%
18 }%
19 }
```

<*debug> </debug>

pretty nice empty nodes (*style*) This is in the ext.ling library mostly because nice empty nodes is in the linguistics library and not because linguists are more picky about their empty nodes than anybody else.

Is this even still useful?

Based on T_EX SE answer: [717677](#). i gwestiwn Based on T_EX SE question [717592](#) by argo.

```
20 \forestset{%
21   pretty nice empty nodes/.style={%
22     for tree={%
23       calign=fixed edge angles,
24       parent anchor=children,
25       delay={%
26         if content={} {%
27           inner sep=0pt,
28           edge path'={(!u.parent anchor) -- (.children)},
29           #1,
30         } {},
31       },
32     },
33   },
34 }
```

*Bug tracker: codeberg.org/cfr/prooftrees/issues | Code: codeberg.org/cfr/prooftrees | Mirror: github.com/cfr42/prooftrees

</sty>

ext.multi

Clea F. Rees*

2026/01/17

<*sty>

```
35 \NeedsTeXFormat{LaTeX2e}
36 %% $Id: forest-ext-multi.dtx 11495 2026-01-17 00:08:58Z cfrees $}
37 \!debug \ProvidesForestLibrary{ext.multi}[2025-12-05 v0.1]
38 \debug \ProvidesForestLibrary{ext.multi-debug}[2025-12-05 v0.1]
39 %
40 \!debug \disable@package@load {forest-lib-ext.multi-debug}
41 \debug \disable@package@load {forest-lib-ext.multi}
42 {%
43 \!debug \PackageWarning {ext.multi (forest library)}
44 \debug \PackageWarning {ext.multi-debug (forest library)}
45 {Only one of ext.multi and ext.multi-debug should be loaded.
46 Since the
47 \!debug ext.multi
48 \debug ext.multi-debug
49 library has already been loaded, I will ignore your request for
50 \!debug ext.multi-debug.%
51 \debug ext.multi.%
52 }%
53 }

54 \forestset{
```

Public options.

`every parent` (*keylist*) Keylists.

`other parents` (*keylist*)

```
55 declare keylist={every parent}{},
56 declare keylist={other parents}{},
```

Generic toks.

Internal options.

```
57 declare boolean={multi@connector}{0},
58 declare count={multi@n@parents}{0},
59 declare count={multi@connects@fosterling}{-1},
60 declare count={multi@connects@foster@parent}{-1},
61 declare keylist={multi@foster@parents}{},
62 declare keylist={multi@fosterlings}{},
63 declare keylist={multi@all@parents}{},
64 declare toks={multi@edge}{},
65 declare toks={multi@edge@subpath}{edge},
66 declare toks={multi@edge@sublast}{--},
67 declare toks={multi@edge@route}{--},
68 declare toks={multi@parent@of}{},
```

*Bug tracker: codeberg.org/cfr/prooftrees/issues | Code: codeberg.org/cfr/prooftrees | Mirror: github.com/cfr42/prooftrees

My answer: [695602](#).. Based on T_EX SE answer: [695600](#) by Alan Munn., which was based on my original answer.

Public registers.

```
69 <debug>      declare boolean register={debug multi phantoms},
70 <debug>      not debug multi phantoms,
```

Internal scratch registers.

```
71 declare count register={multi@temp@counta},
72 multi@temp@counta=0,
73 declare toks register={multi@temp@toksa},
74 multi@temp@toksa={},
75 declare toks register={multi@temp@toksb},
76 multi@temp@toksb={},
```

```
fosterlings (step) Convenience multi-step nodewalk steps.
foster parents (step)
every fosterling (step)
every foster parent (step)
  c fosterling (step)
  c foster parent (step)
77 define long step={fosterlings}{-}{%
78   if multi@fosterlings={-}{-}{%
79     split option={multi@fosterlings}{-}{id}%
80   }%
81 },
82 define long step={foster parents}{-}{%
83   if multi@foster@parents={-}{-}{%
84     split option={multi@foster@parents}{-}{id}%
85   }%
86 },
87 define long step={every fosterling}{n args=1}{%
88   filter={#1}{>0_={!multi@foster@parents}{-}}%
89 },
90 define long step={every foster parent}{n args=1}{%
91   filter={#1}{>0_={!multi@fosterlings}{-}}%
92 },
93 define long step={c fosterling}{-}{%
94   id/.option=multi@connects@fosterling%
95 },
96 define long step={c foster parent}{-}{%
97   id/.option=multi@connects@foster@parent%
98 },
```

multi (style) Make this node a grandchild of its current parent and insert specified parents.

```
99 multi/.style={%
100 <debug>      debug@multi=Execute style multi at,
101 <debug>      debug@multi@option=id,
102   split={#1}{-}{multi@parent},
103 <debug>      debug@multi@option=multi@n@parents,
104 <debug>      debug@multi@option=multi@all@parents,
105   before typesetting nodes={%
106     multi@parents/.process={%
107       000w
108       {name}
109       {multi@n@parents}
110       {multi@all@parents}
111       {#{#1}}%
112     },
113   delay n=2{%
114     multi@edge+= {(child anchor) },
115     multi@temp@counta'=0,
116     split option={multi@all@parents}{-}{multi@also@parent},
```

```

117     },
118     delay n=3{%
119 <debug>         debug@multi@option=id,
120 <debug>         debug@multi@option=multi@edge,
121             edge path'/.option=multi@edge,
122     },
123 },
124 },

```

`multi@also@parent` (*style*) Auxiliary.

```

125 multi@also@parent/.style={%
126 <debug>         debug@multi=Execute style multi@also@parent to for #1 at,
127 <debug>         debug@multi@option=id,
128     multi@temp@counta'+=1,
129     multi@edge+/.process={
130         OR= ? 0 w
131         {multi@n@parents}{multi@temp@counta}
132         {multi@edge@sublast}{multi@edge@subpath}
133         {##1 (#1.parent anchor) }
134     },
135 },

```

`multi@parent` (*style*) Insert a co-parent.

Note delay required in case name specified by user.

```

136 multi@parent/.style={%
137 <debug>         debug@multi=Execute style multi@parent to add #1 at,
138 <debug>         debug@multi@option=id,
139 <debug>         debug@multi@option=every parent,
140     multi@n@parents'+=1,
141     delay/.process={0w{multi@n@parents}{%
142         multi@all@parents+/.process={0w{name}{parent ##1 of #####1}},
143         insert before/.process={%
144             00w2{name}{every parent}
145             {%
146                 [#1,name=parent ##1 of #####1,multi@parent@of=####1,
147                 #####2]%
148             }%
149         },
150     }%
151 },
152 },
153 % \end{fstyle}% ^^A >>>
154 % \begin{fstyle}{multi@parents}% ^^A <<<
155 %   Adjust relns.
156 %
157 %   Arguments: name, no.~parents, parents
158 %   \begin{macrocode}
159 multi@parents/.style n args=3{%
160 <debug>         debug@multi=Executing style multi@parents with,
161 <debug>         debug@multi=options #1 #2 and #3,
162     if={>nw+P{#2}{isodd(##1)}}{%
163         multi@temp@counta/.expanded=\interval{(##2 + 1)/2},
164         for nodewalk={%
165             name/.expanded=parent \forestregister{multi@temp@counta} of #1 %
166         }{%
167             append=#1,
168         },
169     }{%
170         multi@temp@counta/.expanded=\interval{#2/2},

```

```

171 <debug>      debug@multi@register=multi@temp@counta,
172      for nodewalk={%
173      name/.expanded=parent \forestregister{multi@temp@counta} of #1 %
174      }{%
175      insert after={%
176      [,coordinate,no edge,
177      tier=multi@tier@#1@parents,
178      delay={%
179 <debug>      debug@multi=Appending #1 to,
180 <debug>      debug@multi@option=name,
181      append=#1,
182      },
183      ]%
184      },
185      },
186      },
187      },

```

`multi@add@parent` (*style*) Connect an additional parent.

Argument: id of current node; dynamic tree operation; keylist for child; id of additional parent (extant).

```

188 multi@add@parent/.style n args=4{%
189 <debug>      debug@multi=Execute style multi@add@parent to add #4 at,
190 <debug>      debug@multi@option=id,
191 <debug>      debug@multi=Arguments: #1: #2: #3: #4,
192 <debug>      debug@multi@option=every parent,
193      delay n=2{%
194      for nodewalk={%
195      id=#4%
196      }{%
197      multi@fosterlings+=#1,
198 <debug>      debug@multi=dynamic action #2,
199 <debug>      debug@multi@option=parent anchor,
200      #2={%
201      [,
202      multi@phantom,
203      multi@connector,
204      multi@connects@fosterling=#1,
205      multi@connects@foster@parent=#4,
206      for current/.option={!{id=#1}.every parent,
207      delay n=3{%
208      do dynamics,
209      edge path'/.process={%
210      Ow {!!{id=#1}.multi@edge@route} {%
211      (!c fosterling.child anchor)
212      ##1 (!c foster parent.parent anchor) %
213      }%
214      },
215      also={#3},
216 <debug>      debug@multi@option=id,
217 <debug>      debug@multi=edge path and edge,
218 <debug>      typeout/.option=edge path,
219 <debug>      debug@multi@option=edge,
220      },
221      ]%
222      },%
223      },
224      },
225      },

```

`also parent (style)` Make an existing node in the tree an additional parent of the current node. What actually happens is that the specified node gets a new child with a copy of the current node's content.

`+also parent (style)` However, this child is just like a `phantom`, except that it has a visible edge. This edge can then be defined to look as if it connects the current node to the additional parent.

Arguments: dynamic tree operation; parent:options for new node

Why do I need to double hashes twice here?

