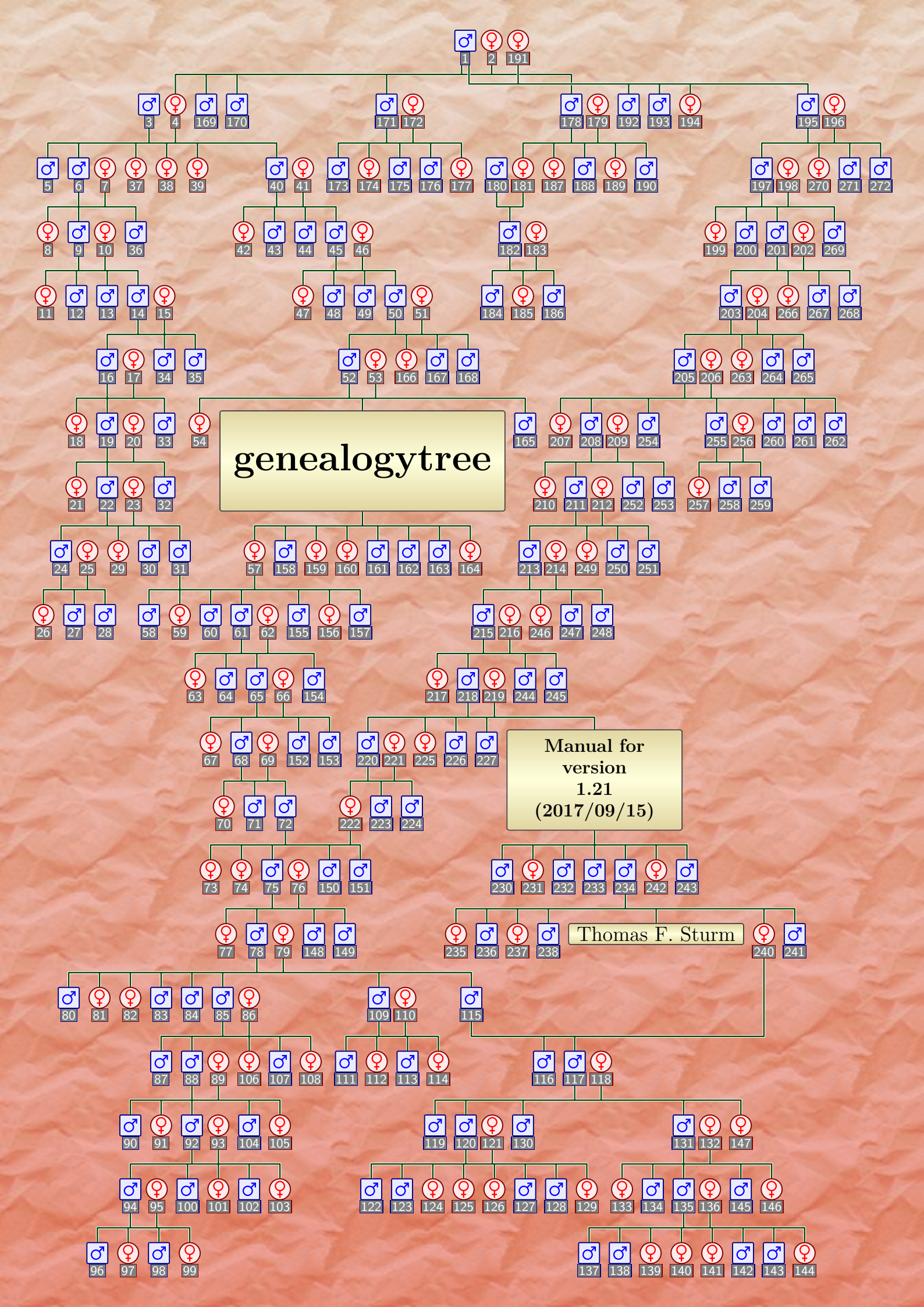

genealogytree
Manual for version 1.21 (2017/09/15)
Thomas F. Sturm

Cover code

```
% \usepackage{incgraph}
% \usetikzlibrary{backgrounds}
\begin{inctext}%
\gtrSymbolsSetCreateSelected{blue}{Male}\gtrSymbolsSetCreateSelected{red}{Female}%
\gtrSymbolsSetCreateSelected{black}{Neuter}%
\begin{tikzpicture}
\genealogytree[template=symbol nodes,level size=8mm,level distance=6.5mm,
  box={title={\gtrnodenumber},height=5mm,attach boxed title to bottom center,
    fonttitle=\fontsize{3mm}{3mm}\sffamily,
    boxed title style={size=tight,height=3mm,valign=center,left=0.2mm,right=0.2mm}},
  tcbset={mytext/.style={notitle,interior style={top color=gtr_Yellow_Frame!50!gtr_Yellow_Back,
    bottom color=gtr_Yellow_Frame!50!gtr_Yellow_Back,middle color=gtr_Yellow_Back}}},
  add parent=a to fam_a,add parent=b to fam_b,add parent=c to fam_c]
{ child{ gm pf
    child{ pm gf cm child{ gm pf cf child{ gm pf cf cm cm child{ gm pf
        child{ gm pf cf child{ gm pf cf child{ gm pf child{ gm pf cf cm cm } cf cm c[id=a]m } cm
          } cm } cm cm } } cm } cf cf cf
      child{ gm pf cf cm cm child{ gm pf cf cm cm child{ gm pf child{ gm pf cf
        child{ g[box={width=6.5cm,notitle},phantom*]m
          child{g[box={width=6.5cm,height=23mm,enlarge top initially by=-15mm,mytext},
            no content interpreter]{\Huge\bfseries genealogytree}
          child[id=fam_a] {gf cm cf cm
            child{ gm pf cf cm child{ gm pf cf child{ gm pf cf cm
              child[id=fam_b]{ gm cf cf child{ gm pf cf
                child{ gm pf cm cf cf cm cm
                  child{ gm pf cm child{ gm pf cm cf child{ gm pf child{ gm pf cm cf cm cf }
                    cm cf cm cf } cm cf } cf cm cf }
                  child{ gm pf cm cf cm cf }
                  child[id=fam_c,pivot shift=2cm]{ g[distance=1cm]m cm child { gm pf cm
                    child{ gm pf cm cm cf cf cf cm cm cf } cm
                    child{ gm pf cf cm child{ gm pf cm cm cf cf cf cm cm cf } cm cf }
                    cf } } }
                cm cm } cm cm } } cm cm } cm } cm cf cm
          } cm cf cf cm cm cm cf }
        } cm } cf cm cm } }
      }
    } cm cm
    child{ gm pf cm cf cm cm cf }
    child{ gm pf child{ pm gf child{ gm pf cm cf cm } } cf cm cf cm }
    union{ pf cm cm cf
      child{ gm pf child{ gm pf cf cm child{ gm pf child{ gm pf
        child{ gm pf cf child{ gm pf cf child{ gm pf child{ gm pf
          child{ gm pf cf child{ gm pf child{ gm pf c[id=b]f cm cm } cf cm cm
          child{ g[box={width=4cm,notitle},phantom*]m
            child{g[box={width=4cm,height=23mm,enlarge top initially by=-15mm,
            mytext,capture=minipage,halign=center},no content interpreter]
            {\large\bfseries Manual for\\ version\\ \version\\(\datum)}
              cm cf cm cm child{ gm cf cm cf cm
              child{ g[box={width=4cm,mytext},no content interpreter]{Thomas F.~Sturm} }
              c[id=c]f cm } cf cm }
        } } cm cm } cf cm cm } cf cm cm } cm cm }
          cm child{ gm pf cf cm cm } cm cm cm } cf cm cm } cf cm cm } cm
      } cf cm cm
    }
  }
} }
\begin{scope}[on background layer]
  \node (bg) [fill tile image*={width=4cm}{crinklepaper.png},minimum width=21cm,
    minimum height=29.7cm,inner sep=0pt,outer sep=0pt] at (current bounding box) {};
  \path[top color=white,bottom color=red,fill opacity=0.25]
    (bg.south west) rectangle (bg.north east);
\end{scope}
\end{tikzpicture}
\end{inctext}
```

The genealogytree package

Manual for version 1.21 (2017/09/15)

Thomas F. Sturm[1]

http://www.ctan.org/pkg/genealogytree

https://github.com/T-F-S/genealogytree

Abstract

Pedigree and genealogy tree diagrams are proven tools to visualize genetic and relational connections between individuals. The naming for mathematical tree structures with parent nodes and child nodes is traded from historical family diagrams. However, even the smallest family entity consisting of two parents and several children is no mathematical tree but a more general graph.
The `genealogytree` package provides a set of tools to typeset such genealogy trees or, more precisely, to typeset a set of special graphs for the description of family-like structures. The package uses an auto-layout algorithm which can be customized to e.g. prioritize certain paths.

Contents

1 Introduction **9**
- 1.1 Genealogy Trees . . . 9
- 1.2 Package Design Principles and Philosophy . . . 10
- 1.3 Comparison with Other Packages . . . 11
- 1.4 Installation . . . 12
- 1.5 Loading the Package . . . 12
- 1.6 Libraries . . . 13
- 1.7 How to Get Started . . . 13

2 Tutorials **15**
- 2.1 Tutorial: First Steps (Ancestor Tree) . . . 15
  - 2.1.1 Document Setup . . . 15
  - 2.1.2 Creation of a Basic Ancestor Diagram . . . 16
  - 2.1.3 Applying options . . . 18
  - 2.1.4 Growing the Tree . . . 19
  - 2.1.5 Prioritize and Colorize a Path . . . 22
  - 2.1.6 Changing the Timeflow . . . 24
- 2.2 Tutorial: Diagram Manipulation by ID values (Descendant Tree) . . . 25
  - 2.2.1 Creation of a Basic Descendant Diagram . . . 25
  - 2.2.2 Growing the Tree . . . 26
  - 2.2.3 Separating Diagram Data and Diagram Drawing . . . 29
  - 2.2.4 Emphasizing a Relationship Path . . . 30
  - 2.2.5 Coloring Subtrees . . . 32
- 2.3 Tutorial: A Database Family Diagram (Sand Clock) . . . 34
  - 2.3.1 Creation of a Basic Sand Clock Diagram . . . 34
  - 2.3.2 Node Content in Database Format . . . 35

[1]Prof. Dr. Dr. Thomas F. Sturm, Institut für Mathematik und Informatik, Universität der Bundeswehr München, D-85577 Neubiberg, Germany; email: thomas.sturm@unibw.de

2.3.3 Formatting the Node Content . . . 36
2.3.4 Adding Images . . . 38
2.3.5 Full Example with Frame . . . 39
2.4 Tutorial: Descendants of the Grandparents (Connecting Trees) . . . 43
2.4.1 Descendants of the Two Grandparents . . . 43
2.4.2 Connected Diagram . . . 44
2.5 Tutorial: Multi-Ancestors . . . 46
2.5.1 Triple Ancestor Example . . . 46
2.5.2 Adding Edges Manually . . . 47
2.5.3 Manual Position Adjustments . . . 48
2.6 Tutorial: Externalization . . . 49
2.6.1 Externalization Process . . . 49
2.6.2 Document Setup . . . 49
2.6.3 Marking Diagrams for Externalization . . . 50
2.7 Tutorial: Conversion to Pixel Images . . . 52
2.7.1 Command Line Conversion with Ghostscript . . . 52
2.7.2 Command Line Conversion with ImageMagick . . . 53
2.7.3 Conversion with the 'standalone' Package . . . 53
2.7.4 Conversion during Externalization . . . 54

3 Genealogy Tree Macros 55
3.1 Creating a Genealogy Tree . . . 55
3.2 Using Tree Options . . . 58
3.3 Accessing Information inside Nodes . . . 59

4 Graph Grammar 61
4.1 Graph Structure . . . 61
4.2 Subgraph 'parent' . . . 63
4.3 Subgraph 'child' . . . 65
4.4 Subgraph 'union' . . . 67
4.5 Subgraph 'sandclock' . . . 69
4.6 Node 'c' . . . 71
4.7 Node 'p' . . . 71
4.8 Node 'g' . . . 71
4.9 Data 'input' . . . 72
4.10 Control Sequence 'insert' . . . 73

5 Option Setting 75
5.1 Option Priorities . . . 75
5.1.1 Option Priorities for Nodes . . . 76
5.1.2 Option Priorities for Families . . . 77
5.2 Graph Growth Setting (Time Flow) . . . 78
5.3 Graph Geometry . . . 81
5.4 Identifiers . . . 90
5.5 Node Options . . . 93
5.6 Family Options . . . 102
5.7 Subtree Options . . . 105
5.8 Level Options . . . 107
5.9 Tree Positioning Options . . . 109
5.10 TikZ and Tcolorbox Options . . . 112
5.11 Ignoring Input . . . 115
5.12 Inserting Input . . . 118

5.13 Phantom Nodes and Subtrees . . . 122
5.14 Special and Auxiliary Options . . . 125

6 Node Data (Content) Processing 127
6.1 Setting a Node Data Processing and Processor . . . 128
6.2 Predefined Non-Interpreting Processings . . . 129
6.2.1 fit . . . 129
6.2.2 tcolorbox . . . 133
6.2.3 tcbox . . . 136
6.2.4 tcbox* . . . 139
6.2.5 tikznode . . . 142
6.3 Creating a Customized Non-Interpreting Processor . . . 144
6.4 Content Interpreters . . . 145

7 Database Processing 151
7.1 Database Concept . . . 152
7.2 Example Settings . . . 153
7.3 Data Keys . . . 155
7.4 Input Format for Dates . . . 160
7.5 Formatting the Node Data . . . 162
7.6 Formatting Names . . . 172
7.7 Formatting Dates . . . 174
7.8 Formatting Places . . . 178
7.9 Formatting Events . . . 179
7.10 Formatting Lists of Events . . . 181
7.11 Formatting Comments . . . 183
7.12 Formatting Professions . . . 184
7.13 Formatting Lists of Information . . . 185
7.14 Formatting Sex . . . 186
7.15 Formatting Images . . . 187

8 Edges 189
8.1 Edge Settings . . . 190
8.2 Edge Types . . . 194
8.3 Edge Parameters . . . 199
8.4 Edge Labels . . . 203
8.5 Edge Labels Database . . . 205
8.6 Adding and Removing Nodes from Edge Drawing . . . 207
8.7 Extra Edges . . . 212

9 Genealogy Symbols 217
9.1 Symbol Color Settings . . . 217
9.1.1 Global Color Settings . . . 217
9.1.2 Local Color Settings . . . 218
9.2 List of Symbols . . . 219
9.3 Legend to Symbols . . . 222
9.3.1 Printing a Legend . . . 222
9.3.2 Description Texts and Language Settings . . . 223

10 Language and Text Settings 225
10.1 Preamble Settings . . . 225
10.2 Document Settings . . . 226

11 Debugging: Library debug **227**
11.1 Parser Debugging . . . 227
11.2 Processor Debugging . . . 230
11.3 Graphical Debugging . . . 239
11.4 Show Information . . . 243

12 Templates: Library templates **247**
12.1 Using Templates . . . 247
12.2 Template 'formal graph' . . . 247
12.3 Template 'signpost' . . . 248
12.4 Template 'symbol nodes' . . . 248
12.5 Template 'tiny boxes' . . . 249
12.6 Template 'tiny circles' . . . 249
12.7 Template 'directory' . . . 250
12.8 Template 'database pole' . . . 251
12.9 Template 'database pole reduced' . . . 253
12.10 Template 'database poleportrait' . . . 254
12.11 Template 'database poleportrait reduced' . . . 256
12.12 Template 'database portrait' . . . 257
12.13 Template 'database portrait reduced' . . . 259
12.14 Template 'database traditional' . . . 260
12.15 Template 'database traditional reduced' . . . 261
12.16 Template 'database sideways' . . . 262
12.17 Template 'database sideways reduced' . . . 264
12.18 Template 'database sidewaysportrait' . . . 265
12.19 Template 'database sidewaysportrait reduced' . . . 266
12.20 Template 'database relationship' . . . 267
12.21 Template 'ahnentafel 3' . . . 268
12.22 Template 'ahnentafel 4' . . . 270
12.23 Template 'ahnentafel 5' . . . 272
12.24 Predefined Colors of the Library . . . 274
12.25 Auxiliary Control Sequences . . . 274