```

226  also parent/.style 2 args={%
227  <debug>    debug@multi=Executing style also parent at,
228  <debug>    debug@multi@option=id,
229    delay={%
230      split={#2}{:}{multi@temp@toksa,multi@temp@toksb},
231  <debug>    debug@multi@register=multi@temp@toksa,
232  <debug>    debug@multi@register=multi@temp@toksb,
233    if nodewalk valid={name/.register=multi@temp@toksa}{%
234      multi@temp@counta/.option={!{name/.register=multi@temp@toksa}.id,
235    }{%
236      multi@temp@counta/.option/.process={Rw{multi@temp@toksa}{##1.id}},
237    },
238  <debug>    debug@multi@register=multi@temp@counta,
239    multi@foster@parents+/.register=multi@temp@counta,
240    multi@add@parent/.process={%
241      O_RwR
242      {id}
243      {#1}
244      {multi@temp@toksb}
245      {{##1}}
246      {multi@temp@counta}%
247    },
248  },
249  },
250  +also parent/.style={%
251  <debug>    debug@multi=Executing style +also parent at,
252  <debug>    debug@multi@option=id,
253    also parent={prepend}{#1},
254  },
255  also parent+/.style={%
256  <debug>    debug@multi=Executing style also parent+ at,
257  <debug>    debug@multi@option=id,
258    also parent={append}{#1},
259  },

```

`multi@phantom (style)` Not really phantoms.

```

260  multi@phantom/.style={
261    before drawing tree={%
262  <debug>      if debug multi phantoms={%
263  <debug>        rectangle,
264  <debug>        if content={}{},
265  <debug>        draw=red,
266  <debug>      }{%
267  <debug>        red,
268  <debug>      },
269  <debug>    }{%
270    coordinate,
271  <debug>    },
272    typeset node,
273  },
274  },

```

`add parent (style)` Add a new node to the tree and connect it to the current node.

`debug@multi (style)` Internal styles for debugging. Should not be used directly, but may be applied by loading the

`debug@multi@register (style)` debugging code.

`debug@multi@option (style)`

```

275 <debug>      debug@multi/.code={%
276 <debug>        \ExpandArgs {e} \typeout{[Forest ext.multi debug]:: \detokenize{#1}}%
277 <debug>      },
278 <debug>      debug@multi@register/.code={%
279 <debug>        \ExpandArgs {e} \typeout{[Forest ext.multi debug]:: \detokenize{#1}
280 <debug>        = \forestregister{#1}}%
281 <debug>      }%
282 <debug>    },
283 <debug>    debug@multi@option/.code={%
284 <debug>      \ExpandArgs {e} \typeout{[Forest ext.multi debug]:: \detokenize{#1}
285 <debug>      = \forestoption{#1}}%
286 <debug>    }%
287 <debug>  },

```

`debug multi phantoms (style)` Supply a code key as substitute for the boolean if the debugging code isn't loaded.

`not debug multi phantoms (style)`

```

288 <!debug>      debug multi phantoms/.code={%
289 <!debug>        \PackageWarning{forest-lib-ext.multi}{%
290 <!debug>          You requested the style 'debug multi phantoms',
291 <!debug>          but did not load the debugging code.
292 <!debug>          Either load 'ext.multi-debug' instead of
293 <!debug>          'ext.multi' or remove this style.
294 <!debug>        }%
295 <!debug>      },
296 <!debug>      node debug multi phantoms/.code={%
297 <!debug>        \PackageWarning{forest-lib-ext.multi}{%
298 <!debug>          You requested the style 'not debug multi phantoms',
299 <!debug>          but did not load the debugging code.
300 <!debug>          Either load 'ext.multi-debug' instead of
301 <!debug>          'ext.multi' or remove this style.
302 <!debug>        }%
303 <!debug>      },

```

We need empty defaults here so that e.g. `ext.ling` can be loaded without `edges`, in which case the set of default is empty. If `edges` is loaded, we overwrite this style in a hook at `begindocument`.

```

304 <!debug>      libraries/ext.multi/defaults/.style=
305 <debug>      libraries/ext.multi-debug/defaults/.style=
306   {},
307 }

```

We need conditional code in case `edges` is loaded.

```

308 \AddToHook{begindocument}{%
309   \IfPackageLoadedT{forest-lib-edges}{%
310 <!debug>     \PackageInfo{forest-lib-ext.multi}
311 <debug>     \PackageInfo{forest-lib-ext.multi-debug}
312     {Found the edges library. Enabling support code.}%
313     \usetikzlibrary{ext.paths.ortho}%
314     \forestset{%

```

`multi@forked@edge (style)` I wish forest used booleans or similar a bit more extensively here so this was easier to handle (without clobbering).

```

315     multi@forked@edge/.style={%
316     /forest/.cd,

```

```

317         /tikz/ext/ortho/distance/.process={0w{fork sep}{-##1}},
318     every parent+={%
319         forked edge,
320         /tikz/ext/ortho/distance/.process={0w{fork sep}{-##1}},
321     },
322     multi@edge@subpath={%
323         (.child anchor) -|-
324     },
325     multi@edge@sublast={%
326         (.child anchor) -|-
327     },
328     multi@edge@route={%
329         -|-
330     },
331 },

```

Setup some (hopefully) intuitive defaults.

A negative value for `ext/ortho/distance` is measured from the target coordinate rather than the start. We are constructing the paths backwards in comparison with `forest`. `0.5em` is the `forest` default. Do **not** try passing the option value here!

```

332 <!debug>     libraries/ext.multi/defaults/.style=
333 <debug>      libraries/ext.multi-debug/defaults/.style=
334     {%
335     default preamble+={
336         Autoforward={fork sep}{%
337             every parent+={%
338                 fork sep=##1,
339                 edge+={/tikz/ext/ortho/distance=-##1},
340             },
341             edge+={/tikz/ext/ortho/distance=-##1},
342         },
343         fork sep'=0.5em,
344         forked edge'/.forward to=/forest/multi@forked@edge,
345     },
346 },

```

There's no 'hook' mechanism for this, so there is not really a nice or robust way of doing this, I don't think. For the `edges` library, in particular, there are, in fact, no defaults at all ...

```

347     libraries/edges/defaults/.append style={%
348 <!debug>     libraries/ext.multi/defaults,
349 <debug>      libraries/ext.multi-debug/defaults,
350     /utils/exec={%
351 <!debug>         \PackageInfo{forest-lib-ext.multi}
352 <debug>         \PackageInfo{forest-lib-ext.multi-debug}
353     }%
354     Appending compatibility code for forked edges to default settings.%
355     }%
356 },
357 }%
358 }%
359 }%
360 }

```

</sty>

ext.tagging

Clea F. Rees*

2026/01/17

<*sty> <@@=tagforest>

```
361 \NeedsTeXFormat{LaTeX2e}
362 %% $Id: forest-ext-tagging.dtx 11499 2026-01-17 00:49:37Z cfrees $}
363 <!debug> \ProvidesForestLibrary{ext.tagging}[v0.1]
364 <debug> \ProvidesForestLibrary{ext.tagging-debug}[v0.1]
365 %
366 <!debug> \disable@package@load {forest-lib-ext.tagging-debug}
367 <debug> \disable@package@load {forest-lib-ext.tagging}
368 {%
369 <!debug> \PackageWarning {ext.tagging (forest library)}
370 <debug> \PackageWarning {ext.tagging-debug (forest library)}
371 {Only one of ext.tagging and ext.tagging-debug should be loaded.
372 Since the
373 <!debug> ext.tagging
374 <debug> ext.tagging-debug
375 library has already been loaded, I will ignore your request for
376 <!debug> ext.tagging-debug.%
377 <debug> ext.tagging.%
378 }%
379 }
```

As the name suggests, we need *tagging keylists* from ext.utils.

```
380 <!debug> \useforestlibrary*{ext.utils}
381 <debug> \useforestlibrary*{ext.utils-debug}
382 \ExplSyntaxOn
```

\l__tagforest_toks_tl expl3 variable to store non-expl3 *toks*.