13 Auto-Layout Algorithm **277**
13.1 Preliminaries . . . 277
13.1.1 Aesthetic Properties . . . 277
13.1.2 Genealogy Trees . . . 278
13.1.3 Graph Grammar . . . 278
13.2 Requirements . . . 279
13.2.1 Parent and Child Alignment . . . 279
13.2.2 Patchwork Families . . . 280
13.2.3 Graph Growing Direction . . . 281
13.3 Algorithmic Steps . . . 282
13.3.1 Recursive Family and Node Placement . . . 282
13.3.2 Contours . . . 282
13.3.3 Combining Subtrees . . . 283
13.4 Known Problems . . . 284

14 Example Graph Files **287**
14.1 example.option.graph . . . 287
14.2 example.database.graph . . . 288
14.3 example.formal.graph . . . 289

15 Stack Macros **291**
15.1 Creating a Stack . . . 291
15.2 Push to a Stack . . . 291
15.3 Pop from a Stack . . . 292
15.4 Peek into a Stack . . . 292
15.5 Creating Stack Shortcuts . . . 293

16 Version History **295**

Bibliography **299**

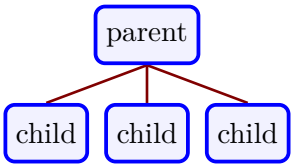
Index **301**

1

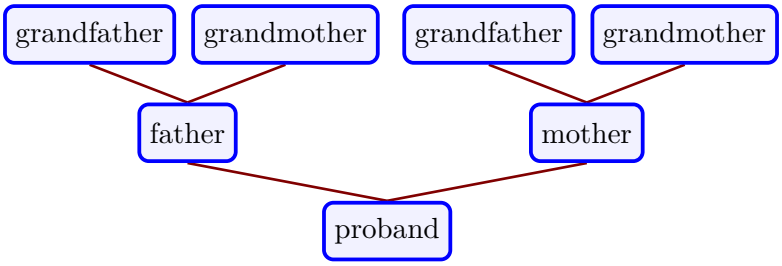
Introduction

1.1 Genealogy Trees

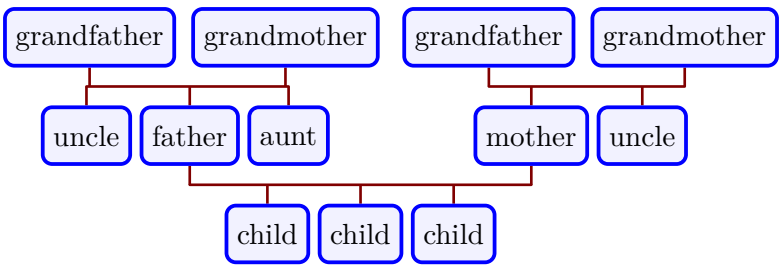
The naming for mathematical tree structures with parent nodes and child nodes is traded from historical family diagrams. But, creating a family diagram for medical and sociological studies or family research can become surprisingly difficult with existing tools for tree visualization. The simple reason is, that a mathematical tree has only *one* parent node for its direct children nodes.



With reverse logic, this can be used to visualize ancestor diagrams starting from an individual to its predecessors:



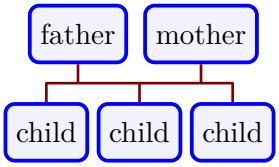
However, even the smallest *family* entity consisting of two parents and several children is no mathematical tree but a more general graph:



The `genealogytree` package aims to support such graphs which will be denoted *genealogy trees* in the following. The graphs to be processed cannot become arbitrarily complex. A set of special graphs for the description of family-like structures is supported by the package algorithms. From at theoretical point of view, these graphs can be seen as a sort of annotated mathematical trees.

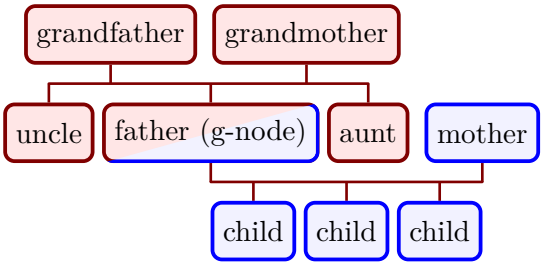
1.2 Package Design Principles and Philosophy

The emphasis of a *genealogy tree* is not the node or individual, but the *family*. A family is a set of arbitrarily many parents and children. From an algorithmic point of view, there could be more than two parents in a family.



A single family

A node is either a parent or a child to a specific family. A node can also be child to one family and parent to another (or many) families. Such a node is called a g-node (genealogy node) in the following.



Two families

The main restriction of the graph grammar is that there is exactly *one* g-node which connects its enclosing family to another one. In the example above, the father node is the g-node in the grandparents family. It is linked to the family with mother and children.

A strong driving force for elaborating and creating this package was to balance two contradictory goals for diagram generation: **automatism** and **customization**. In the ideal case, a diagram would be constructed automatically by data provided from an external data source and would also be freely customizable in any thinkable way starting changing colors, lines, shapes, node positioning, etc. In the real world, a trade-off between these goals has to be found.

Automatism:

- For a set of *genealogy trees* described by a grammar, see Chapter 4 on page 61, an auto-layout algorithms computes the node positioning.
- The graph grammar is family-centric and supports ancestors and descendants diagrams. For the later, multiple marriages can be used to a certain degree.
- The graph data can be written manually, but the package design is adjusted to process automatically generated data. There are many genealogy programs which manage family related data. The general idea is that such programs export selected diagram data in a text file using the provided grammar. Processing GEDCOM[1] files directly by the package is not possible.
- While manipulations like coloring a single node can be done directly at node definition, the package design makes a lot of efforts to allow manipulations *aside* from the actual place of data definition, see Section 5.1.1 on page 76 and Section 5.1.2 on page 77. The idea is that automatically generated data has not to be edited, but can be manipulated from outside. Also, an automatically or manually generated data set can be used for several distinct diagrams; e.g. the graph data in Section 14.1 on page 287 is used numerous times inside this document for different diagrams.

[1] GEDCOM (GEnealogical Data COMmunication) is a widely used data exchange format.

- The auto-layout algorithm is implemented in pure TEX/LATEX (without Lua). This imposes all programming restrictions of this macro language on the implementation, but makes the package independent of external tools and fosters LATEX customization.

Customization:

- The auto-layout algorithm can be customized to e.g. prioritize certain paths or exclude selected subtrees. Also, several node dimensions and distances can be changed globally or locally.
- The appearance of a node can be customized with all capabilities of Ti*k*Z [4] and `tcolorbox` [3]. Also, the node text can be processed.
- For the node content, a database concept can be used, see Chapter 7 on page 151. This gives a high degree of customizing the data visualization inside the node.
- The geometry of edges between nodes is not considered by the auto-layout algorithm, but edges can also be customized in many ways, see Chapter 8 on page 189.
- Several *genealogy tree* diagrams can be interconnected manually to form a more complex total diagram.

On the technical side, the package is based on *The TikZ and PGF Packages* [4] and uses *The tcolorbox package* [3] for the nodes. Since all processing is done in TEX/LATEX without Lua and external tools, one should expect a lot of processing time for complex diagrams. Therefore, using an externalization of the diagrams is recommended.

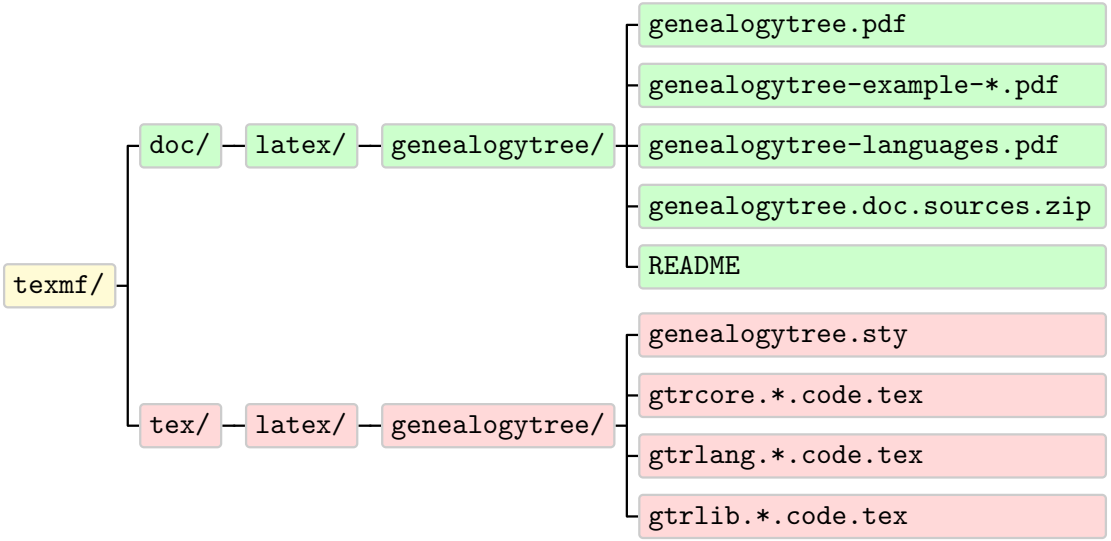
1.3 Comparison with Other Packages

This is not really a comparison, but more a hinting to other possibilities for graph drawing. I am not aware of another package with focus on *genealogy trees* as defined here, but surely there are excellent other graph drawing packages. The first to name is Ti*k*Z itself. There, one will find a bunch of graph drawing tools with different algorithms, partly implemented in Lua. The second one is the `forest` package which is also very powerful and does not need Lua.

1.4 Installation

Typically, `genealogytree` will be installed as part of a major LaTeX distribution and there is nothing special to do for a user.

If you intend to make a local installation *manually*, you have to install not only `tcolorbox.sty`, but also all `*.code.tex` files in the local `texmf` tree.



1.5 Loading the Package

The base package `genealogytree` loads the package `tcolorbox` [3] with its `skins`, `fitting`, and `external` libraries. This also loads several other packages are loaded, especially `tikz` [4] with its `arrows.meta` and `fit` libraries.

`genealogytree` itself is loaded in the usual manner in the preamble:

```
\usepackage{genealogytree}
```

The package takes option keys in the key-value syntax. Alternatively, you may use these keys later in the preamble with `\gtruselibrary` → P. 13 (see there). For example, the key to use debug code is:

```
\usepackage[debug]{genealogytree}
```

1.6 Libraries

The base package **genealogytree** is extendable by program libraries. This is done by using option keys while loading the package or inside the preamble by applying the following macro with the same set of keys.

U 2017-01-20 `\gtruselibrary{⟨key list⟩}`

Loads the libraries given by the ⟨*key list*⟩.

```
\gtruselibrary{all}
```

The following keys are used inside `\gtruselibrary` respectively `\usepackage` without the key tree path `/gtr/library/`.

`/gtr/library/debug` (LIB debug)

Loads additional code for debugging a *genealogy tree*. This is also useful for displaying additional informations during editing a graph; see Chapter 11 on page 227.

`/gtr/library/templates` (LIB templates)

Loads additional code for templates. These are styles to set various options by one key; see Chapter 12 on page 247.

`/gtr/library/all` (style, no value)

Loads all libraries listed above.

For the curious readers: There are additional *core* libraries which are loaded automatically and which are not mentioned here. Also, languages are special libraries which are loaded by `\gtrloadlanguage` → P. 226.

Third party libraries (denoted external libraries) can also be loaded using `\gtruselibrary`, if they follow the file naming scheme `gtrlib.⟨key⟩.code.tex`

```
% Loading 'gtrlib.foobar.code.tex'
\gtruselibrary{foobar}
```

Note that such external libraries are not version-checked as internal libraries are.

1.7 How to Get Started

You don't have to read this long document to start creating your first *genealogy tree*. A good starting point is to browse through the tutorials in Chapter 2 on page 15 and simply try some of them on your computer. The package provides a lot of options and allows many adjustments to node setting, but you do not need to know them in advance to create the first examples.

You should also take a look at Chapter 12 on page 247, where template examples are shown which could be useful instantly.

For using advanced features, it is not harmful to know at least the basics of Ti*k*Z [4] and **tcolorbox** [3], since **genealogytree** is based on both.

2

Tutorials

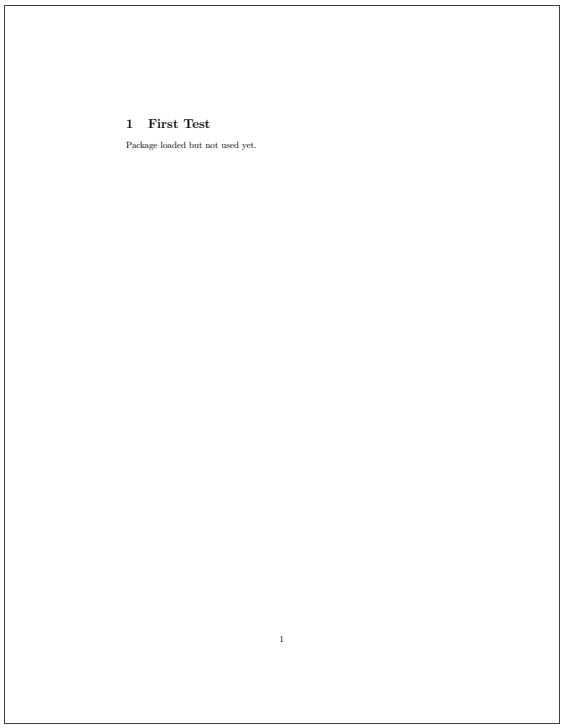
2.1 Tutorial: First Steps (Ancestor Tree)

2.1.1 Document Setup

Most examples in this documentation will display some *code snippets* which one can use in a document with proper set-up. This very basic tutorial will create a tiny full document. If this does not work on your system, there is probably some installation problem. Typically, this can be solved by just updating the TEX distribution.

The very first document just tests, if the package is installed:

```
\documentclass{article}
\usepackage[all]{genealogytree}
\begin{document}
  \section{First Test}
  Package loaded but not used yet.
\end{document}
```

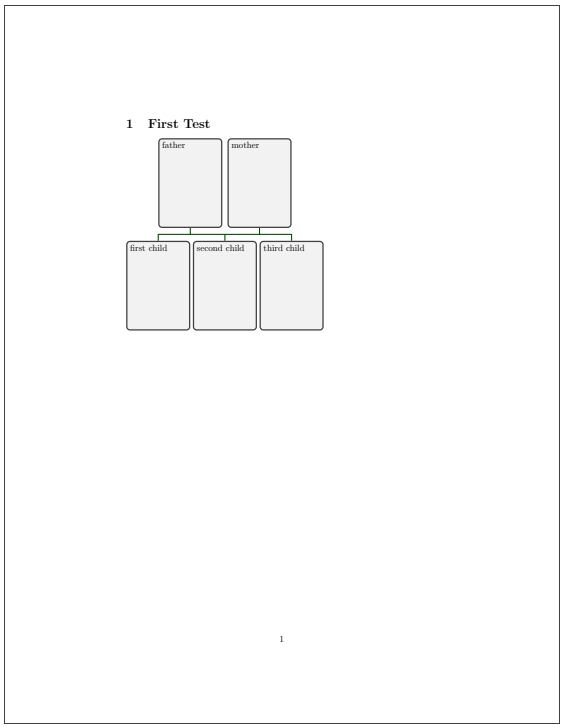


2.1.2 Creation of a Basic Ancestor Diagram

Now, we start with the very first *genealogy tree*. Such trees are *family-centric*. So, let us begin with a family consisting of mother and father and three children. Chapter 4 on page 61 tells us, that there are different kinds of families; the two main ones are `parent` and `child`. For a single family, the choice is quite irrelevant. Here, we think about extending the example to grandparents. Therefore, we take the `parent` construct.