```
383 \tl_new:N \l__tagforest_toks_tl
```

\l__tagforest_tmpa_str

```
384 \str_new:N \l__tagforest_tmpa_str
```

forestext_tagging_custom_bool Public boolean to allow custom config to override e.g. prooftrees.

custom tagging (*code key*)

```
385 \bool_new:N \l_forestext_tagging_custom_bool
386 \bool_set_false:N \l_forestext_tagging_custom_bool
387 \forestset{
388 custom tagging/.code={
389 \use:c {bool_set_#1:N} \l_forestext_tagging_custom_bool
390 },
391 custom tagging/.default=true,
```

*Bug tracker: codeberg.org/cfr/prooftrees/issues | Code: codeberg.org/cfr/prooftrees | Mirror: github.com/cfr42/prooftrees

```

392 not custom tagging/.code={
393   \bool_set_false:N \l_foresttext_tagging_custom_bool
394 },
395 }

tagforest_pgftikz_tag_bbox:nmn Retrieve saved coordinates.
tagforest_pgftikz_tag_bbox:enn
396 \cs_new_nopar:Npn \__tagforest_pgftikz_tag_bbox:nnn #1#2#3
397 {
398   \__tagforest_pgftikz_tag_bbox_aux:eenn
399   {
400     \property_ref:ee {#1}{xpos}
401   }
402   {
403     \property_ref:ee {#1}{ypos}
404   }
405   {#2}{#3}
406 }
407 \cs_generate_variant:Nn \__tagforest_pgftikz_tag_bbox:nnn {enn}

```

rest_pgftikz_tag_bbox_aux:nnnn The tagging code requires the bounding box for *alt*. Getting the exact value would require something more complicated, but this calculates a reasonable approximation for simple cases. It is not exact because it does not, for instance, account for line widths at the least and greatest coordinates. A more serious deficiency is that it ignores any annotations added to the tree, including labels, edge labels, additional drawing commands etc.

The problem here is that if we wait until the end of the `tikzpicture`, the tokens required to create the `alt` text no longer exist. A better solution might be to `memoize` the tree and get the bounding box that way. Then we can simply write the tokens we need to file and access them at any point during subsequent compilations. Or maybe it would be better to just save the tokens and write this after the tree is drawn? But then we probably have to save them globally in order to ‘smuggle’ them out, which is a bit obnoxious.

```

408 \cs_new_nopar:Npn \__tagforest_pgftikz_tag_bbox_aux:nnnn #1#2#3#4
409 {
410   \dim_to_decimal_in_bp:n {#1sp}
411   \c_space_tl
412   \dim_to_decimal_in_bp:n {#2sp}
413   \c_space_tl
414   \dim_to_decimal_in_bp:n {#1sp+#3}
415   \c_space_tl
416   \dim_to_decimal_in_bp:n {#2sp+#4}
417 }
418 \cs_generate_variant:Nn \__tagforest_pgftikz_tag_bbox_aux:nnnn {eenn}

```

tagsupport/forest/init (socket) Is there an equivalent of the macro environment for sockets/plugs/hooks?

tagsupport/forest/tag (socket) The support for `memoize` is currently `noop`, but we create the sockets.

support/forest/tag/mmz (socket) tagsupport/forest/setup doesn’t correspond to anything in latex-lab (L^AT_EX Project 2025a) because I’m not sure how to make it.

```

419 \socket_new:nn {tagsupport/forest/init}{0}
420 \socket_new:nn {tagsupport/forest/tag}{2}
421 \socket_new:nn {tagsupport/forest/tag/mmz}{2}
422 \socket_new:nn {tagsupport/forest/setup}{0}

```

__tagforest_tag_suspend:n We are going to redefine the standard `\tag_suspend:n` and `\tag_resume:n` to prevent the tagging code being continuously stopped and started during tree construction. Before doing that, we make private copies of both commands so that we can (i) still stop/start tagging ourselves and (ii) restore the original definitions when we’re done.

This rather less than ideal solution is required because there is no way to disable the tagging support for `tikz` locally: the only documented way to disable is global. But we do not want to interfere with `latex-lab`'s tagging code for *other* `tikzpicture` environments. We just want to stop it interfering in `forest` trees. Hence the hacks.

```
423 \cs_new_eq:NN \__tagforest_tag_suspend:n \tag_suspend:n
424 \cs_new_eq:NN \__tagforest_tag_resume:n \tag_resume:n
```

`__tagforest_noop:n` Something to `\let` the suspend/resume functions to.

```
425 \cs_new_nopar:Npn \__tagforest_noop:n #1 {}
```

`tagsupport/forest/inittag (plug)` This plug corresponds roughly to `tagsupport/tikz/picture/init`, but the division of labour between sockets/plugs is a bit different for `forest`.

```
426 \socket_new_plug:nnn {tagsupport/forest/init}{tag}
427 {
```

This part is modified from L^AT_EX Project (2025b), but runs in a different socket. I had a note that using socket `para/begin` didn't work here. That's probably from `tableaux`? But I can't remember what I thought the problem was

```
428 \mode_if_vertical:T
429 {
430   \if@inlabel
431     \mode_leave_vertical:
432   \else
433     \tag_socket_use:n {para/begin}
434   \fi
435 }
436 \tag_mc_end_push:
```

Note that assigning `noop` to all of the `latex-lab` sockets and suspending tagging is **not** sufficient to suspend tagging. This is because hook code includes tagging commands, including commands which start/stop tagging, unconditionally.

```
437 \socket_assign_plug:nn {tagsupport/tikz/picture/init}{noop}
438 \socket_assign_plug:nn {tagsupport/tikz/picture/begin}{noop}
439 \socket_assign_plug:nn {tagsupport/tikz/picture/end}{noop}
```

This does the `forest` (Živanović 2017) setup before the *tagging keylists* are turned into *keylist options*.

```
440 \socket_use:n {tagsupport/forest/setup}
```

Since we can't disable that code only locally, we instead redefine the relevant commands. Even this does not completely pause the tagging code, but it stops enough to yield a valid structure, albeit one with a lot of empty mcs.

```
441 \cs_set_eq:NN \tag_suspend:n \__tagforest_noop:n
442 \cs_set_eq:NN \tag_resume:n \__tagforest_noop:n
```

Suspend tagging using private copy of public function.

```
443 \__tagforest_tag_suspend:n {tagforest}
444 }
```

`tagsupport/forest/setup alt (plug)` This doesn't correspond to anything in L^AT_EX Project (2025b) because I'm not sure how to make it.

```
445 \socket_new_plug:nnn {tagsupport/forest/setup}{alt}
446 {
447   \forestset{
```

```

448     tag plug=alt,
449     tag nodes uses=alt text,
450     collate tags uses=alt text,
451     tag tree uses=alt,
452   }
453 }

```

`tagsupport/forest/tag alt (plug)` This plug corresponds to latex-lab's `alt` plug for `tikz` (L^AT_EX Project 2025b). So far as possible, these plugs are verbatim copies of the official plugs¹, but some changes are necessary for `forest`.

```

454 \socket_new_plug:nnn {tagsupport/forest/tag}{alt}
455 {

```

Straight from latex-lab.

```

456   \tag_struct_begin:n
457   {
458     tag=Figure,
459     alt=\l__tagforest_toks_tl,
460   }
461   \tag_mc_begin:n {tag=Figure}
462   \cs_new:cpe {tagforest@mark@pos@the\tagforest@id}
463   {

```

The only real differences are that, as noted above, some code is used in the `init` socket rather than here and that we use different functions to determine the coordinates of the origin and size of the bounding box. Whereas latex-lab uses pgf's `remember picture` functionality to record the origin, we use `lproperties`. Similarly, where latex-lab uses pgf to determine the extent of the bounding box, we use `forest` to calculate approximate dimensions for the tree before it is drawn.

The use of `lproperties` is quite all right, I think, and necessary as latex-lab's method is not compatible with `forest`. However, latex-lab's bounding box calculation is *far* superior to the method used here, so it would be useful to see if that can be modified for use once the rest of the code works. (It should also be significantly faster.)

```

464   \__tagforest_pgftikz_tag_bbox:enn {tagforest-id\the\tagforest@id}
465   {#1}{#2}
466 }

```

Revert to copying latex-lab verbatim.

```

467   \tag_struct_gput:ene
468   {\tag_get:n {struct_num}}
469   {attribute}
470   {
471     /O /Layout /BBox
472     [
473       \use:c
474       {tagforest@mark@pos@the\tagforest@id}
475     ]
476   }
477 }

```

`__tagforest_init:` Corresponds to the analogous latex-lab function. Tests whether tagging is active and sets a `forest` boolean accordingly.

```

478 \cs_new_nopar:Npn \__tagforest_init:
479 {
480   \global\advance\tagforest@id by 1\relax
481   \tag_if_active:TF

```

¹In other words, the bits that work are shamelessly copied from Ulrike Fischer's code, while the bits which don't are mine.

```

482 {
483   \forestset{
484     tagging=1,
485 <debug>   debug tagforest={Tagging active.},
486   }

```

If tagging is active and unless custom tagging is set, installs sets register `plug` to `alt` and assigns plugs and sockets. If custom tagging is set, we just set the `forest` boolean `tagging` and make `__tagforest_end`: `noop`. The custom stages are still in place, but these should hopefully have no effect on anything. In any case, they are partially overridden by e.g. `prooftrees` which installs a somewhat different and more complicated set.

```

487   \bool_if:NTF \l_forestext_tagging_custom_bool
488   {
489 <debug>   \tagforest@debug@typeout{Custom tagging configured.}
490   \cs_set_eq:NN \__tagforest_end: \__tagforest_noop:n
491   }{
492 <debug>   \tagforest@debug@typeout{Custom tagging not configured.}
493   \str_if_eq:eeT {tagforest@plug@NONE} {\foresteregister{setup plug}}
494   {
495 <debug>   \tagforest@debug@typeout{Looking for setup plug as none configured.}
496   \socket_get_plug:nN {tagsupport/tikz/picture/begin} \l__tagforest_tmpa_str
497   \exp_args:NnV \socket_if_plug_exist:nnTF {tagsupport/forest/setup}
498   \l__tagforest_tmpa_str
499   {
500     \PackageInfo{ext.tagging (forest lib)}{
501       Installing setup plug for tagging forest trees to match selection for
502       tikz pictures.
503     }
504     \exp_args:Ne \forestset{setup plug \exp_not:N = \l__tagforest_tmpa_str}
505   } {
506     \PackageWarning{ext.tagging (forest lib)}{
507       Using alt setup plug for tagging as no match exists for plug
508       selected for tikz pictures
509     }
510     \forestset{setup plug=alt}
511   }
512 }
513 \str_if_eq:eeT {tagforest@plug@NONE} {\foresteregister{tag plug}}
514 {
515 <debug>   \tagforest@debug@typeout{Looking for tag plug as none configured.}
516   \exp_args:NnV \socket_if_plug_exist:nnTF {tagsupport/forest/tag}
517   \l__tagforest_tmpa_str
518   {
519     \PackageInfo{ext.tagging (forest lib)}{
520       Installing tag plug for tagging forest trees to match selection for
521       tikz pictures.
522     }
523     \exp_args:Ne \forestset{tag plug \exp_not:N = \l__tagforest_tmpa_str}
524   } {
525     \PackageWarning{ext.tagging (forest lib)}{
526       Using alt tag plug for tagging as no match exists for plug
527       selected for tikz pictures
528     }
529     \forestset{tag plug=alt}
530   }
531 }
532 <debug>   \tagforest@debug@typeout{Assigning plug tag to tagsupport/forest/init.}
533   \socket_assign_plug:nn {tagsupport/forest/init}{tag}
534 <debug>   \tagforest@debug@typeout{Using socket tagsupport/forest/init.}
535   \socket_use:n {tagsupport/forest/init}

```

We also do what we can to stop the residual tagging code from marking up useless content. This mitigates the problem, but does not entirely solve it.

```

536     \def\pgfsys@begin@text{}
537     \def\pgfsys@end@text{}
538   }
539 }{
540   \forestset{tagging=0,
541 <debug>   debug tagforest={Tagging inactive.},
542   }
543 }
544 }
```

`__tagforest_end:` Again, analogous to the corresponding latex-lab function (L^AT_EX Project 2025b).

```

545 \cs_new_nopar:Npn \__tagforest_end: {
546   \__tagforest_tag_resume:n {tagforest}
```

Restore the format’s definitions of `\tag_suspend:n` and `\tag_resume:n`.

```

547   \cs_set_eq:NN \tag_suspend:n \__tagforest_tag_suspend:n
548   \cs_set_eq:NN \tag_resume:n \__tagforest_tag_resume:n
```

Standardish?

```

549 <debug>   \if@tagforest@debug
550 <debug>   \ShowTagging{mc-current}
551 <debug>   \fi
552   \tag_mc_end:
553   \tag_struct_end:
554   \tag_mc_begin_pop:n {}
555 }
```

`__tagforest_tag_tree_tag:nnn` This function is responsible for recording the tree’s page coordinates, tidying up the collected `\tagforest@tag@tree@tag` tokens for the alt text and utilising the tagging socket.

```

556 \cs_new_nopar:Npn \__tagforest_tag_tree_tag:nnn #1#2#3
557 {
558   \tex_savepos:D
559   \property_record:ee {tagforest-id\the\tagforest@id}
560   {xpos,ypos}
561   \tex_savepos:D
562   \tl_set:Nx \l__tagforest_toks_tl {
563     \exp_args:No \text_purify:n { \the\tagforest@toks }
564   }
```

Utilising the socket requires briefly reenabling tagging else the commands would have no useful effects.

```

565   \__tagforest_tag_resume:n {tagforest}
566   \socket_assign_plug:nn {tagsupport/forest/tag}{#1}
567   \ifmemoizing
568     \socket_assign_plug:nn {tagsupport/forest/tag/mnz}{#1}
569   \fi
570   \socket_use:nnn {tagsupport/forest/tag}{#2}{#3}
571   \socket_use:nnn {tagsupport/forest/tag/mnz}{#2}{#3}
572   \__tagforest_tag_suspend:n {tagforest}
573 }
```

Alias for use in pgf syntax.

```

574 \cs_new_eq:NN \tagforest@tag@tree@tag \__tagforest_tag_tree_tag:nnn
```

\tagforest@init Alias.

```
575 \cs_new_eq:NN \tagforest@init \__tagforest_init:
```

\tagforest@end Alias.

```
576 \cs_new_eq:NN \tagforest@end \__tagforest_end:
```

```
577 \hook_gput_code:nnn {env/forest/end}{.}
578 {
579   \__tagforest_end:
580 }
581 \hook_gput_code:nnn {env/forest/begin}{.}
582 {
583   \__tagforest_init:
584 }
```

```
585 \ExplSyntaxOff
```

\tagforest@toks Name for a new toks and some ways to peek when debugging.

\LogTagForestToks

```
586 \newtoks\tagforest@toks
587 <debug> \newcommand \LogTagForestToks{%
588 <debug> \expandafter\typeout\expanded{[tagforest debug]:: current id:
589 <debug> \expandafter\noexpand\the\tagforest@toks}%
590 <debug> }
```

\tagforest@id Name for a new count.

\LogTagForestId

```
591 \newcount\tagforest@id
592 <debug> \newcommand \LogTagForestId{%
593 <debug> \expandafter\typeout\expanded{[tagforest debug]::
594 <debug> current id: \the\tagforest@id}%
595 <debug> }
```

\if@tagforest@debug Conditional for debugging.

\@tagforest@debugfalse

\@tagforest@debugtrue

\tagforest@debug@typeout

```
596 \newif\if@tagforest@debug
597 <!debug> \@tagforest@debugfalse
598 <debug> \@tagforest@debugtrue
599 \newcommand \tagforest@debug@typeout [1]{%
600   \if@tagforest@debug
601     \ExpandArgs {e} \typeout{[tagforest debug]:: \detokenize{#1}}%
602   \fi
603 }
```

```
604 \forestset{
```

For debugging. <*debug>

```
605 debug tagforest/.code={
606   \tagforest@debug@typeout{#1}%
607 },
```

</debug> Various additions in the form of forest options and registers. By default, these are noop.

```
608 declare boolean register={tagging},
609 tagging=0,
610 declare toks register={setup plug},
611 setup plug=tagforest@plug@NONE,
612 declare toks register={tag plug},
```

```

613 tag plug=tagforest@plug@NONE,
614 Autoforward register={setup plug}{%
615   TeX={%
616     \IfSocketPlugExistsTF {tagsupport/forest/setup}{#1}{%
617       \AssignSocketPlug {tagsupport/forest/setup}{#1}%
618     }{%
619       \PackageError{ext.tagging (forest library)}{%
620         No plug named '#1' exists for socket 'tagsupport/forest/setup'.%
621       }{%
622         See the forest-ext manual for details.%
623       }%
624     }%
625   },
626 },
627 Autoforward register={tag plug}{%
628   TeX={%
629     \IfSocketPlugExistsTF {tagsupport/forest/tag}{#1}{%
630       \AssignSocketPlug {tagsupport/forest/tag}{#1}%
631     }{%
632       \PackageError{ext.tagging (forest library)}{%
633         No plug named '#1' exists for socket 'tagsupport/forest/tag'.%
634       }{%
635         See the forest-ext manual for details.%
636       }%
637     }%
638   },
639 },
640 plug/.style={%
641   setup plug={#1},
642   tag plug={#1},
643 },

```

`tag nodes` (*tag. keylist*) I wanted to use a nodewalk styles for tagging and collation, but couldn't (easily) figure out how,
`collate tags` (*tag. keylist*) so sticking to keylists and processing orders for now. Hence, only the final step is a stage
 But I suspect there's a performance hit here (Or maybe not without comparing internals with public interfaces? `prooftrees` uses keylists and I don't think Sašo suggested substituting code keys for speed at any point? But that was a long time ago)

```

644 declare tagging keylist={tag nodes}{},
645 declare tagging keylist={collate tags}{},

```

`before tagging nodes` (*keylist*) Regular keylist options.

`before collating tags` (*keylist*)

`before tagging tree` (*keylist*)

```

646 declare keylist={before tagging nodes}{},
647 declare keylist={before collating tags}{},
648 declare keylist={before tagging tree}{},

```

`node@ttoks` (*auto. toks*) Private and public.

`alttext` (*auto. toks*)

```

649 declare autowrapped toks={node@ttoks}{},
650 declare autowrapped toks={alt text}{},

```

`is root` (*auto. toks reg.*) Structural descriptors.

`is leaf` (*auto. toks reg.*)

`is child` (*auto. toks reg.*)

`is edge label` (*auto. toks reg.*)

`has branches` (*auto. toks reg.*)

`is branch` (*auto. toks reg.*)

```

651 declare autowrapped toks register={is root},
652 is root={root},
653 declare autowrapped toks register={is leaf},
654 is leaf={end branch},
655 declare autowrapped toks register={is child},
656 is child={child},
657 declare autowrapped toks register={is edge label},

```

```

658   is edge label={edge label},
659   declare autowrapped toks register={has branches},
660   has branches={branches},
661   declare autowrapped toks register={is branch},
662   is branch={branch},

```

The tagging code depends on injecting additional processing steps into forest's processing of the tree. This requires redefining `stages` to include the extra steps. This has global effect, but hopefully does no harm

```

663   stages/.style={
664     for root'={
665       process keylist register=default preamble,
666       process keylist register=preamble
667     },
668     process keylist=given options,
669     process keylist=before typesetting nodes,
670     typeset nodes stage,
671     process keylist=before packing,
672     pack stage,
673     process keylist=before computing xy,
674     compute xy stage,

```

The additions for tagging are inserted between `compute xy stage` and before drawing tree.