Before the details are discussed, let us try a full example:

```
\documentclass{article}
\usepackage[all]{genealogytree}
\begin{document}
  \section{First Test}
  \begin{tikzpicture}
    \genealogytree{
      parent{
        g{first child}
        c{second child}
        c{third child}
        p{father}
        p{mother}
      }
    }
  \end{tikzpicture}
\end{document}
```



The environment `tikzpicture` is the main picture environment from the Ti*k*Z [4] package. `\genealogytree` → P. 55 can only be used inside such an environment.

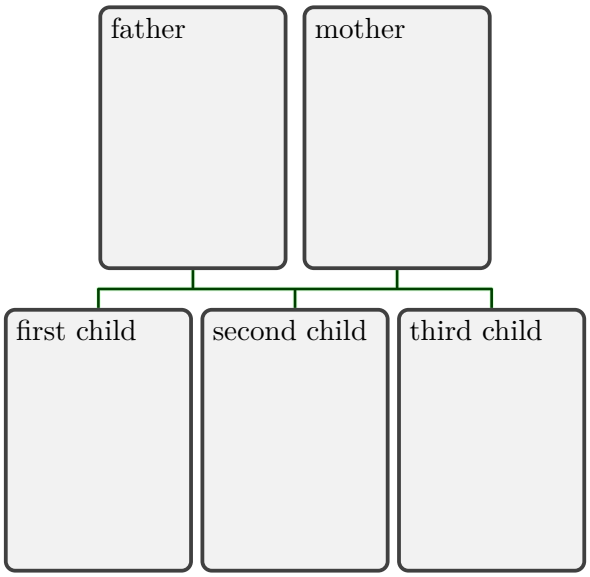
When testing this example, be very sure about setting all braces properly. The internal parser will react very sensitive on input errors. Of course, this is nothing new for a TEX user, but larger trees will have a lot of hierarchical braces and error messages will likely not be very talkative about *where* the error is.

The `genealogytree` package uses `{}` pairs for structuring and `[]` pairs for options like *typical* LATEX does.

In the following, we will not see full documents but *code snippets* and their output. Note that the full example used the `all` option to load all libraries of `genealogytree`, see Section 1.6 on page 13. You should also add all libraries for testing the examples. Later, you may choose to reduce the libraries.

Let us look at our example again with focus on the relevant part:

```
\begin{tikzpicture}
  \genealogytree{
    parent{
      g{first child}
      c{second child}
      c{third child}
      p{father}
      p{mother}
    }
  }
\end{tikzpicture}
```

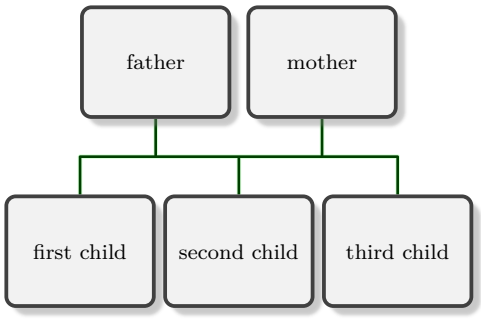


Our `parent` family has two parents denoted by `p` and three children, two of them denoted by `c` as expected. But one child, not necessarily the first one, is denoted by `g`. This is the `g`-node which connects a family *uplink* to another family. Here, we have a single family which is the root family where no *uplink* exists. Nevertheless, a `g`-node has to be present.

2.1.3 Applying options

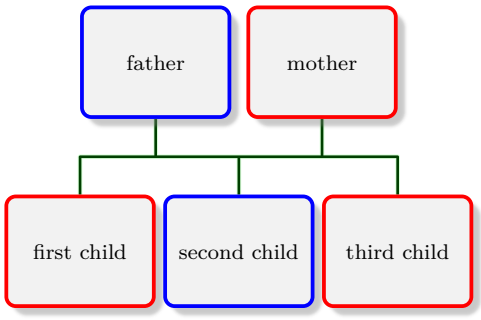
Certainly, the size and distance of the nodes can be changed. A quick way to adapt the graph is to use preset values from a given /gtr/template→ P. 247. We put this to the option list of \genealogytree→ P. 55.

```
\begin{tikzpicture}
  \genealogytree[template=signpost]{
    parent{
      g{first child}
      c{second child}
      c{third child}
      p{father}
      p{mother}
    }
  }
\end{tikzpicture}
```



Options can also be set for families and nodes. We enhance our *genealogy tree* by giving /gtr/male→ P. 99 and /gtr/female→ P. 99 settings to the nodes:

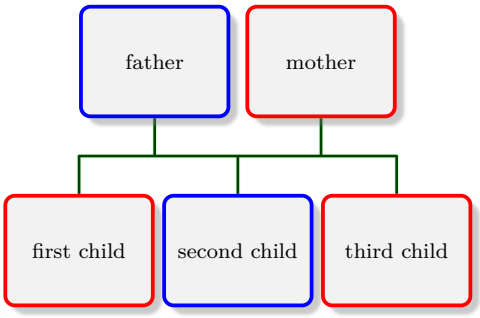
```
\begin{tikzpicture}
  \genealogytree[template=signpost]{
    parent{
      g[female]{first child}
      c[male]{second child}
      c[female]{third child}
      p[male]{father}
      p[female]{mother}
    }
  }
\end{tikzpicture}
```



2.1.4 Growing the Tree

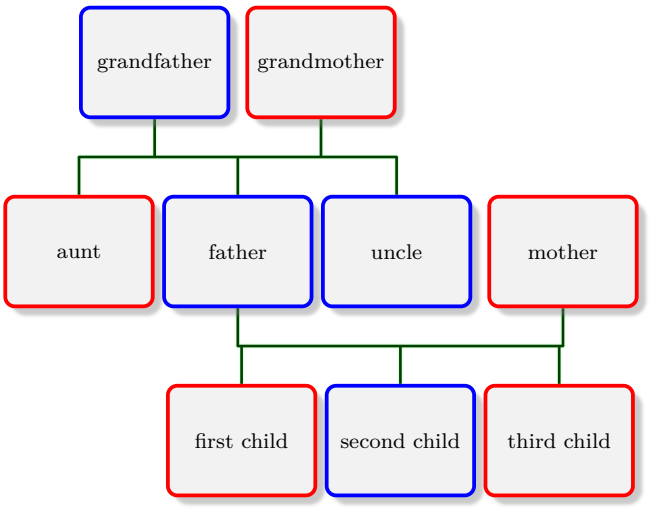
As next step, the father node shall get a grandfather and a grandmother. For this, the father node has to become a g-node which links the grandparents family to the root family:

```
\begin{tikzpicture}
  \genealogytree[template=signpost]{
    parent{
      g[female]{first child}
      c[male]{second child}
      c[female]{third child}
      parent{
        g[male]{father}
      }
      p[female]{mother}
    }
  }
\end{tikzpicture}
```



Visually, nothing happened. But, the father node is now g-node of a new family. As in our root family, we can add parents p and even other children c. Of course, these other children are the siblings of the father node:

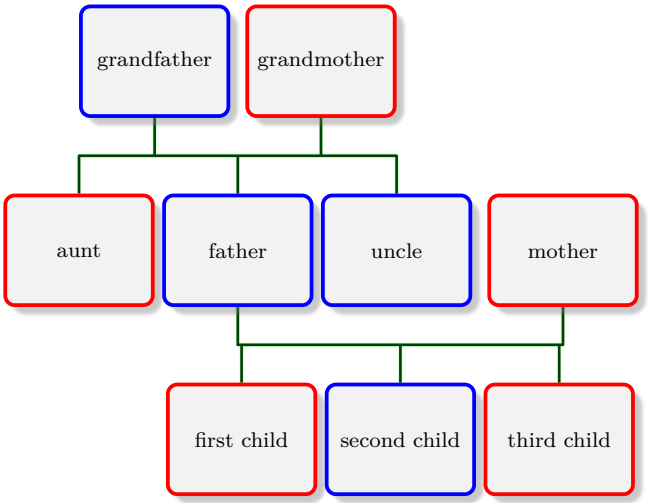
```
\begin{tikzpicture}
  \genealogytree[template=signpost]{
    parent{
      g[female]{first child}
      c[male]{second child}
      c[female]{third child}
      parent{
        c[female]{aunt}
        g[male]{father}
        c[male]{uncle}
        p[male]{grandfather}
        p[female]{grandmother}
      }
      p[female]{mother}
    }
  }
\end{tikzpicture}
```



One could replace all parents p by parent families with a single g-node. This would increase the expense, but can be a good thing when editing and compiling a tree step by step.

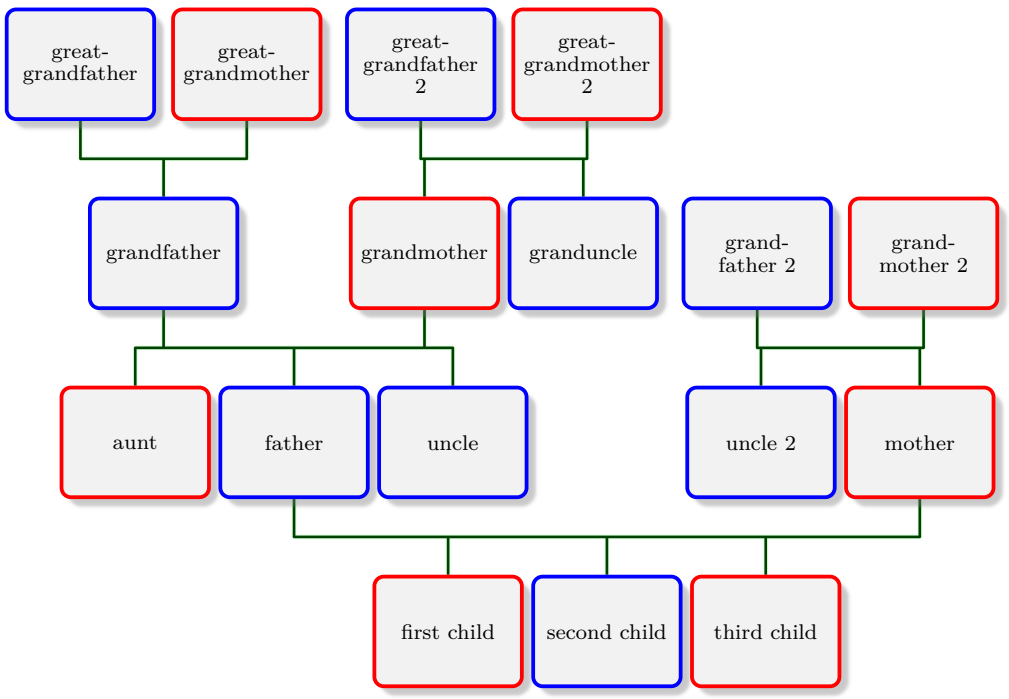
We now prepare our tree for expansion and replace mother, grandfather, and grandmother with appropriate `parent` families.

```
\begin{tikzpicture}
  \genealogytree[template=signpost]{
    parent{
      g[female]{first child}
      c[male]{second child}
      c[female]{third child}
      parent{
        c[female]{aunt}
        g[male]{father}
        c[male]{uncle}
        parent
        {
          % former 'p' node
          g[male]{grandfather}
        }
        parent{
          % former 'p' node
          g[female]{grandmother}
        }
      }
      parent
      {
        % former 'p' node
        g[female]{mother}
      }
    }
  }
\end{tikzpicture}
```



Again, we populate the three added families with parents `p` and children `c`.

```
\begin{tikzpicture}
  \genealogytree[template=signpost]{
    parent{
      g[female]{first child}
      c[male]{second child}
      c[female]{third child}
      parent{
        c[female]{aunt}
        g[male]{father}
        c[male]{uncle}
        parent
        {
          g[male]{grandfather}
          p[male]{great-grandfather}
          p[female]{great-grandmother}
        }
        parent{
          g[female]{grandmother}
          p[male]{great-grandfather 2}
          p[female]{great-grandmother 2}
          c[male]{granduncle}
        }
      }
      parent
      {
        c[male]{uncle 2}
        g[female]{mother}
        p[male]{grandfather 2}
        p[female]{grandmother 2}
      }
    }
  }
\end{tikzpicture}
```



2.1.5 Prioritize and Colorize a Path

After the tree has been grown to its final size, we want to influence the node positions. Let us assume that the lineage from *first child* to *great-grandmother 2* has to be especially emphasized.

To prioritize a node, the /gtr/pivot → P. 95 option can be used. This will place a node centered in relation to its ancestors and/or descendants. If this option is used for several connected nodes, a straight lineage is generated. All other nodes are placed automatically to respect this lineage.

```
%...
      g[pivot,female]{first child}
%...
```

To emphasize this lineage further, the respective nodes should be colorized differently. With standard settings, every node is drawn as a `tcolorbox`. Box options are given by /gtr/box → P. 96. The options inside /gtr/box → P. 96 are `tcolorbox` options [3]. To add a yellowish background color and glow, we use:

```
%...
      g[pivot,box={colback=yellow!20,no shadow,fuzzy halo},female]{first child}
%...
```

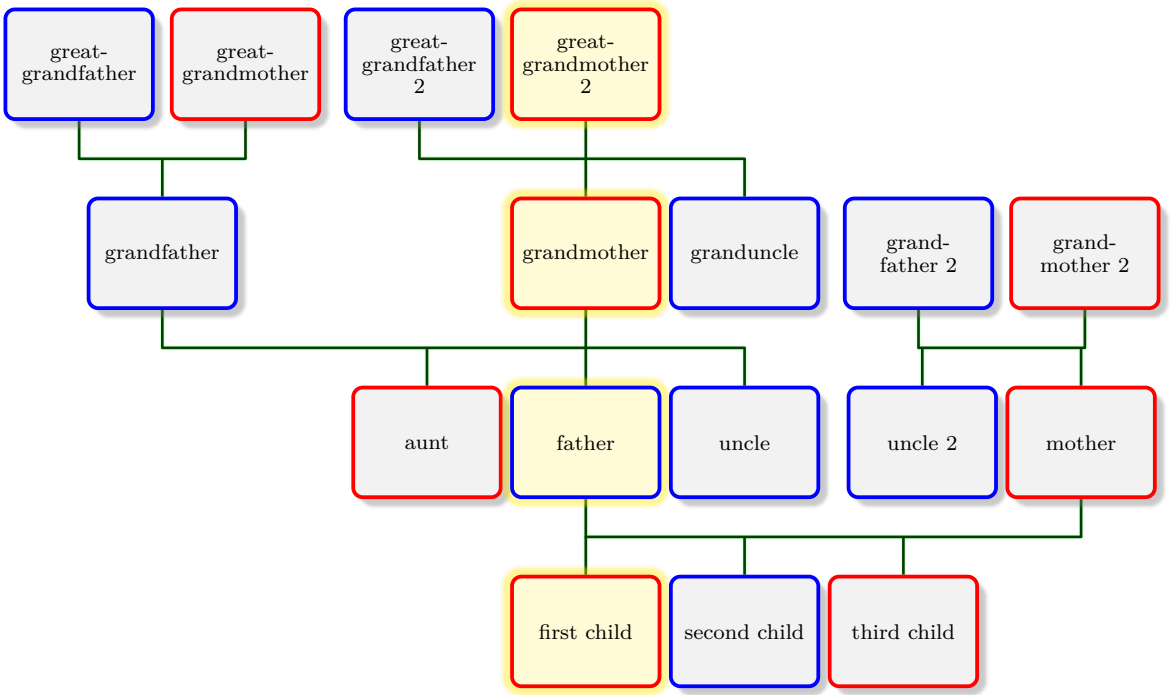
All option settings are `pgfkeys` options. So, it is easy to create a new option style `highlight` which can be used for each node in the lineage. This can be done by \gtrset → P. 58 or inside the option list of \genealogytree → P. 55.

```
\gtrset{highlight/.style={pivot,box={colback=yellow!20,no shadow,fuzzy halo}}}
```

Now, `highlight` can be used to apply /gtr/pivot → P. 95 and /gtr/box → P. 96 settings with one key word:

```
%...
      g[highlight,female]{first child}
%...
```

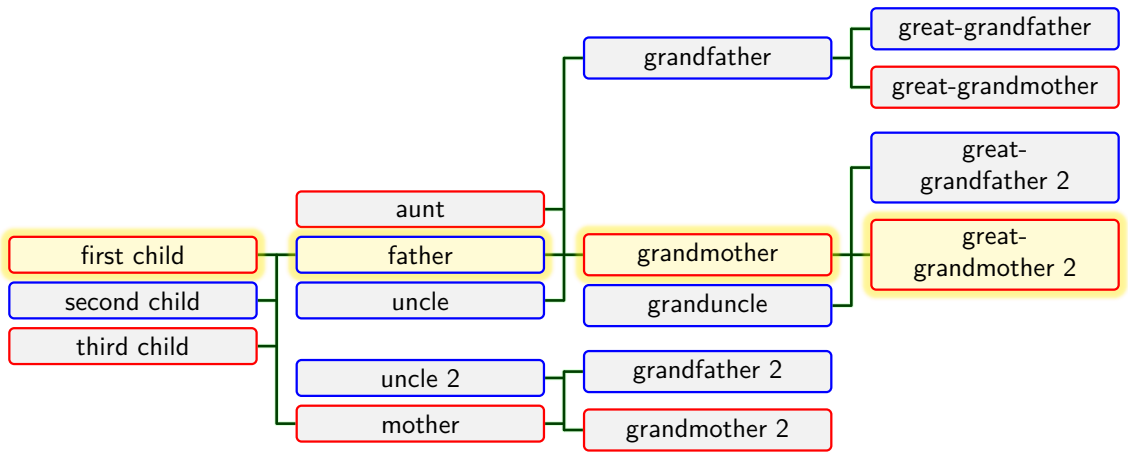
```
\begin{tikzpicture}
  \genealogytree[template=signpost,
    highlight/.style={pivot,box={colback=yellow!20,no shadow,fuzzy halo}},
  ]{
    parent{
      g[highlight,female]{first child}
      c[male]{second child}
      c[female]{third child}
      parent{
        c[female]{aunt}
        g[highlight,male]{father}
        c[male]{uncle}
        parent
        {
          g[male]{grandfather}
          p[male]{great-grandfather}
          p[female]{great-grandmother}
        }
        parent{
          g[highlight,female]{grandmother}
          p[male]{great-grandfather 2}
          p[highlight,female]{great-grandmother 2}
          c[male]{granduncle}
        }
      }
      parent
      {
        c[male]{uncle 2}
        g[female]{mother}
        p[male]{grandfather 2}
        p[female]{grandmother 2}
      }
    }
  }
\end{tikzpicture}
```



2.1.6 Changing the Timeflow

A *genealogy tree* may be grown in four directions depending on the given /gtr/timeflow→ P. 78. Now, we will let the time flow to the left. Additionally, we replace the /gtr/template→ P. 247 setting by individual settings for /gtr/processing→ P. 128, /gtr/level size→ P. 82, /gtr/node size from→ P. 84, and /gtr/box→ P. 96.

```
\begin{tikzpicture}
  \genealogytree[
    timeflow=left,
    processing=tcolorbox,
    level size=3.3cm,node size from=5mm to 4cm,
    box={size=small,halign=center,valign=center,fontupper=\small\sffamily},
    highlight/.style={pivot,box={colback=yellow!20,no shadow,fuzzy halo}},
  ]{
    parent{
      g[highlight,female]{first child}
      c[male]{second child}
      c[female]{third child}
      parent{
        c[female]{aunt}
        g[highlight,male]{father}
        c[male]{uncle}
        parent
        {
          g[male]{grandfather}
          p[male]{great-grandfather}
          p[female]{great-grandmother}
        }
        parent{
          g[highlight,female]{grandmother}
          p[male]{great-grandfather 2}
          p[highlight,female]{great-grandmother 2}
          c[male]{granduncle}
        }
      }
      parent
      {
        c[male]{uncle 2}
        g[female]{mother}
        p[male]{grandfather 2}
        p[female]{grandmother 2}
      }
    }
  }
\end{tikzpicture}
```



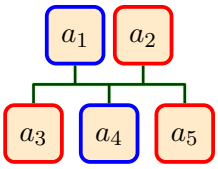
2.2 Tutorial: Diagram Manipulation by ID values (Descendant Tree)

This tutorial shows how set up and save a descendant diagram which is going to be manipulated without changing the base data.

2.2.1 Creation of a Basic Descendant Diagram

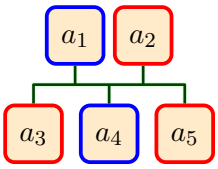
For a *genealogy tree* displaying a descendant lineage, we take the `child` construct. As a first step, we start with a single family. As always, this root family has to have a `g`-node which serves no important role for a root family, but stands for a parent here. The resulting *genealogy tree* will contain just small nodes without names to display some interconnection. For this, a preset value from a given `/gtr/template` → P. 247 is used for quick setup.

```
\begin{tikzpicture}
  \genealogytree[template=formal graph]{
    child{
      g[male]{a_1}
      p[female]{a_2}
      c[female]{a_3}
      c[male]{a_4}
      c[female]{a_5}
    }
  }
\end{tikzpicture}
```



The nodes of the diagram already have some options settings. To select and manipulate some or many nodes later without editing the data, the nodes and families can be given unique `/gtr/id` → P. 90 values.

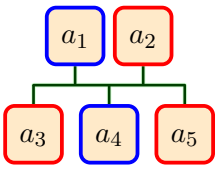
```
\begin{tikzpicture}
  \genealogytree[template=formal graph]{
    child[id=fam_A]{
      g[id=na1,male]{a_1}
      p[id=na2,female]{a_2}
      c[id=na3,female]{a_3}
      c[id=na4,male]{a_4}
      c[id=na5,female]{a_5}
    }
  }
\end{tikzpicture}
```



2.2.2 Growing the Tree

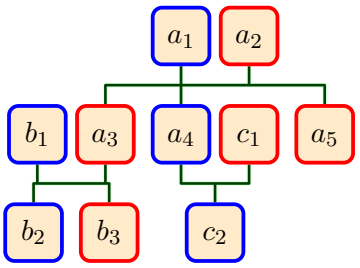
The nodes a_3 and a_4 shall become parent of their own families. To proceed in small steps, we make them g-nodes of single-member child families which does not change the diagram. Both new families get their own /gtr/id → P. 90 values for later reference.

```
\begin{tikzpicture}
  \genealogytree[template=formal graph]{
    child[id=fam_A]{
      g[id=na1,male]{a_1}
      p[id=na2,female]{a_2}
      child[id=fam_B]{
        g[id=na3,female]{a_3}
      }
      child[id=fam_C]{
        g[id=na4,male]{a_4}
      }
      c[id=na5,female]{a_5}
    }
  }
\end{tikzpicture}
```



Now, the new families are populated by a second parent and children.

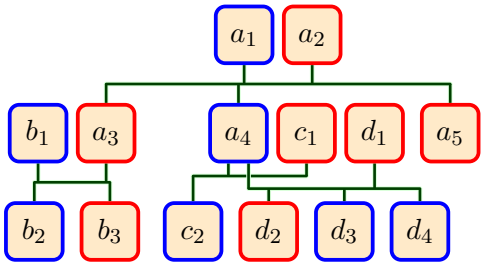
```
\begin{tikzpicture}
  \genealogytree[template=formal graph]{
    child[id=fam_A]{
      g[id=na1,male]{a_1}
      p[id=na2,female]{a_2}
      child[id=fam_B]{
        p[id=nb1,male]{b_1}
        g[id=na3,female]{a_3}
        c[id=nb2,male]{b_2}
        c[id=nb3,female]{b_3}
      }
      child[id=fam_C]{
        g[id=na4,male]{a_4}
        p[id=nc1,female]{c_1}
        c[id=nc2,male]{c_2}
      }
      c[id=na5,female]{a_5}
    }
  }
\end{tikzpicture}
```



As a specialty, a union construct can be used inside a child family. This represents a second husband or wife including children for the g-node of the current child family. A union does not get its own g-node but shares the g-node of the child family.

In our example, node a_4 gets a union which has to be placed inside the family with id value fam_C:

```
\begin{tikzpicture}
  \genealogytree[template=formal graph]{
    child[id=fam_A]{
      g[id=na1,male]{a_1}
      p[id=na2,female]{a_2}
      child[id=fam_B]{
        p[id=nb1,male]{b_1}
        g[id=na3,female]{a_3}
        c[id=nb2,male]{b_2}
        c[id=nb3,female]{b_3}
      }
      child[id=fam_C]{
        g[id=na4,male]{a_4}
        p[id=nc1,female]{c_1}
        c[id=nc2,male]{c_2}
        union[id=fam_D]{
          p[id=nd1,female]{d_1}
          c[id=nd2,female]{d_2}
          c[id=nd3,male]{d_3}
          c[id=nd4,male]{d_4}
        }
      }
      c[id=na5,female]{a_5}
    }
  }
\end{tikzpicture}
```

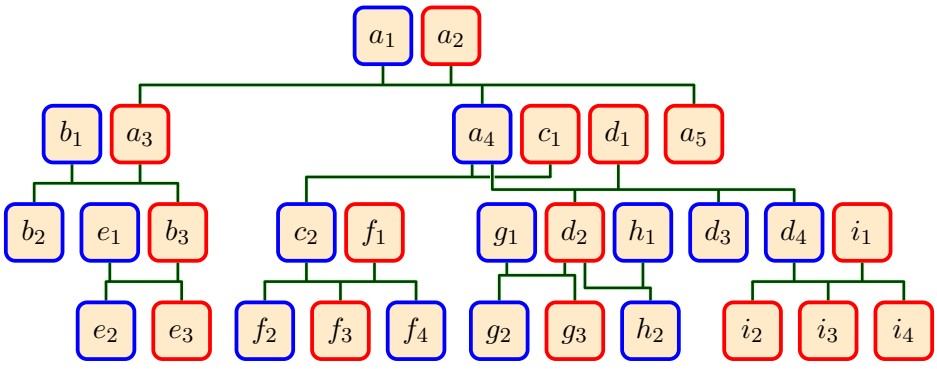


As the reader may note, for union constructs, the edges between the nodes are likely to overlap. Therefore, to attenuate the effect, the vertical positions of the edges for fam_C and fam_D are shifted automatically. Also, note the small visual separation at the cross-point of both family edges. This is generated by using /gtr/edge/foreground → P. 199 and /gtr/edge/background → P. 200 (here, as preset values).

In some context, fam_C and fam_D will be seen as a single aggregated family and will be called *patchwork* family.

The tree is now grown further following the previous construction pattern.

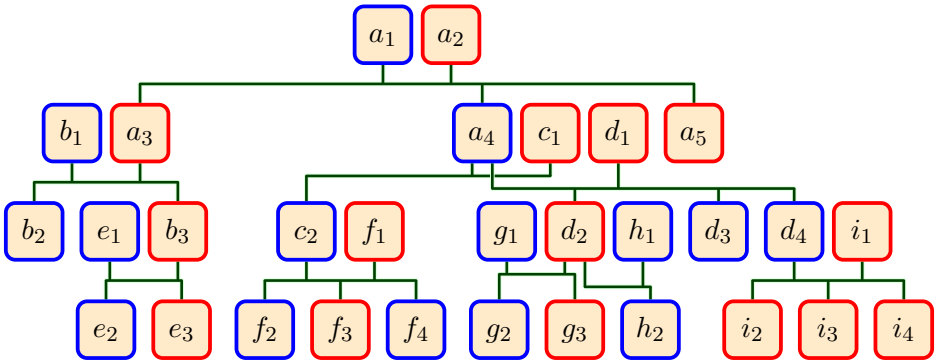
```
\begin{tikzpicture}
  \genealogytree[template=formal graph]{
    child[id=fam_A]{
      g[id=na1,male]{a_1}
      p[id=na2,female]{a_2}
      child[id=fam_B]{
        p[id=nb1,male]{b_1}
        g[id=na3,female]{a_3}
        c[id=nb2,male]{b_2}
        child[id=fam_E]{
          p[id=ne1,male]{e_1}
          g[id=nb3,female]{b_3}
          c[id=ne2,male]{e_2}
          c[id=ne3,female]{e_3}
        }
      }
      child[id=fam_C]{
        g[id=na4,male]{a_4}
        p[id=nc1,female]{c_1}
        child[id=fam_F]{
          g[id=nc2,male]{c_2}
          p[id=nf1,female]{f_1}
          c[id=nf2,male]{f_2}
          c[id=nf3,female]{f_3}
          c[id=nf4,male]{f_4}
        }
        union[id=fam_D]{
          p[id=nd1,female]{d_1}
          child[id=fam_G]{
            p[id=ng1,male]{g_1}
            g[id=nd2,female]{d_2}
            c[id=ng2,male]{g_2}
            c[id=ng3,female]{g_3}
            union[id=fam_H]{
              p[id=nh1,male]{h_1}
              c[id=nh2,male]{h_2}
            }
          }
          c[id=nd3,male]{d_3}
          child[id=fam_I]{
            g[id=nd4,male]{d_4}
            p[id=ni1,female]{i_1}
            c[id=ni2,female]{i_2}
            c[id=ni3,female]{i_3}
            c[id=ni4,female]{i_4}
          }
        }
      }
      c[id=na5,female]{a_5}
    }
  }
\end{tikzpicture}
```



2.2.3 Separating Diagram Data and Diagram Drawing

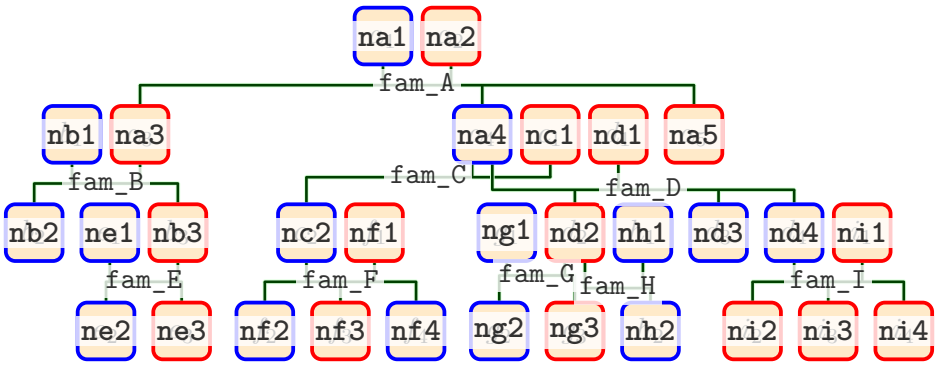
For the second part of this tutorial, the final diagram data is now saved into a file `example.formal.graph`, see Section 14.3 on page 289. That is everything inside `\genealogytree` → P. 55 without the options of `\genealogytree` → P. 55. Using the `input` construct, graph drawing is done simply by the following:

```
\begin{tikzpicture}
  \genealogytree[template=formal graph]
    {input{example.formal.graph}}
\end{tikzpicture}
```



In our example, the given `/gtr/id` → P. 90 values are easy to remember since we choose them nearly identical to the node content. For a not-so-formal example, this will be different. To avoid digging into the data source for finding some `/gtr/id` → P. 90 value, the `/gtr/show id` → P. 243 setting from the LIB debug library is useful:

```
\begin{tikzpicture}
  \genealogytree[template=formal graph,show id]
    {input{example.formal.graph}}
\end{tikzpicture}
```



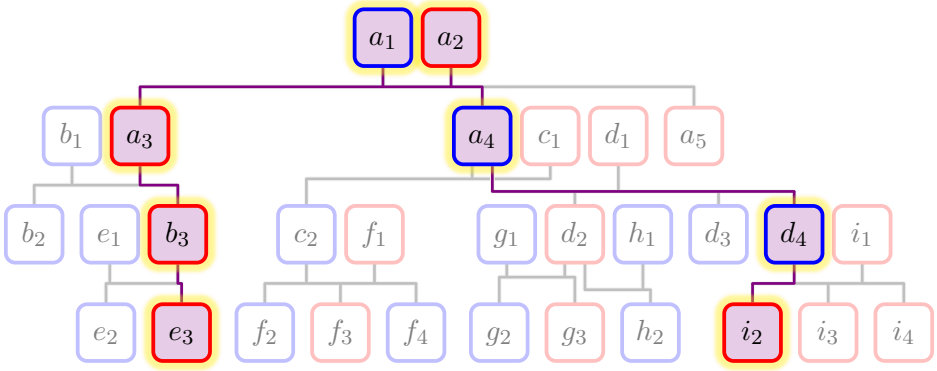
2.2.4 Emphasizing a Relationship Path

For the given example data, we will emphasize the relationship between node e_3 and node i_2 in our graph. The diagram above exposes the id values along the relationship path as `ne3`, `nb3`, `na3`, `na1` and `na2`, `na4`, `nd4`, `ni2`. For emphasizing, we dim the colors of all other nodes and brighten the colors for the nodes along this path.