```

675 <debug>   debug tagforest={Process keylist: before tagging nodes ...},
676   process keylist=before tagging nodes,
677 <debug>   debug tagforest={Process keylist: tag nodes ...},
678   process keylist=tag nodes,
679 <debug>   debug tagforest={Process keylist: before collating tags ...},
680   process keylist=before collating tags,
681 <debug>   debug tagforest={Process keylist: collate tags ...},
682   process keylist=collate tags,
683 <debug>   debug tagforest={Process keylist: before tagging tree ...},
684   process keylist=before tagging tree,
685 <debug>   debug tagforest={Stage: tag tree stage ...},
686   tag tree stage,
687 <debug>   debug tagforest={Completed all tagging stages!},
688   process keylist=before drawing tree,
689   draw tree stage
690 },
691 tag nodes processing order/.nodewalk style={unique=tree},
692 collate tags processing order={unique=tree depth first},
693 tag tree stage/.style={for root'=tag tree},

```

By default, the crucial stage does nothing.

```

694   tag tree/.style={},

```

Redefine various of the additions to `stages` to do something useful. The remaining additions are to allow user interventions.

We split this up so bits can be used more flexibly e.g. by `prooftrees`. `prooftrees` doesn't want the code which generates tags, but it does want `tag tree`. (Well, `prooftrees` had this code first, so it wants what `ext.tagging` is pinching.

tag nodes uses (*choice*) How to tag individual nodes. Currently, only nodes can be tagged.

```

695   tag nodes uses/.is choice,
696   tag nodes uses/noop/.style={%
697     redeclare tagging keylist={tag nodes}{},
698   },
699   tag nodes uses/alt text/.style={%

```

```

700     redeclare tagging keylist={tag nodes}{%
701         delay={%
702             if level=0{%
703                 if alt text={}{%
704                     +node@ttoks/.process={Rw{is root}{##1\ }},
705                 },
706             },
707         },
708         if phantom={}{%
709             if alt text={}{%
710                 if edge label={}{%
711 <debug>     debug tagforest/.process={0w {id}{Node id: ##1}},
712 <debug>     debug tagforest/.process={0w{content}{No edge label.
713 <debug>     Content is ##1}},
714                 node@ttoks/.process={0w{content}{##1}},
715             },
716 <debug>     debug tagforest/.process={0w {id}{Node id: ##1}},
717 <debug>     debug tagforest/.process={0w{edge label}{Edge label is ##1}},
718 <debug>     debug tagforest/.process={0w{content}{Content is ##1}},
719                 node@ttoks/.process={R00w3{is edge label}{content}{edge label}{\ ##1 ##2 ##3\
720             }},
721             if={>0_>{n children}{1}}{%
722 <debug>     debug tagforest/.process={00w2 {id}{n children}{Node id: ##1 has ##2 branches}},
723                 node@ttoks+/.process={ORw2{n children}{has branches}{\ ##1 ##2\ }},
724                 delay={%
725                     for children={%
726 <debug>     debug tagforest/.process={0w {id}{Node id: ##1}},
727 <debug>     debug tagforest/.process={0w{n}{Branch no. is ##1}},
728                 +node@ttoks/.process={R0w2{is branch}{n}{\ ##1 ##2\ }},
729                 },
730             },
731             },
732             if n children=1{%
733                 delay={%
734 <debug>     debug tagforest/.process={0w {id}{Node id: ##1 has 1 child}},
735                 !1.+node@ttoks/.process={Rw{is child}{\ ##1\ }},
736                 },
737             },
738 <debug>     debug tagforest/.process={0w {id}{Node id: ##1 is a leaf}},
739                 node@ttoks+/.process={Rw{is leaf}{\ ##1\ }},
740             },
741             },
742             delay n=2{%
743                 alt text'/.option=node@ttoks,
744             },
745             },
746         },
747     },
748 },

```

collate tags uses (*choice*) I don't really like this way of doing this. I'd rather use e.g. a .choice key or something for collate tags, but I'm not sure how to do that and have the keylist be public

```

749 collate tags uses/.is choice,
750 collate tags uses/noop/.style={%
751     redeclare tagging keylist={collate tags}{},
752 },
753 collate tags uses/alt text/.style={%
754     redeclare tagging keylist={collate tags}{%
755         collate tag/.option=alt text,

```



```

756     },
757 },

```

`collate tag` (*code key*) How to collate the tags.

```

758 collate tag/.code={%
759 <debug>          \tagforest@debug@typeout{Appending toks #1 .}%
760          \foresttext@toksapp\tagforest@toks{#1 }%
761 },

```

`tag tree uses` (*choice*) Calculate dimensions used to determine an approximate bounding box size.

```

762 tag tree uses/.is choice,
763 tag tree uses/noop/.style={%
764   tag tree/.style={},
765 },
766 tag tree uses/alt/.style={% wrong bbox!!
767   tag tree/.style={%
768     tempdimc/.max={>00w2+d{x}{max x}{####1+####2}}{tree},
769     tempdimc-/.min={>00w2+d{x}{min x}{####1+####2}}{tree},
770     tempdimd/.max={>00w2+d{y}{max y}{####1+####2}}{tree},
771     tempdimd-/.min={>00w2+d{y}{min y}{####1+####2}}{tree},
772 <debug>      debug tagforest={Dimensions (x then y) are },
773 <debug>      debug tagforest/.register=tempdimc,
774 <debug>      debug tagforest/.register=tempdimd,

```

The next line should create the tagging structure and insert the assembled `alt` text.

```

775 <debug>      debug tagforest/.process={RRRw3{tag plug}{tempdimc}{tempdimd}{%
776 <debug>      Tagging tree now with tag plug=####1, x=####2, y=####3 ...}},
777   TeX/.process={RRRw3{tag plug}{tempdimc}{tempdimd}{%
778     \tagforest@tag@tree@tag{####1}{####2}{####3}}%
779   },
780 },
781 },

782 }

</sty>

```

ext.utils

Clea F. Rees*

2026/01/17

<*sty> <@@=forestext>

```
783 \NeedsTeXFormat{LaTeX2e}
784 %% $Id: forest-ext-utils.dtx 11499 2026-01-17 00:49:37Z cfrees $}
785 \!debug \ProvidesForestLibrary{ext.utils}[2025-12-05 v0.1]
786 \debug \ProvidesForestLibrary{ext.utils-debug}[2025-12-05 v0.1]
787 %
788 \!debug \disable@package@load {forest-lib-ext.utils-debug}
789 \debug \disable@package@load {forest-lib-ext.utils}
790 {%
791 \!debug \PackageWarning {ext.utils (forest library)}
792 \debug \PackageWarning {ext.utils-debug (forest library)}
793 {Only one of ext.utils and ext.utils-debug should be loaded.
794 Since the
795 \!debug ext.utils
796 \debug ext.utils-debug
797 library has already been loaded, I will ignore your request for
798 \!debug ext.utils-debug.%
799 \debug ext.utils.%
800 }%
801 }
```

7 Toks etc.

Only used if other stuff isn't loaded.

```
802 \ExplSyntaxOn
```

`\forestext@toksapp` Avoid standard name in case the user loads code which defines the macro after loading our package.

```
803 \cs_new:Npn \forestext@toksapp#1#2{#1\expandafter{\the #1#2}}
804 \@ifpackageloaded{memoize}
805 {}{
806 \newif\ifmemoizing\memoizingfalse
807 }
```

`\socket_get_plug:nN` See <https://github.com/latex3/latex2e/issues/1851#issuecomment-3566374363>. I don't know the implementation status of Ulrike Fischer's suggestion.

```
808 \cs_if_free:NT \socket_get_plug:nN
809 {
810 \cs_new_protected_nopar:Npn \socket_get_plug:nN #1#2
811 {
```

*Bug tracker: codeberg.org/cfr/prooftrees/issues | Code: codeberg.org/cfr/prooftrees | Mirror: github.com/cfr42/prooftrees

```

812   \str_set_eq:cN { l__socket_#1_plug_str } #2
813 }
814 }

```

8 ‘Tagging keylists’

A bit like `expl3` property lists outside `forest` environments; just like `forest keylist options` inside them.

Mostly intended for tagging, but possibly useful in some other context so here. Sylwad jps: <https://chat.stackexchange.com/transcript/message/68670752#68670752>.

```

815 \ExplSyntaxOn
816 \tl_new:N \l__forestext_tmpa_tl
817 \prop_new:N \l__forestext_tmpa_prop
818 \seq_new:N \l__forestext_tmpa_seq

```

`forestext_fkeylist_declare:nn` Wrapper.

```

819 \cs_new_protected_nopar:Npn \__forestext_fkeylist_declare:nn #1#2
820 {
821   <debug>      \typeout{[Forest ext.utils debug]:: Declare #1 with #2.}
822   \prop_new:c {l__forestext_#1_prop}
823   \__forestext_fkeylist_put_from_keyval:nn {#1}{#2}
824   <debug>      \__forestext_fkeylist_log:n {#1}
825 }

```

`forestext_fkeylist_redeclare:nn` This one is the point, after all. That is, it is here that `forest` (Živanović 2017) seems to lack capacity (as far as I can tell).