All these manipulations are done inside the option list of `\genealogytree` → P. 55 without changing the diagram data directly.

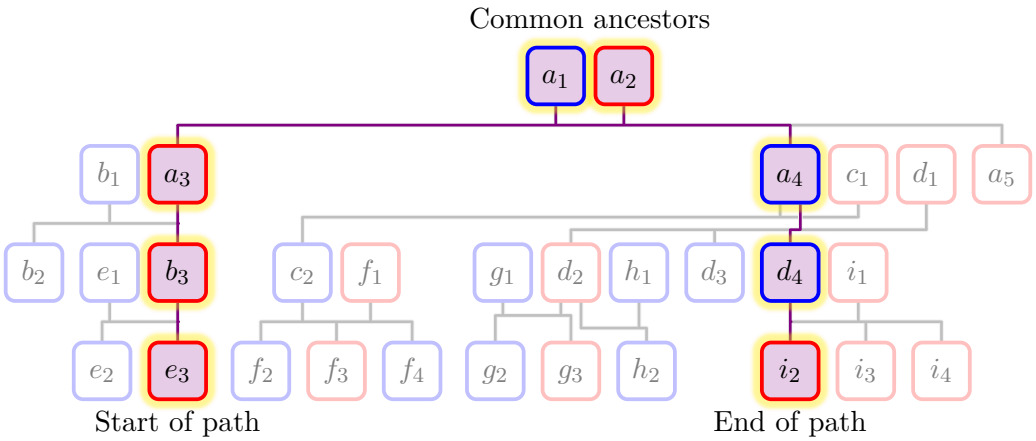
1. `/gtr/box` → P. 96 sets options to wash out all nodes.
2. `/gtr/edges` → P. 190 sets options to wash out all edges.
3. `/gtr/options for node` → P. 93 sets box options to all nodes along the selected path to display them emphasized.
4. `/gtr/extra edges for families` → P. 213 sets extra edge options to all emphasized the connection line along the selected path.

```
\begin{tikzpicture}
  \genealogytree[template=formal graph,
    box={colback=white,colupper=black!50,opacityframe=0.25},
    edges={foreground=black!25,background=black!5},
    options for node={ne3,nb3,na3,na1,na2,na4,nd4,ni2}%
      {box={colback=blue!50!red!20,colupper=black,opacityframe=1,fuzzy halo}},
    extra edges for families={
        x={fam_E}{nb3}{ne3},x={fam_B}{na3}{nb3},
        x={fam_A}{na1,na2}{na3,na4},
        x={fam_D}{na4}{nd4},x={fam_I}{nd4}{ni2}
      }{foreground=blue!50!red,no background},
  ]
    {input{example.formal.graph}}
\end{tikzpicture}
```



Also, the parameters for the auto-layout algorithm can be changed using the known id values. Our selected relationship path is emphasized further by straightening the lineages. This is done by inserting /gtr/pivot → P. 95 values through /gtr/options for node → P. 93.

```
\begin{tikzpicture}
  \genealogytree[template=formal graph,
    box={colback=white,colupper=black!50,opacityframe=0.25},
    edges={foreground=black!25,background=black!5},
    options for node={ne3,nb3,na3,na1,na2,na4,nd4,ni2}%
      {box={colback=blue!50!red!20,colupper=black,opacityframe=1,fuzzy halo}},
    extra edges for families={
        x={fam_E}{nb3}{ne3},x={fam_B}{na3}{nb3},
        x={fam_A}{na1,na2}{na3,na4},
        x={fam_D}{na4}{nd4},x={fam_I}{nd4}{ni2}
      }{foreground=blue!50!red,no background},
    options for node={ne3,nb3,nd4,ni2}{pivot},
    options for node={na3,na4}{pivot=parent},
  ]
    {input{example.formal.graph}}

\node[below] at (ne3.south) {Start of path};
\node[below] at (ni2.south) {End of path};
\path (na1) -- node[above=5mm]{Common ancestors} (na2);

\end{tikzpicture}
```



All given /gtr/id → P. 90 values are also Ti*k*Z nodes. Therefore, a *genealogy tree* can easily be annotated and extended by Ti*k*Z instructions.

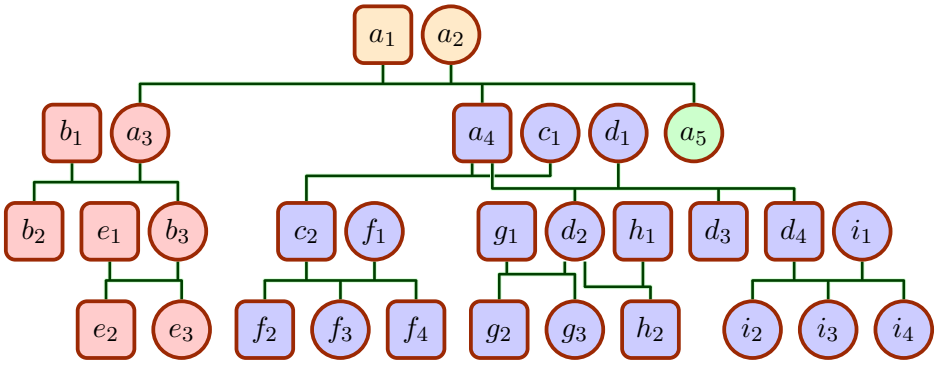
2.2.5 Coloring Subtrees

For the given example data, the descendants of the root family should now by colored with three different colors. All in-law nodes should be visually separated from descendants of a_1 and a_2.

As a first step, the subtree denoted by `fam_B` is colored in red by `/gtr/options for subtree` → P. 105. Analogously, `fam_C` is colored in blue. Node a_5 is a leaf node without own family and, therefore, is colored using `/gtr/options for node` → P. 93. Also, the preset `/gtr/male` → P. 99 and `/gtr/female` → P. 99 styles are made ineffective for this drawing.

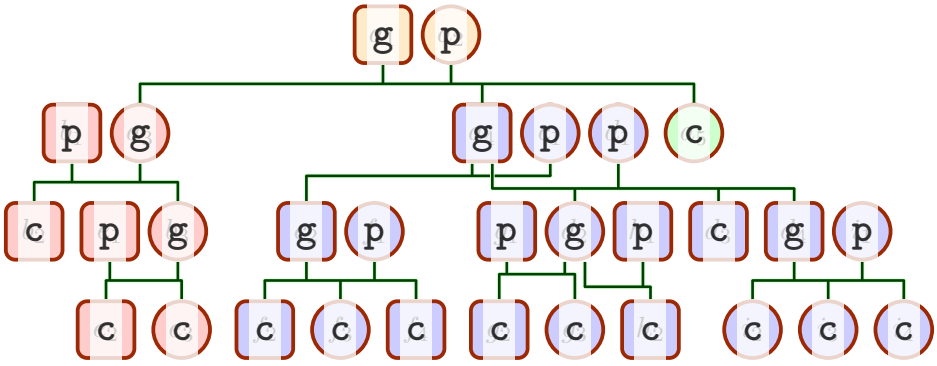
This gives a colored *genealogy tree*, but not only the direct descendents are colored, but all members of descendant families:

```
\begin{tikzpicture}
  \genealogytree[template=formal graph,
  male/.style={},female/.style={box={circular arc}},
  options for subtree={fam_B}{box={colback=red!20!white}},
  options for subtree={fam_C,fam_D}{box={colback=blue!20!white}},
  options for node={na5}{box={colback=green!20!white}},
  ]
    {input{example.formal.graph}}
\end{tikzpicture}
```



As can be inspected using `/gtr/show type` → P. 245 from the LIB debug library, the nodes to be excluded are all p-nodes:

```
\begin{tikzpicture}
  \genealogytree[template=formal graph,show type,
  male/.style={},female/.style={box={circular arc}},
  options for subtree={fam_B}{box={colback=red!20!white}},
  options for subtree={fam_C,fam_D}{box={colback=blue!20!white}},
  options for node={na5}{box={colback=green!20!white}},
  ]
    {input{example.formal.graph}}
\end{tikzpicture}
```

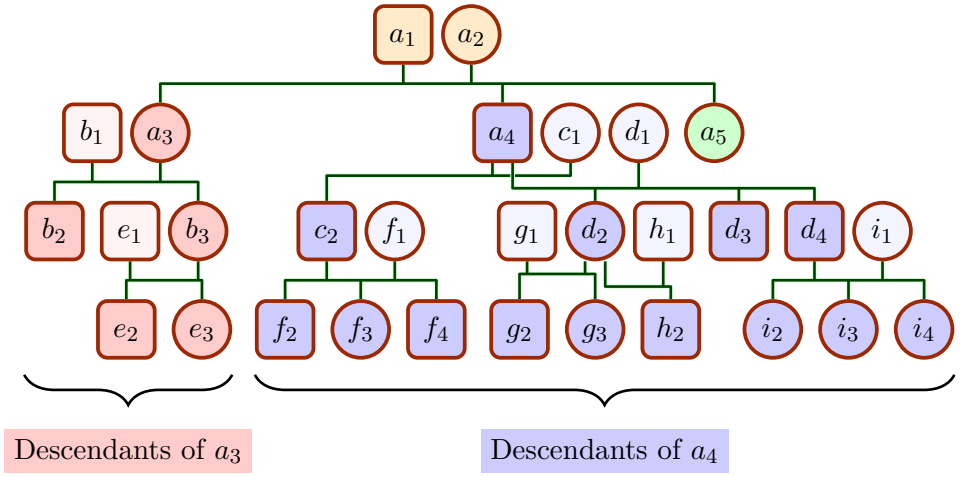


This node type is accessible by \gtrnodetype → P. 59 or \gtrifpnode → P. 59. We use this to set up a tcolorbox style bleach p which wash out the in-law nodes, when \gtrifpnode → P. 59 expands to ⟨*true*⟩. This style is formulated locally by /gtr/tcbset → P. 113:

```
%...
tcbset={bleach p/.code={%
  \gtrifpnode{\tcbset{enhanced jigsaw,opacityback=0.2}}{}%
}},
%...
```

This gives:

```
\begin{tikzpicture}
  \genealogytree[template=formal graph,
  male/.style={},female/.style={box={circular arc}},
  tcbset={bleach p/.code={%
    \gtrifpnode{\tcbset{enhanced jigsaw,opacityback=0.2}}{}%
  }},
  options for subtree={fam_B}{box={colback=red!20!white,bleach p}},
  options for subtree={fam_C,fam_D}{box={colback=blue!20!white,bleach p}},
  options for node={na5}{box={colback=green!20!white}},
  ]
    {input{example.formal.graph}}

\draw[decorate,decoration={brace,amplitude=4mm,mirror,raise=2mm},
  line width=1pt,yshift=0pt] (nb2.south west|-ne3.south) -- (ne3.south east)
  node [align=center,below=9mm,midway,fill=red!20!white] {Descendants of $a_3$};
\draw[decorate,decoration={brace,amplitude=4mm,mirror,raise=2mm},
  line width=1pt,yshift=0pt] (nf2.south west) -- (ni4.south east)
  node [align=center,below=9mm,midway,fill=blue!20!white] {Descendants of $a_4$};
\end{tikzpicture}
```



2.3 Tutorial: A Database Family Diagram (Sand Clock)

This tutorial shows the application of a database concept for representing the node content. Also, the sand clock diagram is shown which units ancestor and descendant graphs.

2.3.1 Creation of a Basic Sand Clock Diagram

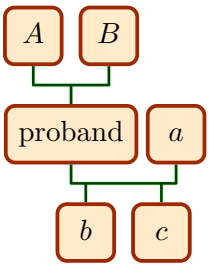
The `sandclock` construct is the starting point for a sand glass type *genealogy tree*. The *proband* is the constriction for the sand glass where the ancestors and descendants of the proband meet. Therefore, a `sandclock` can and should contain `child` and `parent` constructs. There has to be exactly one `child`, because a `sandclock` has no own `g`-node but inherits it from the `child`.

For the following examples, we use `genealogypicture` → P. 57 to create *genealogy trees*. This is a handy combination of `tikzpicture` and `\genealogytree` → P. 55.

```
% minimal sandclock diagram
\begin{genealogypicture}[template=formal graph]
sandclock
{
  child{
    g{\text{ proband }}
  }
}
\end{genealogypicture}
```

Now, we can add `parent` and `child` constructs. Here, we use single-member families since the tree will be grown later on.

```
% basic sandclock diagram (ready to be extended)
\begin{genealogypicture}[template=formal graph]
sandclock
{
  child{
    g{\text{ proband }}
    p{a}
    child{% grows in child direction
      g{b}
    }
    child{% grows in child direction
      g{c}
    }
  }
  parent{% grows in parent direction
    g{A}
  }
  parent{% grows in parent direction
    g{B}
  }
}
\end{genealogypicture}
```



2.3.2 Node Content in Database Format

In the following, we will construct a family diagram for Carl Friedrich Gauß (1777–1855).

We step back a little bit and consider the minimal sand clock diagram as starting point. The node content, of course, may be any formatted LaTeX text.

```
\begin{genealogypicture}
sandclock
{
  child{
    g{Carl Friedrich \textbf{Gau\ss{}},
      born 1777, died 1855
    }
  }
}
\end{genealogypicture}
```

Carl Friedrich **Gauß**, born 1777, died 1855

In this context, the database approach means that the node content should not contain a formatted text but just the data core which is going to be formatted later. This is the same principle as for creating a bibliography with `biblatex` or `bibtex`.

So, we tell `genealogytree` that we want to use such a database concept by setting `/gtr/processing` → P. 128 to `database`. Now, the content can be given as a key-value list. See Chapter 7 on page 151 for all feasible keys.

Further, we tell `genealogytree` how to format this given data by setting `/gtr/database format` → P. 162 to some predefined value. Everything can be customized later.

The basic information for a person is `/gtr/database/name` → P. 155, `/gtr/database/male` → P. 155 or `/gtr/database/female` → P. 155, `/gtr/database/birth` → P. 157 and `/gtr/database/death` → P. 158.

```
\begin{genealogypicture}[
  processing=database,
  database format=medium marriage below,
]
sandclock
{
  child{
    g[id=GauxCarl1777]{
      male,
      name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
      birth={1777-04-30}{Braunschweig (Niedersachsen)},
      death={1855-02-23}{G\"ottingen (Niedersachsen)},
      profession={Mathematiker, Astronom, Geod\"at und Physiker},
      image={Carl_Friedrich_Gauss.jpg},
    }
  }
}
\end{genealogypicture}
```

Johann *Carl Friedrich* Gauss
★ April 30, 1777 in Braunschweig (Niedersachsen)
† February 23, 1855 in Göttingen (Niedersachsen)
Mathematiker, Astronom, Geodät und Physiker.

In the example above, we also added a `/gtr/database/profession` → P. 156 which appears in the output, and an `/gtr/database/image` → P. 156 which is not used. Note the markup with `\pref` → P. 172 and `\surn` → P. 172 inside the `/gtr/database/name` → P. 155 which marks preferred name parts and the surname. There is no name parsing as known from `bib(la)tex`.