```

826 \cs_new_protected_nopar:Npn \__forestext_fkeylist_redeclare:nn #1#2
827 {
828   <debug>      \typeout{[Forest ext.utils debug]:: Redeclare #1 with #2.}
829   \prop_clear:c {l__forestext_#1_prop}
830   \__forestext_fkeylist_put_from_keyval:nn {#1}{#2}
831   <debug>      \__forestext_fkeylist_log:n {#1}
832 }

```

`__forestext_fkeylist_put_from_keyval:nn` This is ugly as sin, but `!3prop` does not like keys without values.

jps: <- ‘we’ll need two steps of full expansion’ I don’t understand this at all.

jps: <https://chat.stackexchange.com/transcript/message/68672267#68672267> ‘`\exp_args:Nne \prop_set_from_keyval:ce` will in combination expand the entire thing two times inside an e-argument, hence two steps of full expansion. It’s necessary because `\keyval_parse:nnn` returns its result inside `\exp_not:n`, but we want to also expand all the auxiliary functions, hence two steps.’

```

833 \cs_new_protected_nopar:Npn \__forestext_fkeylist_put_from_keyval:nn #1#2
834 {
835   <debug>      \typeout{[Forest ext.utils debug]:: Processing #2 for #1.}
836   \exp_args:Nne \prop_put_from_keyval:ce {l__forestext_#1_prop}
837   {
838     \keyval_parse:NNn
839     \__forestext_fkeylist_put_from_keyval_aux:n
840     \__forestext_fkeylist_put_from_keyval_aux:nn
841     {#2}
842   }
843 }

```

fkeylist_put_from_keyval_aux:n jps. I would never have thought to do it this way?

```

keylist_put_from_keyval_aux:nn
844 \cs_new_nopar:Npn \__foretext_fkeylist_put_from_keyval_aux:n #1
845 {
846   \__foretext_fkeylist_put_from_keyval_aux:nn {#1} {\q_no_value}
847 }
848 \cs_new_nopar:Npn \__foretext_fkeylist_put_from_keyval_aux:nn #1#2
849 {
850   \exp_not:n { {#1} = {#2} },
851 }

```

foretext_fkeylist_tokeyval:n Wrapper.

```

852 \cs_new_nopar:Npn \__foretext_fkeylist_to_keyval:n #1
853 {
854   \prop_map_function:cN {l__foretext_#1_prop} \__foretext_fkeylist_to_keyval_aux:nn
855 }

```

ext_fkeylist_to_keyval_aux:nn Ugly as sin in reverse.

```

856 \cs_new_nopar:Npn \__foretext_fkeylist_to_keyval_aux:nn #1#2
857 {
858   \str_if_eq:nnTF {\q_no_value} {#2}
859   {\exp_not:n{#1},}{\exp_not:n{#1}=\exp_not:n{#2}},}
860 }

```

__foretext_fkeylist_put:nn Wrapper.

```

861 \cs_new_protected_nopar:Npn \__foretext_fkeylist_put:nn #1#2
862 {
863   \__foretext_fkeylist_put_from_keyval:nn {#1} {#2}
864   <debug> \__foretext_fkeylist_log:n {#1}
865 }

```

__foretext_fkeylist_remove:nn ‘<< Unconditionally remove a key.

```

866 \cs_new_protected_nopar:Npn \__foretext_fkeylist_remove:nn #1#2
867 {
868   \prop_remove:cn {l__foretext_#1_prop} {#2}
869   <debug> \__foretext_fkeylist_log:n {#1}
870 }

```

ext_fkeylist_remove_if_match:nn Conditional removal.

```

871 \cs_new_protected_nopar:Npn \__foretext_fkeylist_remove_if_match:nn #1#2
872 {
873   <debug> \typeout{[Forest ext.utils debug]:: Remove #2 from #1 if value match.}
874   \prop_set_eq:Nc \l__foretext_tmpa_prop {l__foretext_#1_prop}
875   \keyval_parse:NNn
876   \__foretext_fkeylist_remove_from_keyval_aux:n
877   \__foretext_fkeylist_remove_from_keyval_aux:nn
878   {#2}
879   \prop_set_eq:cN {l__foretext_#1_prop} \l__foretext_tmpa_prop
880   <debug> \__foretext_fkeylist_log:n {#1}
881 }

```

fkeylist_remove_from_keyval_aux:n Auxiliaries.

```

fkeylist_remove_from_keyval_aux:nn
882 \cs_new_protected_nopar:Npn \__foretext_fkeylist_remove_from_keyval_aux:n #1
883 {
884   \__foretext_fkeylist_remove_from_keyval_aux:nn {#1} {\q_no_value}

```

```

885 }
886 \cs_new_protected_nopar:Npn \__forestext_fkeylist_remove_from_keyval_aux:nn #1#2
887 {
888   <debug> \typeout{[Forest ext.utils debug]:: Remove #1 if value is #2.}
889   \prop_get:NnN \l__forestext_tmpa_prop {#1} \l__forestext_tmpa_tl
890   \tl_if_eq:NnT \l__forestext_tmpa_tl {#2}
891   {
892     \prop_remove:Nn \l__forestext_tmpa_prop {#1}
893   }
894 }

```

\forestext@keylist@declare 2e aliases.

```

\forestext@prop@to@keylist
\forestext@keylist@put
\forestext@keylist@remove@key
\forestext@keylist@remove
\forestext@keylist@redeclare
895 \cs_new_eq:NN \forestext@keylist@declare \__forestext_fkeylist_declare:nn
896 \cs_new_eq:NN \forestext@prop@to@keylist \__forestext_fkeylist_to_keyval:n
897 \cs_new_eq:NN \forestext@keylist@put \__forestext_fkeylist_put:nn
898 \cs_new_eq:NN \forestext@keylist@remove@key \__forestext_fkeylist_remove:nn
899 \cs_new_eq:NN \forestext@keylist@remove \__forestext_fkeylist_remove_if_match:nn
900 \cs_new_eq:NN \forestext@keylist@redeclare \__forestext_fkeylist_redeclare:nn

```

ext_fkeylist_protected_show:n

```

\forestext@keylist@log
901 <debug> \cs_new_nopar:Npn \__forestext_fkeylist_log:n #1
902 <debug> {
903   <debug> \typeout{[tagforext debug]:: #1: }
904   <debug> \prop_log:c {l__forestext_#1_prop}
905   <debug> }
906 <debug> \cs_new_eq:NN \forestext@keylist@log \__forestext_fkeylist_log:n

907 \cs_generate_variant:Nn \prop_to_keyval:N {c}
908 \cs_generate_variant:Nn \prop_put_from_keyval:Nn {ce}
909 \ExplSyntaxOff
910 \newtoks\forestext@toksa

```

Avoid using a hook.

```

911 \forestset{%

```

forestext utils debug (*style*) Debugging.

```

912 forestext utils debug/.style={%
913   typeout={[Forest ext.utils debug]:: #1},
914 },

```

lare tagging keylist (*code key*) Wrappers for primary functionality of these bits.

lare tagging keylist (*code key*)

```

915 declare tagging keylist/.code 2 args={%
916 <debug> \typeout{[Forest ext.utils debug]:: Declaring tagging keylist #1}%
917 <debug> \typeout{[Forest ext.utils debug]:: with default #2.}%
918 \forestext@keylist@declare {#1}{#2}%
919 <debug> \forestext@keylist@log{#1}%
920 \forestext@toksapp\forestext@toksa{%
921   declare keylist/.process={_x{#1}{\forestext@prop@to@keylist{#1}}},
922   }%
923 <debug> \expandafter\typeout\expandafter{\the\forestext@toksa}%
924 \pgfqkeys{/forest}{%
925   forestext utils debug={Setting processing order for #1 to unique=tree.},
926   #1 processing order/.nodewalk style={unique=tree},
927   }%
928 },
929 redeclare tagging keylist/.code 2 args={%
930 <debug> \typeout{[Forest ext.utils debug]:: Redefining tagging keylist #1}%

```

```

931 <debug>          \typeout{[Forest ext.utils debug]:: with default #2.}%
932   \foresttext@keylist@redeclare {#1}{#2}%
933 <debug>          \foresttext@keylist@log{#1}%
934   },

```

tagging keylist put (*code key*) Wrappers for manipulating these keylists.

```

g keylist remove key (code key)
gging keylist remove (code key)
935   tagging keylist put/.code 2 args={%
936     \foresttext@keylist@put {#1}{#2}%
937   },
938   tagging keylist remove key/.code 2 args={%
939     \foresttext@keylist@remove@key {#1}{#2}%
940   },
941   tagging keylist remove/.code 2 args={%
942     \foresttext@keylist@remove {#1}{#2}%
943   },
944 }

```

Declare ‘tagging keylist’ options so we get defaults applied to nodes. Then zap all the user-facing keys used to manipulate them.

We want this to happen really early, but we do need the group or there’s no point.

```

945 \AddToHook{cmd/forest@config/before}{.}%
946 <debug>   \typeout{[Forest ext.utils debug]:: Creating options set with declare tagging
          keylist.}%
947 <debug>   \expandafter\typeout\expandafter{\the\foresttext@toksa}%
948   \expandafter\forestset\expandafter{\the\foresttext@toksa}%
949   \forestset{%
950     tagging keylist error/.code={%
951       \PackageError{ext.tagging (forest library)}{%
952         The key '#1' cannot be used inside a forest environment.%
953       }{%
954         You need to use this key inside the document but outside the forest.
955         Please see forest-ext's documentation for details.%
956       }%
957     },
958     declare tagging keylist/.style 2 args={%
959       tagging keylist error=declare tagging keylist},
960     redeclare tagging keylist/.style 2 args={%
961       tagging keylist error=redeclare tagging keylist},
962     tagging keylist put/.style 2 args={%
963       tagging keylist error=tagging keylist put},
964     tagging keylist remove/.style 2 args={%
965       tagging keylist error=tagging keylist remove},
966     tagging keylist remove key/.style 2 args={%
967       tagging keylist error=tagging keylist remove key},
968   }%
969 }

```

9 Styles

```

970 \forestset{

```

align middle child (*style*) Based on T_EX SE answer: [436985](#). Based on T_EX SE question [436881](#) by A. D.