As `/gtr/id` → P. 90 for Carl Friedrich Gauß, «`GauxCarl1777`» was chosen. Such id values could be chosen to your liking. As a common guideline, they should be human readable/understandable, because they may be needed to manipulate the graph afterwards and something like

«gd0h-xhag-0ugh-opod-89sq-sdqj-8pah» may not be easily associated with Gauß. Also, they should be automatically producible for the comfortable case, that a genealogy program exports data in this format.

In this tutorial, this common guideline is sharpened to follow these rules:

- A person id is build as `XxxxYyyyZzzz`, where `Xxxx` are four letters of the surname, `Yyyy` are four letters of the (preferred) first name, and `Zzzz` is the year of birth (maybe, estimated).
- A family id is build as `AaaaBbbbZzzz`, where `Aaaa` are four letters of the husbands surname, `Bbbb` are four letters of the wifes surname, and `Zzzz` is the year of marriage (maybe, estimated).
- Only `a`,...,`z`, `A`, ..., `Z` letters are used. Accented letters like umlauts are replaced by letters from the masks above. If a name part is shorter than four letters, letters from the masks are used for complement.
- If two identical id values are produced for two or more persons or families following these rules, they are distinguished by adding `-`⟨*counter*⟩.

2.3.3 Formatting the Node Content

First, we adapt some graph geometry settings to our liking. `/gtr/node size` → P. 83, `/gtr/level size` → P. 82, and `/gtr/level distance` → P. 81 set size and distance values.

With `/gtr/box` → P. 96, we set `tcolorbox` options for the appearance of the node box. Note that `\gtrDBsex` is set to `male` by the database values inside the node content. There are predefined `/tcb/male` → P. 99 and `/tcb/female` → P. 99 styles, but with `/gtr/tcbset` → P. 113 we change them to colorize also the interior of the box.

```
\begin{genealogypicture}[
  processing=database,
  database format=medium marriage below,
  node size=2.4cm,
  level size=3.5cm,
  level distance=6mm,
  tcbset={male/.style={colframe=blue,colback=blue!5},
          female/.style={colframe=red,colback=red!5}},
  box={fit basedim=7pt,boxsep=2pt,segmentation style=solid,
    halign=left,before upper=\parskip1pt,
    \gtrDBsex,drop fuzzy shadow,
  },
]
sandclock
{
  child{
    g[id=GauxCarl1777]{
      male,
      name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
      birth={1777-04-30}{Braunschweig (Niedersachsen)},
      death={1855-02-23}{G\"ottingen (Niedersachsen)},
      profession={Mathematiker, Astronom, Geod\"at und Physiker},
      image={Carl_Friedrich_Gauss.jpg},
    }
  }
}
\end{genealogypicture}
```

Johann *Carl Friedrich* Gauss
★ April 30, 1777 in Braunschweig (Niedersachsen)
† February 23, 1855 in Göttingen (Niedersachsen)
Mathematiker, Astronom, Geodät und Physiker.

As second step, we adapt the format of the given data inside the node output.

```
%...
list separators hang,
name font=\bfseries,
surn code={\textcolor{red!50!black}{#1}},
place text={\newline}{},
date format=d/mon/yyyy,
%...
```

With /gtr/list separators hang → P. 182, the event list is formatted with hanging indent. /gtr/name font → P. 173 and /gtr/surn code → P. 172 are used to format the name of the person. /gtr/place text → P. 178 inserts a \newline before the place of an event is printed in our example. Finally, /gtr/date format → P. 174 is used to change the way dates are printed.

```
\begin{genealogypicture}[
  processing=database,
  database format=medium marriage below,
  node size=2.4cm,
  level size=3.5cm,
  level distance=6mm,
  list separators hang,
  name font=\bfseries,
  surn code={\textcolor{red!50!black}{#1}},
  place text={\newline}{},
  date format=d/mon/yyyy,
  tcbset={male/.style={colframe=blue,colback=blue!5},
          female/.style={colframe=red,colback=red!5}},
  box={fit basedim=7pt,boxsep=2pt,segmentation style=solid,
    halign=left,before upper=\parskip1pt,
    \gtrDBsex,drop fuzzy shadow,
  },
]
sandclock
{
  child{
    g[id=GauxCarl1777]{
      male,
      name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
      birth={1777-04-30}{Braunschweig (Niedersachsen)},
      death={1855-02-23}{G\"ottingen (Niedersachsen)},
      profession={Mathematiker, Astronom, Geod\"at und Physiker},
      image={Carl_Friedrich_Gauss.jpg},
    }
  }
}
\end{genealogypicture}
```

Johann *Carl Friedrich* Gauß
★ 30/Apr/1777
Braunschweig
(Niedersachsen)
† 23/Feb/1855
Göttingen
(Niedersachsen)
Mathematiker, Astronom, Geodät und Physiker.

2.3.4 Adding Images

The predefined /gtr/database format → P. 162 options do not consider images. But we can add image code easily to be /gtr/box → P. 96 definition which accepts tcolorbox settings.

/tcb/if image defined → P. 187 decides, if an image is present, and sets tcolorbox options accordingly. The file name of this image is \gtrDBimage which is set to Carl_Friedrich_Gauss.jpg[1] by the database values inside the node content.

Options from *The tcolorbox package* [3] are used to enlarge the box width by 25mm and fill the space with this image:

```
\begin{genealogypicture}[
  processing=database,
  database format=medium marriage below,
  node size=2.4cm,
  level size=3.5cm,
  level distance=6mm,
  list separators hang,
  name font=\bfseries,
  surn code={\textcolor{red!50!black}{#1}},
  place text={\newline}{},
  date format=d/mon/yyyy,
  tcbset={male/.style={colframe=blue,colback=blue!5},
          female/.style={colframe=red,colback=red!5}},
  box={fit basedim=7pt,boxsep=2pt,segmentation style=solid,
    halign=left,before upper=\parskip1pt,
    \gtrDBsex,drop fuzzy shadow,
    if image defined={add to width=25mm,right=25mm,
      underlay={\begin{tcbclipinterior}\path[fill overzoom image=\gtrDBimage]
        ([xshift=-24mm]interior.south east) rectangle (interior.north east);
      \end{tcbclipinterior}},
    }{},
  },
]
sandclock
{
  child{
    g[id=GauxCarl1777]{
      male,
      name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
      birth={1777-04-30}{Braunschweig (Niedersachsen)},
      death={1855-02-23}{G\"ottingen (Niedersachsen)},
      profession={Mathematiker, Astronom, Geod\"at und Physiker},
      image={Carl_Friedrich_Gauss.jpg},
    }
  }
}
\end{genealogypicture}
```

[1] http://commons.wikimedia.org/wiki/File:Carl_Friedrich_Gauss.jpg

2.3.5 Full Example with Frame

The sandclock example is now extended with family and ancestors and descendants of Gauß as shown at the beginning of this tutorial. The full sandclock example is saved as «example.gauss.graph»:

File «example.gauss.graph»

```
sandclock{
  child[id=GauaOsth1805]{
    p[id=OsthJoha1780]{
      female,
      name={\pref{Johanna} Elisabeth Rosina \surn{Osthoff}},
      birth={1780-05-08}{Braunschweig (Niedersachsen)},
      marriage={1805-10-09}{Braunschweig (Niedersachsen)},
      death={1809-10-11}{G\"ottingen (Niedersachsen)},
      comment={Wei\ss{}gerberstochter},
    }
    g[id=GauxCarl1777]{
      male,
      name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
      birth={1777-04-30}{Braunschweig (Niedersachsen)},
      death={1855-02-23}{G\"ottingen (Niedersachsen)},
      profession={Mathematiker, Astronom, Geod\"at und Physiker},
      image={Carl_Friedrich_Gauss.jpg},
    }
    c[id=GauxCarl1806]{
      male,
      name={\pref{Carl} Joseph \surn{Gau\ss{}}},
      birth={1806-08-21}{Braunschweig (Niedersachsen)},
      death={1873-07-04}{Hannover (Niedersachsen)},
    }
    c[id=GauxWilh1808]{
      female,
      name={\pref{Wilhelmina} \surn{Gau\ss{}}},
      birth={1808-02-29}{G\"ottingen (Niedersachsen)},
      death={1840-08-12}{T\"ubingen (Baden-W\"urttemberg)},
    }
    c[id=GauxLudw1809]{
      male,
      name={\pref{Ludwig} \surn{Gau\ss{}}},
      birth={1809-09-10}{G\"ottingen (Niedersachsen)},
      death={1810-03-01}{G\"ottingen (Niedersachsen)},
    }
    union[id=GauaWald1810]{
      p[id=WaldFrie1788]{
        female,
        name={\pref{Friederica} Wilhelmine \surn{Waldeck}},
        birth={1788-04-15}{G\"ottingen (Niedersachsen)},
        marriage={1810-08-14}{G\"ottingen (Niedersachsen)},
        death={1831-09-12}{G\"ottingen (Niedersachsen)},
        comment={Rechtswissenschaftlerstochter},
      }
      c[id=GauxEuge1811]{
        male,
        name={\pref{Eugen} Peter Samuel Marius \surn{Gau\ss{}}},
        birth={1811-07-29}{G\"ottingen (Niedersachsen)},
        death={1896-07-04}{Columbia (Missouri)},
        profession={Rechtswissenschaftler, Kaufmann},
      }
      c[id=GauxWilh1813]{
        male,
        name={\pref{Wilhelm} August Carl Matthias \surn{Gau\ss{}}},
```

```
      birth={1813-10-23}{G\"ottingen (Niedersachsen)},
      death={1879-08-23}{St. Louis (Missouri)},
    }
    c[id=GauxTher1816]{
      female,
      name={Henriette Wilhelmine Karoline \pref{Therese} \surn{Gau\ss{}}},
      birth={1816-06-09}{G\"ottingen (Niedersachsen)},
      death={1864-02-11}{Dresden (Sachsen)},
    }
  }
}
parent[id=GoosEgge1735]{
  g[id=GauxGebh1743]{
    male,
    name={\pref{Gebhard} Dietrich \surn{Gau\ss{}}},
    birth={1743-02-13}{Braunschweig (Niedersachsen)},
    death={1808-04-14}{Braunschweig (Niedersachsen)},
    profession={G\"artner, Wasserkunstmeister, Rechnungsf\"uhrer},
  }
  parent[id=GoosLbtk1705]{
    g[id=GoosJyrg1715]{
      male,
      name={\pref{J\"urgen} \surn{Gooss}},
      birth={1715}{V\"olkenrode (Niedersachen)},
      death={1774-07-05}{Braunschweig (Niedersachsen)},
      profession={Lehmmaurer},
    }
    p[id=GoosHinr1655]{
      male,
      name={\pref{Hinrich} \surn{Gooss}},
      birth={(caAD)1655}{},
      death={1726-10-25}{V\"olkenrode (Niedersachen)},
    }
    p[id=LxtkKath1674]{
      female,
      name={\pref{Katharina} \surn{L\"utken}},
      birth={1674-08-19}{V\"olkenrode (Niedersachen)},
      marriage={1705-11-24}{V\"olkenrode (Niedersachen)},
      death={1749-04-15}{V\"olkenrode (Niedersachen)},
    }
  }
  p[id=EggeKath1710]{
    female,
    name={\pref{Katharina} Magdalena \surn{Eggenlings}},
    birth={(caAD)1710}{Rethen},
    marriage={(caAD)1735}{V\"olkenrode (Niedersachen)},
    death={1774-04-03}{Braunschweig (Niedersachsen)},
  }
}
parent[id=BentKron1740]{
  g[id=BenzDoro1743]{
    female,
    name={\pref{Dorothea} \surn{Benze}},
    birth={1743-06-18}{Velpke (Niedersachsen)},
    marriage={1776-04-25}{Velpke (Niedersachsen)},
    death={1839-04-18}{G\"ottingen (Niedersachsen)},
    comment={Steinhauerstochter},
  }
  parent[id=BentBbbb1740]{
    g[id=BentChri1717]{
      male,
      name={\pref{Christoph} \surn{Bentze}},
      birth={1717}{Velpke (Niedersachsen)},
      death={1748-09-01}{Velpke (Niedersachsen)},
```

```
        profession={Steinhauer},
      }
      p[id=BentAndr1687]{
        male,
        name={\pref{Andreas} \surn{Bentze}},
        birth={1687-02}{},
        death={(caAD)1750}{Velpke (Niedersachsen)},
      }
    }
    p[id=KronKath1710]{
      female,
      name={\pref{Katharina} \surn{Krone}},
      birth={(caAD)1710}{},
      death={1743/}{Velpke (Niedersachsen)},
    }
  }
}
```

As final polish, /gtr/edges → P. 190 are set to be **rounded** and the used symbols are recorded by /gtr/symbols record reset → P. 222 and displayed by \gtrSymbolsLegend → P. 223 inside /gtr/after tree → P. 113.

Finally, the whole diagram is put into a titled `tcolorbox` to exhibit the example:

```
\begin{tcolorbox}[enhanced,sharp corners,boxrule=0.6pt,left=0pt,right=0pt,
  colback=blue!50!black,interior style image=goldshade.png,
  halign=center,center title,fonttitle=\bfseries,
  title={The Family of Carl Friedrich Gau\ss{} (1777--1855)}  ]
\begin{genealogypicture}[
  processing=database,
  database format=medium marriage below,
  node size=2.4cm,
  level size=3.5cm,
  level distance=6mm,
  list separators hang,
  name font=\bfseries,
  surn code={\textcolor{red!50!black}{#1}},
  place text={\newline}{},
  date format=d/mon/yyyy,
  tcbset={male/.style={colframe=blue,colback=blue!5},
          female/.style={colframe=red,colback=red!5}},
  box={fit basedim=7pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,
    \gtrDBsex,drop fuzzy shadow,
    if image defined={add to width=25mm,right=25mm,
      underlay={\begin{tcbclipinterior}\path[fill overzoom image=\gtrDBimage]
        ([xshift=-24mm]interior.south east) rectangle (interior.north east);
      \end{tcbclipinterior}},
    }{},
  },
  edges=rounded,
  symbols record reset,
  after tree={\node[font=\scriptsize\itshape,text width=1.8cm,below left,
    fill=white,fill opacity=0.4,text opacity=1]
    at (current bounding box.north east) {\gtrSymbolsLegend};},
]
  input{example.gauss.graph}
\end{genealogypicture}
\end{tcolorbox}
```

The Family of Carl Friedrich Gauß (1777–1855)

★=*born*, ⚭=*married*, †=*died*.

Hinrich Gooss
★ ca. 1655
† 25/Oct/1726 Völkenrode (Niedersachsen)

Katharina Lütken
★ 19/Aug/1674 Völkenrode (Niedersachsen)
† 15/Apr/1749 Völkenrode (Niedersachsen)
⚭ 24/Nov/1705 Völkenrode (Niedersachsen)

Andreas Bentze
★ Feb/1687
† ca. 1750 Velpke (Niedersachsen)

Jürgen Gooss
★ 1715 Völkenrode (Niedersachsen)
† 5/Jul/1774 Braunschweig (Niedersachsen)
Lehmmaurer.

Katharina Magdalena Eggenlings
★ ca. 1710 Rethen
† 3/Apr/1774 Braunschweig (Niedersachsen)
⚭ ca. 1735 Völkenrode (Niedersachsen)

Christoph Bentze
★ 1717 Velpke (Niedersachsen)
† 1/Sep/1748 Velpke (Niedersachsen)
Steinhauer.

Katharina Krone
★ ca. 1710
† after 1743 Velpke (Niedersachsen)

Gebhard Dietrich Gauß
★ 13/Feb/1743 Braunschweig (Niedersachsen)
† 14/Apr/1808 Braunschweig (Niedersachsen)
Gärtner, Wasserkunstmeister, Rechnungsführer.

Dorothea Benze
★ 18/Jun/1743 Velpke (Niedersachsen)
† 18/Apr/1839 Göttingen (Niedersachsen)
Steinhauerstochter.
⚭ 25/Apr/1776 Velpke (Niedersachsen)

Johanna Elisabeth Rosina Osthoff
★ 8/May/1780 Braunschweig (Niedersachsen)
† 11/Oct/1809 Göttingen (Niedersachsen)
Weißgerberstochter.
⚭ 9/Oct/1805 Braunschweig (Niedersachsen)

Johann Carl Friedrich Gauß
★ 30/Apr/1777 Braunschweig (Niedersachsen)
† 23/Feb/1855 Göttingen (Niedersachsen)
Mathematiker, Astronom, Geodät und Physiker.

Friederica Wilhelmine Waldeck
★ 15/Apr/1788 Göttingen (Niedersachsen)
† 12/Sep/1831 Göttingen (Niedersachsen)
Rechtswissenschaftlerstochter.
⚭ 14/Aug/1810 Göttingen (Niedersachsen)

Carl Joseph Gauß
★ 21/Aug/1806 Braunschweig (Niedersachsen)
† 4/Jul/1873 Hannover (Niedersachsen)

Wilhelmina Gauß
★ 29/Feb/1808 Göttingen (Niedersachsen)
† 12/Aug/1840 Tübingen (Baden-Württemberg)

Ludwig Gauß
★ 10/Sep/1809 Göttingen (Niedersachsen)
† 1/Mar/1810 Göttingen (Niedersachsen)

Eugen Peter Samuel Marius Gauß
★ 29/Jul/1811 Göttingen (Niedersachsen)
† 4/Jul/1896 Columbia (Missouri)
Rechtswissenschaftler, Kaufmann.

Wilhelm August Carl Matthias Gauß
★ 23/Oct/1813 Göttingen (Niedersachsen)
† 23/Aug/1879 St. Louis (Missouri)

Henriette Wilhelmine Karoline Therese Gauß
★ 9/Jun/1816 Göttingen (Niedersachsen)
† 11/Feb/1864 Dresden (Sachsen)

2.4 Tutorial: Descendants of the Grandparents (Connecting Trees)

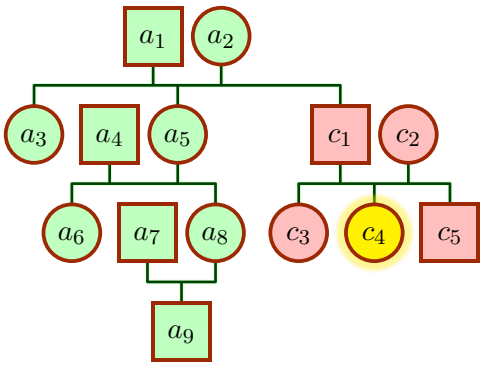
This tutorial will show how to create a «descendants of the grandparents» type of diagram. For this, two *genealogy trees* have to be connected.

2.4.1 Descendants of the Two Grandparents

Since «descendants of the grandparents» cannot be formulated by the grammar of this package, see Chapter 4 on page 61, a descendants tree for each pair of grandparents is considered.

In this example, the proband is c_4. First, we take a look at the descendants of the father's parents a_1 and a_2. Note that we arranged the red colored father's family at the right hand side and that the father node c_1 has a combined `/gtr/id` → P. 90 of `c1@a`, because we added an `/gtr/id suffix` → P. 92 of `@a` to every id value of our first tree.

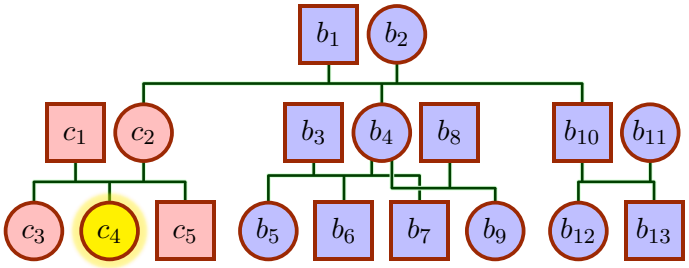
```
\begin{tikzpicture}
  \genealogytree[template=formal graph,id suffix=@a,
    tcbset={male/.style={sharp corners},female/.style={circular arc}},
    edges={anchoring=center},box={colback=green!25}]
  {
    child{
      g[male]{a_1}  p[female]{a_2}  c[female]{a_3}
      child{
        p[male]{a_4}  g[female]{a_5}  c[female]{a_6}
        child{
          p[male]{a_7}  g[female]{a_8}  c[male]{a_9}
        }
      }
      child[family box={colback=red!25}]{
        g[male,id=c1]{c_1}  p[female]{c_2}
        c[female]{c_3}  c[female,box={fuzzy halo,colback=yellow}]{c_4}  c[male]{c_5}
      }
    }
  }
\end{tikzpicture}
```



The other settings in this example are less important, but one may observe that the `/tcb/male` → P. 99 and `/tcb/female` → P. 99 styles were redefined to show not different colors but different shapes.

Secondly, we take a look at the descendants of the mother's parents b_1 and b_2. Note that this time we arranged the red colored mother's family at the left hand side and that the father node c_1 has a different /gtr/id → P. 90 of `c1@b`, because we added an /gtr/id suffix → P. 92 of `@b` to every id value of our second tree.

```
\begin{tikzpicture}
  \genealogytree[template=formal graph,id suffix=@b,
    tcbset={male/.style={sharp corners},female/.style={circular arc}},
    edges={anchoring=center},box={colback=blue!25}]
  {
    child{
      g[male]{b_1}  p[female]{b_2}
      child[family box={colback=red!25}]{
        p[male,id=c1]{c_1}  g[female]{c_2}
        c[female]{c_3}  c[female,box={fuzzy halo,colback=yellow}]{c_4}  c[male]{c_5}
      }
      child{
        p[male]{b_3}  g[female]{b_4}
        c[female]{b_5}  c[male]{b_6}  c[male]{b_7}
        union{
          p[male]{b_8}  c[female]{b_9}
        }
      }
      child{
        g[male]{b_{10}}  p[female]{b_{11}}
        c[female]{b_{12}}  c[male]{b_{13}}
      }
    }
  }
\end{tikzpicture}
```



2.4.2 Connected Diagram

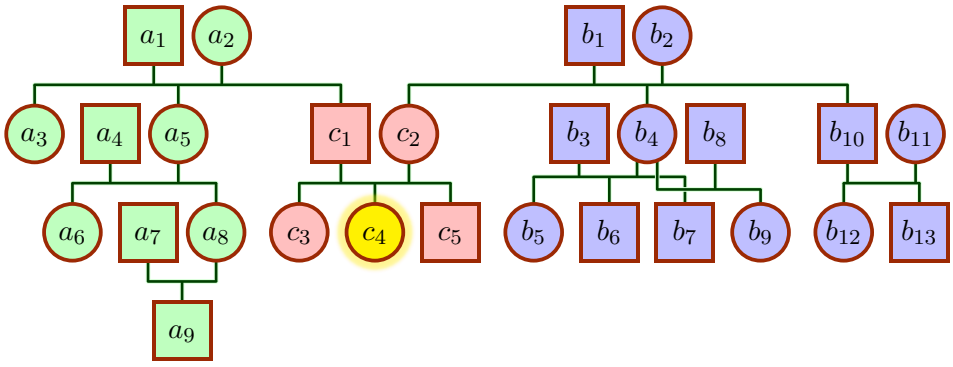
After the preparations, the \genealogytree → P. 55 diagrams can easily be put together.

Using /gtr/set position → P. 109 with value `c1@b at c1@a` for the second \genealogytree → P. 55 puts node c_1 from the diagram directly on node c_1 of the first \genealogytree → P. 55. Note that in a more complicated situation more manual intervention may be necessary to avoid unwanted overlapping of other nodes.

In the first \genealogytree → P. 55, one sees a /gtr/phantom* → P. 124 option which makes the first family $c_1, \ldots, c_5$ invisible but still space reserving.

Using /gtr/id suffix → P. 92 or /gtr/id prefix → P. 92 allows to distinguish nodes with the same id value in different trees. Otherwise, the id values would have to be changed manually.

```
\begin{tikzpicture}
  \gtrset{template=formal graph,
    tcbset={male/.style={sharp corners},female/.style={circular arc}},
    edges={anchoring=center},
  }
  \genealogytree[box={colback=green!25},id suffix=@a]
  {
    child{
      g[male]{a_1}  p[female]{a_2}  c[female]{a_3}
      child{
        p[male]{a_4}  g[female]{a_5}  c[female]{a_6}
        child{
          p[male]{a_7}  g[female]{a_8}  c[male]{a_9}
        }
      }
      child[phantom*]{
        g[male,id=c1]{c_1}  p[female]{c_2}
        c[female]{c_3}  c[female]{c_4}  c[male]{c_5}
      }
    }
  }
  \genealogytree[box={colback=blue!25}, id suffix=@b, set position=c1@b at c1@a  ]
  {
    child{
      g[male]{b_1}  p[female]{b_2}
      child[family box={colback=red!25}]{
        p[male,id=c1]{c_1}  g[female]{c_2}
        c[female]{c_3}  c[female,box={fuzzy halo,colback=yellow}]{c_4}  c[male]{c_5}
      }
      child{
        p[male]{b_3}  g[female]{b_4}
        c[female]{b_5}  c[male]{b_6}  c[male]{b_7}
        union{
          p[male]{b_8}  c[female]{b_9}
        }
      }
      child{
        g[male]{b_{10}}  p[female]{b_{11}}
        c[female]{b_{12}}  c[male]{b_{13}}
      }
    }
  }
\end{tikzpicture}
```



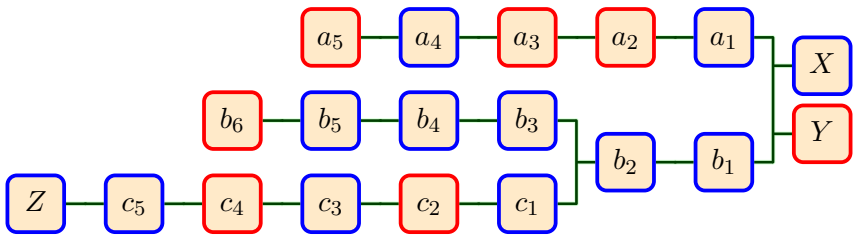
2.5 Tutorial: Multi-Ancestors

In the following, a multi-ancestor denotes an ancestor who is connected over more than one descendency line to the proband, i.e. where descendants have children with other descendents. This situation *is not covered* by the auto-layout algorithm. Depending on the complexity, such a graph can be drawn by manipulating one or more *genealogy trees*.

2.5.1 Triple Ancestor Example

In this example, X and Y are triple ancesters of the proband Z. As first step, a `child` diagram is set up with all three descendency lines from X and Y to Z, but only the c line is drawn fully. In our example, a_5 and b_5 are parents to b_6, also b_6 and c_5 are parents to Z (not yet displayed).

```
\begin{genealogypicture}[template=formal graph,timeflow=left,
]
child{
  g[male]{X}  p[female]{Y}
  child{  g[male]{a_1}
    child{  g[female]{a_2}
      child{  g[female]{a_3}
        child{  g[male]{a_4}
          child{  g[female]{a_5}
  }}}}}
  child{  g[male]{b_1}
    child{  g[male]{b_2}
      child{  g[male]{b_3}
        child{  g[male]{b_4}
          child{  g[male]{b_5}
            child{  g[female]{b_6}
      }}}}
      child{  g[male]{c_1}
        child{  g[female]{c_2}
          child{  g[male]{c_3}
            child{  g[female]{c_4}
              child{  g[male]{c_5}
                child{  g[male]{Z}
      }}}}}}
  }}
}
\end{genealogypicture}
```

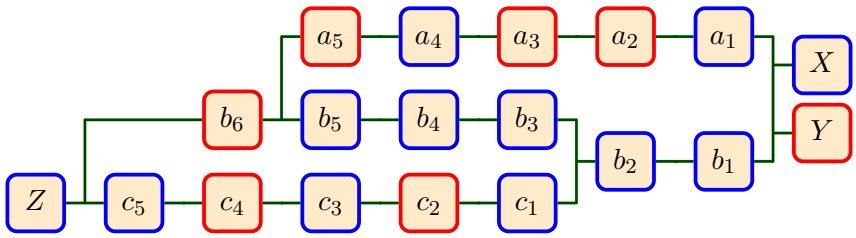


2.5.2 Adding Edges Manually

Now, we add the missing connections. For this, /gtr/id → P. 90 values are added to all involved nodes and families. Then, the connections are drawn using /gtr/add parent → P. 208 to add a_5 as additional parent for b_6, and to add b_6 as additional parent for Z.

The diagram has all necessary edges now, but, currently, is not balanced.

```
\begin{genealogypicture}[template=formal graph,timeflow=left,
  add parent=a5 to AB_fam,
  add parent=b6 to BC_fam,
]
child{
  g[male]{X}  p[female]{Y}
  child{  g[male]{a_1}
    child{  g[female]{a_2}
      child{  g[female]{a_3}
        child{  g[male]{a_4}
          child{  g[female,id=a5]{a_5}
  }}}}}
  child{  g[male]{b_1}
    child{  g[male]{b_2}
      child{  g[male]{b_3}
        child{  g[male]{b_4}
          child[id=AB_fam]{  g[male]{b_5}
            child{  g[female,id=b6]{b_6}
      }}}}
      child{  g[male]{c_1}
        child{  g[female]{c_2}
          child{  g[male]{c_3}
            child{  g[female]{c_4}
              child[id=BC_fam]{  g[male]{c_5}
                child{  g[male]{Z}
      }}}}}}
  }}
}
\end{genealogypicture}
```

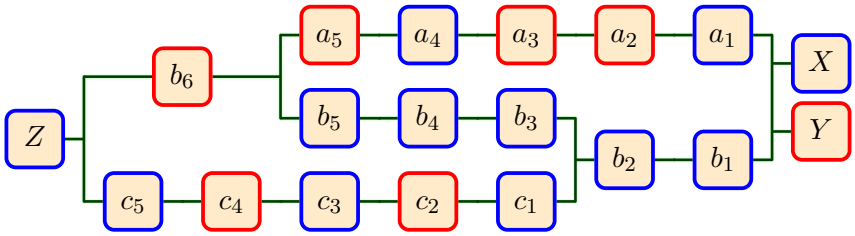


2.5.3 Manual Position Adjustments

To balance the graph, the final position of the b_6 is adjusted using the /gtr/tikz → P. 101 option with some Ti*k*Z shift operations. These final shiftings do not influence the auto-layout algorithm, but the edges move with the nodes.

Alternatively, /gtr/distance → P. 94, /gtr/pivot → P. 95, and /gtr/pivot shift → P. 104 can be used to influence the auto-layout algorithm. /gtr/pivot shift → P. 104 was used for the Z node to move it inside its family. But these manipulations would not move a node from its layer as was done for b_6 to display the generation skip.

```
\begin{genealogypicture}[template=formal graph,timeflow=left,
  add parent=a5 to AB_fam,
  add parent=b6 to BC_fam,
]
child{
  g[male]{X}  p[female]{Y}
  child{  g[male]{a_1}
    child{  g[female]{a_2}
      child{  g[female]{a_3}
        child{  g[male]{a_4}
          child{  g[female,id=a5]{a_5}
  }}}}}
  child{  g[male]{b_1}
    child{  g[male]{b_2}
      child{  g[male]{b_3}
        child{  g[male]{b_4}
          child[id=AB_fam]{  g[male]{b_5}
            child{  g[female,id=b6,tikz={xshift=-6.5mm,yshift=5.5mm}]{b_6}
      }}}}
      child{  g[male]{c_1}
        child{  g[female]{c_2}
          child{  g[male]{c_3}
            child{  g[female]{c_4}
              child[id=BC_fam,pivot shift=-8.25mm]{  g[male]{c_5}
                child{  g[male]{Z}
      }}}}}}
  }}
}
\end{genealogypicture}
```



2.6 Tutorial: Externalization

Creating diagrams requires a considerable amount of compilation time. Especially, if the document contains several diagrams, it is desirable to avoid compiling already finished diagrams on every document run. One solution would be to create a document for every diagram and to include the resulting PDF to the main document. Another way is known as *externalization*.