```

align middle children (style)
971   align middle child/.style={
972     before typesetting nodes={
973       if={
974         > 0w+P {n children}{isodd(##1)}

```

```

975     }{
976         calign child/.process={
977             0w+n {n children}{(##1+1)/2}
978         },
979         calign=#1,
980     }{,
981 },
982 },
983 align middle child/.default=child edge,
984 align middle children/.style={
985     for tree={align middle child=#1},
986 },
987 align middle children/.default=child edge,

```

`utils@outer@label@opts` (*keylist*) Options.

```

988 declare keylist={utils@outer@label@opts}{},

```

`outer labels` (*keylist reg.*) Keys applied to all outer labels.

```

989 declare keylist register={outer labels},
990 outer labels={anchor=base west},

```

`utils@has@outer@labels` (*bool. reg.*) Boolean.

```

991 declare boolean register=utils@has@outer@labels,
992 utils@has@outer@labels=0,

```

`outer labels at` (*toks reg.*) Anchor.

```

993 declare toks register=outer labels at,
994 outer labels at=east,

```

`utils@outer@label` (*auto. toks*) Label.

```

995 declare autowrapped toks={utils@outer@label}{},

996 /forest/ext/utils/outer@label/anchor/.initial=base,
997 /forest/ext/utils/outer@label/anchor/.forward to=/tikz/anchor,
998 /forest/ext/utils/outer@label/.search also={/tikz,/pgf},

```

`outer label` (*style*) Args: options, content

```

999 outer label/.style={%
1000     split={#1}{:}{utils@outer@label,utils@outer@label@opts},
1001     if utils@has@outer@labels={}{%
1002         utils@has@outer@labels,
1003         for root={%
1004             tikz+={%
1005                 \coordinate (utils@outer@labels@align) at
1006                     (current bounding box.\forestregister{outer labels at});
1007             },
1008             before drawing tree={%
1009                 where utils@outer@label={}{:}{%
1010                     tikz+/.process={%
1011                         000Rw4
1012                         {utils@outer@label}
1013                         {utils@outer@label@opts}
1014                         {!u.grow}
1015                         {outer labels}
1016                         {%

```

```

1017         \path [%
1018             rotate=##3,
1019         ] node [%
1020             ##4,
1021             /forest/ext/utls/outer@label/.cd,
1022             ##2,
1023         ] at (.\pgfkeysvalueof{/forest/ext/utls/outer@label/anchor}
1024             |- utls@outer@labels@align)
1025             {##1};
1026         }%
1027     },
1028 },
1029 },
1030 },
1031 },
1032 },

1033 <!debug>     libraries/ext.utls/defaults/.style=
1034 <debug>     libraries/ext.utls-debug/defaults/.style=
1035     {}%
1036 }

</sty>

```


References

- Fischer, Ulrike (2025). *The `tagpdf` Package*. v0.99w. 31st Oct. 2025. CTAN: [tagpdf](#).
- International Organization for Standardization (2025). *Document management applications — Electronic document file format enhancement for accessibility — Part 2: Use of ISO 32000-2 (PDF/UA-2)*. 5th Apr. 2025.
- PDF Association (2024a). *ISO 32000-2:2020 (PDF 2.0) including Errata Collection 2*. 24th Sept. 2024.
- (2024b). *Well-Tagged PDF (WTPDF) Using Tagged PDF for Accessibility and Reuse in PDF 2.0*. 28th Feb. 2024.
- L^AT_EX Project (2025a). *latex-lab*. 2025-11-01a. 1st Nov. 2025. CTAN: [latex-lab](#).
- (2025b). *The latex-lab-tikz Package: Support for the Tagging of TikZ Pictures*. v0.80d. 27th Sept. 2025. CTAN: [latex-lab](#).
- Rees, Clea F. (2026). *prooftrees*. 0.9.2. 16th Jan. 2026. CTAN: [prooftrees](#).
- Živanović, Sašo (2017). *Forest: A PGF/TikZ-Based Package for Drawing Linguistic Trees*. 2.1.5. 14th July 2017. CTAN: [forest](#).

Change History

v0.1

General: First public release. 1

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.

Symbols	
+also parent (style)	<i>15</i> , <i>226</i>
\@ifpackageloaded	<i>804</i>
\@tagforest@debugfalse	<i>596</i>
\@tagforest@debugtrue	<i>596</i>
_	<i>704</i> , <i>719</i> , <i>723</i> , <i>728</i> , <i>735</i> , <i>739</i>
__foresttext_fkeylist_declare:nn	<i>819</i> , <i>895</i>
__foresttext_fkeylist_log:n	<i>824</i> , <i>831</i> , <i>864</i> , <i>869</i> , <i>880</i> , <i>901</i> , <i>906</i>
__foresttext_fkeylist_protected_show:n	<i>901</i>
__foresttext_fkeylist_put:nn	<i>861</i> , <i>897</i>
__foresttext_fkeylist_put_from_keyval:nn	<i>823</i> , <i>830</i> , <i>833</i> , <i>863</i>
__foresttext_fkeylist_put_from_keyval_aux:n	<i>839</i> , <i>844</i>
__foresttext_fkeylist_put_from_keyval_aux:nn	<i>840</i> , <i>844</i>
__foresttext_fkeylist_redeclare:nn	<i>826</i> , <i>900</i>
__foresttext_fkeylist_remove:nn	<i>866</i> , <i>898</i>
__foresttext_fkeylist_remove_from_keyval_aux:n	<i>876</i> , <i>882</i>
__foresttext_fkeylist_remove_from_keyval_aux:nn	<i>877</i> , <i>882</i>
__foresttext_fkeylist_remove_if_match:nn	<i>871</i> , <i>899</i>
__foresttext_fkeylist_to_keyval:n	<i>852</i> , <i>896</i>
__foresttext_fkeylist_to_keyval_aux:nn	<i>854</i> , <i>856</i>
__foresttext_fkeylist_tokeyval:n	<i>852</i>
__tagforest_end:	<i>490</i> , <i>545</i> , <i>576</i> , <i>579</i>
__tagforest_init:	<i>478</i> , <i>575</i> , <i>583</i>
__tagforest_noop:n	<i>425</i> , <i>441</i> , <i>442</i> , <i>490</i>
__tagforest_pgftikz_tag_bbox:enn	<i>396</i> , <i>464</i>
__tagforest_pgftikz_tag_bbox:nnn	<i>396</i>
__tagforest_pgftikz_tag_bbox_aux:enn	<i>398</i> , <i>408</i>
__tagforest_pgftikz_tag_bbox_aux:nnnn	<i>408</i>
__tagforest_tag_resume:n	<i>423</i> , <i>546</i> , <i>548</i> , <i>565</i>
__tagforest_tag_suspend:n	<i>423</i> , <i>443</i> , <i>547</i> , <i>572</i>
__tagforest_tag_tree_tag:nnn	<i>556</i>
A	
add parent (style)	<i>275</i>
\AddToHook	<i>308</i> , <i>945</i>
\advance	<i>480</i>
align middle child (style)	<i>18</i> , <i>971</i>
align middle children (style)	<i>18</i> , <i>971</i>
also parent (style)	<i>15</i> , <i>226</i>
also parent+ (style)	<i>15</i> , <i>226</i>
alt text (autowrapped toks)	<i>4</i>
alttext (autowrapped toks)	<i>649</i>
\AssignSocketPlug	<i>617</i> , <i>630</i>
autowrapped toks registers:	
has branches	<i>5</i> , <i>651</i>
is branch	<i>5</i> , <i>651</i>
is child	<i>5</i> , <i>651</i>
is edge label	<i>5</i> , <i>651</i>
is leaf	<i>5</i> , <i>651</i>

is root	5, 651
autowrapped toks:	
alt text	4
alttext	649
node@ttoks	649
utils@outer@label	995

B

before collating tags (keylist)	4, 646
before tagging nodes (keylist)	4, 646
before tagging tree (keylist)	4, 646
\begin	154, 158
\bool_if:NTF	487
\bool_new:N	385
\bool_set_false:N	386, 393
boolean registers:	
debug multi phantoms	15
not debug multi phantoms	15
tagging	5
utils@has@outer@labels	991

C

c foster parent (step)	15, 77
c fosterling (step)	15, 77
\c_space_tl	411, 413 , 415
choice keys:	
collate tags uses	5, 749
tag nodes uses	5, 695
tag tree uses	5, 762
code keys:	
collate tag	758
custom tagging	7, 385
declare tagging keylist	915
declare tagging keylist,reddeclare tagging keylist	20
not custom tagging	7
redeclare tagging keylist	915
tagging keylist put	21, 935
tagging keylist remove	21, 935
tagging keylist remove key	21, 935
collate tag (code key)	758
collate tags (tagging keylist)	4, 644
collate tags uses (choice key)	5, 749
\coordinate	1005
\cs_generate_variant:Nn	407, 418 , 907 , 908
\cs_if_free:NT	808
\cs_new:cpe	462
\cs_new:Npn	803
\cs_new_eq:NN	423, 424 , 574, 575, 576, 895, 896, 897, 898, 899, 900, 906
\cs_new_nopar:Npn	396, 408, 425, 478, 545, 556, 844, 848, 852, 856, 901
\cs_new_protected_nopar:Npn	810, 819, 826, 833, 861, 866, 871, 882, 886
\cs_set_eq:NN	441, 442, 490, 547, 548
custom tagging (code key)	7, 385

D

debug multi phantoms (bool. reg.)	15
---	----

debug multi phantoms (style)	288
debug@multi (style)	275
debug@multi@option (style)	275
debug@multi@register (style)	275
declare tagging keylist (code key)	915
declare tagging keylist,reddeclare tagging keylist (code key)	20
\def	536, 537
\detokenize	276, 279, 284, 601
\dim_to_decimal_in_bp:n	410, 412, 414, 416
\disable@package@load	6, 7, 40, 41, 366, 367, 788, 789

E

\else	432
\end	153
every foster parent (step)	15, 77
every fosterling (step)	15, 77
every parent (keylist)	14, 55
\exp_args:Ne	504, 523
\exp_args:Nne	836
\exp_args:NnV	497, 516
\exp_args:No	563
\exp_not:N	504, 523
\exp_not:n	850, 859
\expandafter	588, 589, 593, 803, 923, 947, 948
\ExpandArgs	276, 279, 284, 601
\expanded	588, 593
\ExplSyntaxOff	585, 909
\ExplSyntaxOn	382, 802, 815
ext.ling (lib.)	3
ext.ling-debug (lib.)	3
ext.multi (lib.)	3
ext.multi-debug (lib.)	3
ext.tagging (lib.)	3, 3
ext.tagging-debug (lib.)	3, 3
ext.utils (lib.)	3
ext.utils-debug (lib.)	3

F

\fi	434, 551, 569, 602
\foresteoption	285
\foresteregister	165, 173, 280, 493, 513, 1006
forestext utils debug (style)	912
\forestext@keylist@declare	895, 918
\forestext@keylist@log	901, 919, 933
\forestext@keylist@put	895, 936
\forestext@keylist@redeclare	895, 932
\forestext@keylist@remove	895, 942
\forestext@keylist@remove@key	895, 939
\forestext@prop@to@keylist	895, 921
\forestext@toksa	910, 920, 923, 947, 948
\forestext@toksapp	760, 803, 920
\forestset	20, 54, 314, 387, 447, 483, 504, 510, 523, 529, 540, 604, 911, 948, 949, 970
foster parents (step)	15, 77
fosterlings (step)	15, 77

G

\global	480
---------------	-----

H		N	
has branches (autowrapped toks reg.)	5, <u>651</u>	\newcommand	587, 592, 599
\hook_gput_code:nnn	577, <u>581</u>	\newcount	591
		\newif	596, 806
I		\newtoks	586, 910
\if@inlabel	430	node@ttoks (autowrapped toks)	649
\if@tagforest@debug	549, <u>596</u>	\noexpand	589
\ifmemoizing	567, 806	not custom tagging (code key)	7
\IfPackageLoadedT	309	not debug multi phantoms (bool. reg.)	15
\IfSocketPlugExistsTF	616, 629	not debug multi phantoms (style)	288
\interval	163, 170		
is branch (autowrapped toks reg.)	5, <u>651</u>	O	
is child (autowrapped toks reg.)	5, <u>651</u>	other parents (keylist)	55
is edge label (autowrapped toks reg.)	5, <u>651</u>	outer label (style)	19, <u>999</u>
is leaf (autowrapped toks reg.)	5, <u>651</u>	outer labels (keylist register)	19, <u>989</u>
is root (autowrapped toks reg.)	5, <u>651</u>	outer labels at (toks register)	19, <u>993</u>
K		P	
keylists registers:		\PackageError	619, 632, 951
outer labels	19, <u>989</u>	\PackageInfo	310, 311, 351, 352, 500, 519
keylists:		\PackageWarning	9,
before collating tags	4, <u>646</u>	10, 43, 44, 289, 297, 369, 370, 506, 525, 791, 792	
before tagging nodes	4, <u>646</u>	\path	1017
before tagging tree	4, <u>646</u>	\pgfkeysvalueof	1023
every parent	14, 55	\pgfqkeys	924
other parents	55	\pgfsys@begin@text	536
utils@outer@label@opts	988	\pgfsys@end@text	537
\keyval_parse:NNn	838, 875	plugins:	
		tagsupport/forest/innitagg	426
		tagsupport/forest/setup alt	445
		tagsupport/forest/tag alt	454
		pretty nice empty nodes (style)	17, 20
		\prop_clear:c	829
		\prop_get:NnN	889
		\prop_log:c	904
		\prop_map_function:cN	854
		\prop_new:c	822
		\prop_new:N	817
		\prop_put_from_keyval:ce	836
		\prop_put_from_keyval:Nn	908
		\prop_remove:cn	868
		\prop_remove:Nn	892
		\prop_set_eq:cN	879
		\prop_set_eq:Nc	874
		\prop_to_keyval:N	907
		\property_record:ee	559
		\property_ref:ee	400, 403
		\ProvidesForestLibrary	3, 4, 37, 38, 363, 364, 785, 786
		Q	
		\q_no_value	846, 858, 884
		R	
		redeclare tagging keylist (code key)	915
		\relax	480
		S	
		\seq_new:N	818
		\ShowTagging	550
		\socket_assign_plug:nn	437, 438, 439, 533, 566, 568
		\socket_get_plug:nN	496, <u>808</u>
M			
\memoizingfalse	806		
\mode_if_vertical:T	428		
\mode_leave_vertical:	431		
multi (style)	13, <u>99</u>		
multi@add@parent (style)	188		
multi@also@parent (style)	125		
multi@forked@edge (style)	315		
multi@parent (style)	136		
multi@phantom (style)	260		

\socket_if_plug_exist:nnTF	497, 516	\tag_if_active:TF	481
\socket_new:nn	419, 420, 421, 422	\tag_mc_begin:n	461
\socket_new_plug:nnn	426, 445, 454	\tag_mc_begin_pop:n	554
\socket_use:n	440, 535	\tag_mc_end:	552
\socket_use:nnn	570, 571	\tag_mc_end_push:	436
sockets:		\tag_resume:n	424, 442, 548
tagsupport/forest/init	419	\tag_socket_use:n	433
tagsupport/forest/setup	419	\tag_struct_begin:n	456
tagsupport/forest/tag	419	\tag_struct_end:	553
tagsupport/forest/tag/mmz	419	\tag_struct_gput:ene	467
stages:		\tag_suspend:n	423, 441, 547
tag tree stage	4	\tagforest@debug@typeout	489, 492, 495, 515, 532, 534, 596, 606, 759
steps:		\tagforest@end	576
c foster parent	15, 77	\tagforest@id	462, 464, 474, 480, 559, 591
c fosterling	15, 77	\tagforest@init	575
every foster parent	15, 77	\tagforest@tag@tree@tag	556, 778
every fosterling	15, 77	\tagforest@toks	563, 586, 760
foster parents	15, 77	tagging (bool. reg.)	5
fosterlings	15, 77	tagging keylist put (code key)	21, 935
\str_if_eq:eeT	493, 513	tagging keylist remove (code key)	21, 935
\str_if_eq:nnTF	858	tagging keylist remove key (code key)	21, 935
\str_new:N	384	tagging keylists:	
\str_set_eq:cN	812	collate tags	4, 644
styles:		tag nodes	4, 644
+also parent	15, 226	tagsupport/forest/init (socket)	419
add parent	275	tagsupport/forest/inittag (plug)	426
align middle child	18, 971	tagsupport/forest/setup (socket)	419
align middle children	18, 971	tagsupport/forest/setup alt (plug)	445
also parent	15, 226	tagsupport/forest/tag (socket)	419
also parent+	15, 226	tagsupport/forest/tag alt (plug)	454
debug multi phantoms	288	tagsupport/forest/tag/mmz (socket)	419
debug@multi	275	\tex_savepos:D	558, 561
debug@multi@option	275	\text_purify:n	563
debug@multi@register	275	\the	462, 464, 474, 559, 563, 589, 594, 803, 923, 947, 948
forestext utils debug	912	\tl_if_eq:NnT	890
multi	13, 99	\tl_new:N	383, 816
multi@add@parent	188	\tl_set:Ne	562
multi@also@parent	125	toks registers:	
multi@forked@edge	315	outer labels at	19, 993
multi@parent	136	\typeout	276, 279, 284, 588, 593, 601, 821, 828, 835, 873, 888, 903, 916, 917, 923, 930, 931, 946, 947
multi@phantom	260		
not debug multi phantoms	288		
outer label	19, 999		
pretty nice empty nodes	17, 20		

T

tag nodes (tagging keylist)	4, 644
tag nodes uses (choice key)	5, 695
tag tree stage (stage)	4
tag tree uses (choice key)	5, 762
\tag_get:n	468

U

\use:c	389, 473
\useforestlibrary	380, 381
\usetikzlibrary	313
utils@has@outer@labels (bool. reg.)	991
utils@outer@label (autowrapped toks)	995
utils@outer@label@opts (keylist)	988