2.6.1 Externalization Process

Externalization means that diagrams are edited inside the main document as usual, but they are automatically exported to external files, compiled if necessary, and the resulting PDF files are included to the main document as images. At least two externalization options are available:

- Ti*k*Z externalization: Here, the whole original document is compiled in a sophisticated way. The `external` library of Ti*k*Z can automatically externalize all `tikzpicture` environments, see [4].
- `tcolorbox` externalization: Here, marked code snippets are compiled in a not so sophisticated but more robust way. The `external` library of `tcolorbox` only externalizes marked and named snippets. Besides `tikzpicture` environments, `genealogypicture` environments and other constructs can be externalized. These snippets are written to external files, compiled and the resulting PDF files are included to the main document as images, see [3].

The further tutorial describes the externalization using the `external` library of `tcolorbox`. This library is already included by the `genealogytree` package.

2.6.2 Document Setup

To use the externalization, the preamble of the main document has to contain the `\tcbEXTERNALIZE` command. Without this command, no externalization operation will be executed. Typically, `\tcbEXTERNALIZE` is the *last* entry of the preamble. Everything between `\tcbEXTERNALIZE` and `\begin{document}` is thrown away in the external document.

```
\documentclass{article}
\usepackage[all]{genealogytree}

\tcbEXTERNALIZE

\begin{document}
  \section{My heading}
  % ...
  Your main document containing texts, diagrams, figures, etc.
  % ...
\end{document}
```

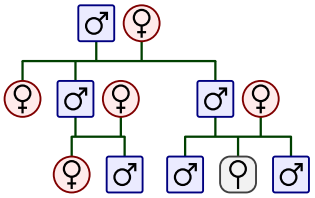
To use the externalization options, the compiler has to be called with the `-shell-escape` permission to authorize potentially dangerous system calls. Be warned that this is a security risk.

2.6.3 Marking Diagrams for Externalization

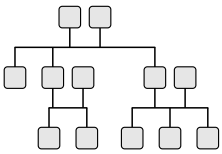
Before we care about externalization, we set up an example with two genealogy tree diagrams. One uses a `tikzpicture` and the other one the genealogypicture^{→ P. 57} shortcut.

```
This is the first example:\par\smallskip
\begin{tikzpicture}
\genealogytree[template=symbol nodes]{
  child{
    gm pf cf
    child{gm pf cf cm}
    child{gm pf cm c- cm}
  }
}
\end{tikzpicture}

\bigskip Now follows the second example:\par\smallskip
\begin{genealogypicture}[template=tiny boxes]
  child{
    g-p-c-
    child{g-p-c-c-}
    child{g-p-c-c-c-}
  }
\end{genealogypicture}
```

This is the first example:



Now follows the second example:



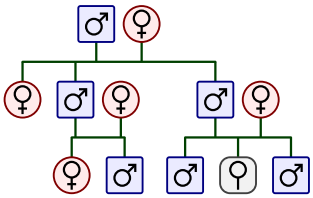
To externalize the diagrams, the document has to be set up as described in the previous subsection. Further, both diagrams have to be marked for externalization:

- Replace `tikzpicture` by `extikzpicture` and add a *unique name* as additional parameter.
- Replace genealogypicture^{→ P. 57} by exgenealogypicture^{→ P. 57} and add a *unique name* as additional parameter.

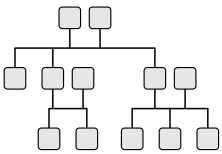
By default, these *unique names* are the names of the external files inside an `external` sub-directory. Depending on the operation system, the sub-directory `external` may have to be generated manually.

```
This is the first externalized example:\par\smallskip
\begin{extikzpicture}{first_example}
\genealogytree[template=symbol nodes]{
  child{
    gm pf cf
    child{gm pf cf cm}
    child{gm pf cm c- cm}
  }
}
\end{extikzpicture}

\bigskip Now follows the second externalized example:\par\smallskip
\begin{exgenealogypicture}{second_example}[template=tiny boxes]
  child{
    g-p-c-
    child{g-p-c-c-}
    child{g-p-c-c-c-}
  }
\end{exgenealogypicture}
```

This is the first externalized example:



Now follows the second externalized example:



After the diagrams are generated, they are compiled only again, if the diagram content changes. Changes caused by global settings will not be recognized.

To force recreation, just delete the external files. Another way is to add an exclamation mark as an option.

```
% The next example is always compiled because of '!' :
\begin{exgenealogypicture}[!]{second_example}[template=tiny boxes]
  child{
    g-p-c-
    child{g-p-c-c-}
    child{g-p-c-c-c-}
  }
\end{exgenealogypicture}
```

More details about controlling the externalization process are found in [3].

2.7 Tutorial: Conversion to Pixel Images

This tutorial is somewhat off-topic, because it considers the conversion of vector images to pixel images and is not directly related to genealogy trees. On the other hand, the need for conversion of such diagrams arises often, e.g. for third-party photo quality printing, web content, and import to pixel-focused software.

- This tutorial only considers the conversion from PDF (vector image) to PNG (pixel image). Further conversions to e.g. JPEG can be adapted or easily done from PNG to JPEG with many available tools.
- This presentation is not exhaustive. It only gives a short glimpse of some selected options for the conversion tools.

2.7.1 Command Line Conversion with Ghostscript

Here, we assume to already have a vector image file `example.pdf` which has to be converted to PNG.

First, Ghostscript[2] has to be installed on the system, if it not already is. It provides a command line tool `gs` or `gswin32c` or `gswin64c` for conversion which is not necessarily allocatable by the standard system path settings.

The following example conversion has to be given on a single line using a command shell (command window) or inside a script:

Command line example (a single line)

```
"c:/Program Files/gs/gs9.16/bin/gswin64c.exe" -dSAFER -dBATCH -dNOPAUSE
    -sDEVICE=png16m -r600 -dTextAlphaBits=4 -dGraphicsAlphaBits=4
    -sOutputFile="example.png" "example.pdf"
```

- Replace `"c:/Program Files/gs/gs9.16/bin/gswin64c.exe"` by an appropriate adaption for your system. Here, a 64 bit Windows installation with a specific version of Ghostscript is used.
- The example input file is `"example.pdf"` and the example output file is `"example.png"`.
- Adapt `-r600` to get larger or smaller pixel images. It defines the dots per inch resolution.

[2] www.ghostscript.com

2.7.2 Command Line Conversion with ImageMagick

Here, we assume to already have a vector image file `example.pdf` which has to be converted to PNG.

First, ImageMagick[3] has to be installed on the system, if it not already is. It provides a command line tool `convert` for conversion which is typically allocatable by the standard system path settings.

The following example conversion has to be given on a single line using a command shell (command window) or inside a script:

Command line example

```
convert -density 600 -alpha Remove -quality 90 "example.pdf" "example.png"
```

- Replace `convert` by an appropriate adaption for your system, if needed. On Windows, there a other programs named `convert` which may lead to conflicts.
- The example input file is `"example.pdf"` and the example output file is `"example.png"`.
- Adapt `-density 600` to get larger or smaller pixel images. It defines the dots per inch resolution.

2.7.3 Conversion with the 'standalone' Package

Conversion to pixel images can be done using the `standalone` package. Also, this package is designed to create standalone graphics.

By default, the package uses ImageMagick for conversion in the background. See the package documentation for specific considerations for Windows.

```
\documentclass[
  border=2mm,
  convert={ density=600 -alpha Remove, outext=.png }
  ]{standalone}
\usepackage[all]{genealogytree}
\usepackage{lmodern}

\begin{document}

\begin{genealogypicture}[template=symbol nodes]
  parent{
    g{male}
    insert{gtrparent4}
  }
\end{genealogypicture}

\end{document}
```

- If the document above is compiled with the `-shell-escape` permission, the compiled PDF files is converted to PNG automatically.
- Adapt `density=600` to get larger or smaller pixel images. It defines the dots per inch resolution.

[3] http://www.imagemagick.org

2.7.4 Conversion during Externalization

If externalization with `tcolorbox` is used, see Section 2.6 on page 49, possibly many PDF vector images are created. With the following hack, an automatic conversion with Ghostscript is added to the external compilation.

```
\documentclass{article}
\usepackage[all]{genealogytree}
\usepackage{lmodern}

\tcbEXTERNALIZE

\makeatletter
\appto\tcbexternal@corecompile{%
  \ShellEscape{%
    "c:/Program Files/gs/gs9.16/bin/gswin64c.exe"
    -dSAFER
    -dBATCH
    -dNOPAUSE
    -sDEVICE=png16m
    -r600
    -dTextAlphaBits=4
    -dGraphicsAlphaBits=4
    -sOutputFile="\tcbexternal@job@name.png"
    "\tcbexternal@job@name.pdf"
  }%
}
\makeatother

\begin{document}

\begin{exgenealogypicture}[!]{export}[template=symbol nodes]
  parent{
    g{male}
    insert{gtrparent4}
  }
\end{exgenealogypicture}

\end{document}
```

- If the document above is compiled with the `-shell-escape` permission, all externalized graphics (here: one) are converted to PNG automatically.
- Replace `"c:/Program Files/gs/gs9.16/bin/gswin64c.exe"` by an appropriate adaption for your system. Here, a 64 bit Windows installation with a specific version of Ghostscript is used.
- Adapt `-r600` to get larger or smaller pixel images. It defines the dots per inch resolution.
- Ghostscript can be replaced by ImageMagick.
- Note that this is a hack of internals of `tcolorbox`. This hack may become useless in the future. Especially, for a single image, the `standalone` package should be preferred.

3

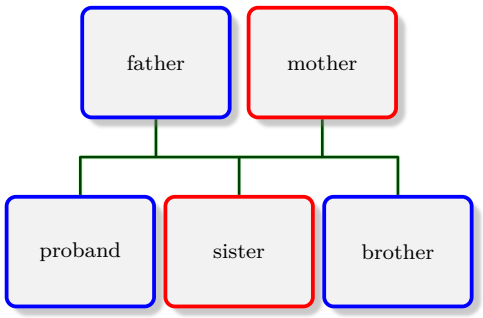
Genealogy Tree Macros

3.1 Creating a Genealogy Tree

`\genealogytree[⟨options⟩]{⟨tree content⟩}`

This is the main genealogy tree drawing macro of the package. The ⟨*tree content*⟩ has to obey to the tree grammar rules documented in Chapter 4 on page 61.
The ⟨*options*⟩ control how the drawing is done. These ⟨*options*⟩ are `pgf` keys with the key tree path `/gtr/` and they are described in the following.
The actual drawing is done with help of the Ti*k*Z package. Therefore, every `\genealogytree` has to be placed into a `tikzpicture` environment. It is possible to put several `\genealogytree` macros into the same `tikzpicture` and interconnect them.

```
\begin{tikzpicture}
\genealogytree[template=signpost]
{
  parent{
    g[male]{proband}
    c[female]{sister}
    c[male]{brother}
    p[male]{father}
    p[female]{mother}
  }
}
\end{tikzpicture}
```



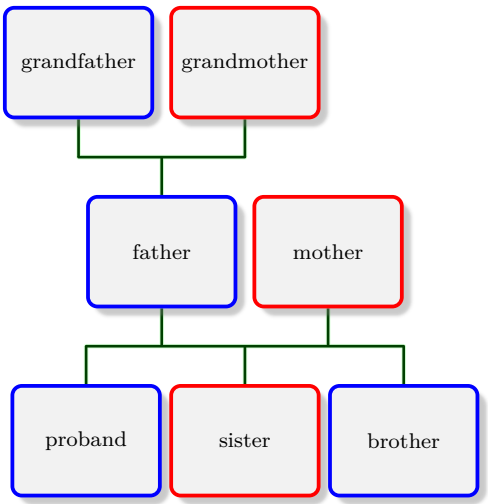
Detailed information about the genealogy tree grammar is found in Chapter 4 on page 61.
The short version is that a genealogy tree can have three types of nodes:

- `c` nodes are *child* nodes to a family,
- `p` nodes are *parent* nodes to a family,
- `g` nodes are usually *child* nodes to one family and *parent* nodes another family or even several families. Here, `g` can be memorized as *genealogy* node.

A *family* is a set of *parent* and *child* nodes which has to contain exactly one *genealogy* node (`g` node). All nodes of a family are interconnected with an edge set. In contrast to ordinary tree structures where an edge connects one node to another node, here, an edge connects a node to a family.
A genealogy tree can have following types of families:

- `parent`: A *parent* family may contain other `parent` families. Trees with this construction grow into *ancestor direction*.
- `child`: A *child* family may contain other `child` families or `union` families. Trees with this construction grow into *descendant direction*.
- `union`: A *union* ties a second child-type family to a `g` node as parent of this family.
- `sandclock`: A *sandclock* connects ancestors and descendants starting from a single proband.

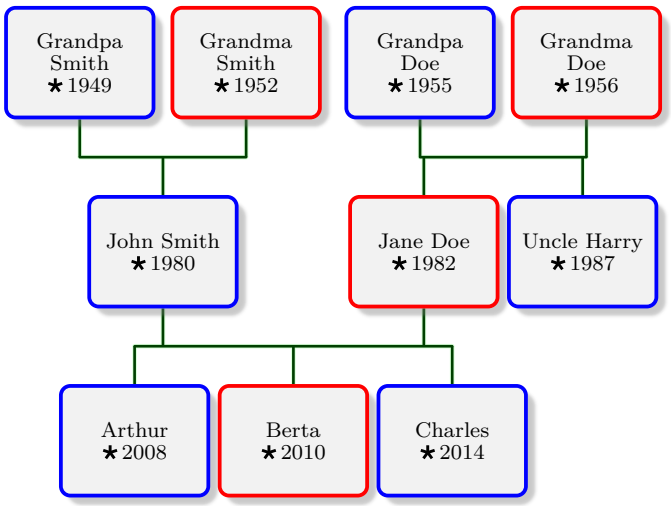
```
\begin{tikzpicture}
\genealogytree[template=signpost]
{
  parent{
    g[male]{proband}
    c[female]{sister}
    c[male]{brother}
    parent{
      g[male]{father}
      p[male]{grandfather}
      p[female]{grandmother}
    }
    p[female]{mother}
  }
}
\end{tikzpicture}
```



`\genealogytreeinput[`⟨*options*⟩`]{`⟨*file name*⟩`}`

Uses the content of the file denoted by ⟨*file name*⟩ to create a `\genealogytree` → P. 55 with the given ⟨*options*⟩. See Section 14.1 on page 287 for the file of the following example.

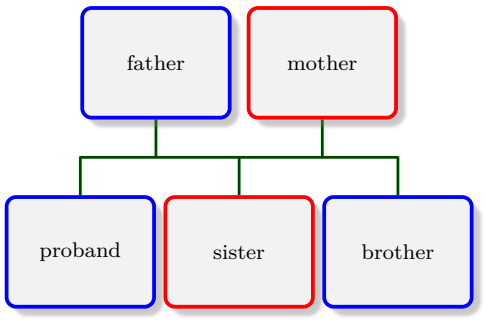
```
\begin{tikzpicture}
\genealogytreeinput[template=signpost]{example.option.graph}
\end{tikzpicture}
```



```
\begin{genealogypicture}[⟨options⟩]
  ⟨tree content⟩
\end{genealogypicture}
```

This is a shortcut combination of one `\genealogytree` → P. 55 inside a `tikzpicture`. For ⟨*options*⟩ and ⟨*tree content*⟩ see `\genealogytree` → P. 55. This environment allows more compact source code, but one cannot combine several trees and adding additional Ti*k*Z commands has to be done by `/gtr/tikzpicture` → P. 112 or `/gtr/after tree` → P. 113.

```
\begin{genealogypicture}
[template=signpost]
  parent{
    g[male]{proband}
    c[female]{sister}
    c[male]{brother}
    p[male]{father}
    p[female]{mother}
  }
\end{genealogypicture}
```




```
\begin{exgenealogypicture}[⟨externalization options⟩]{⟨name⟩}[⟨options⟩]
  ⟨tree content⟩
\end{exgenealogypicture}
```

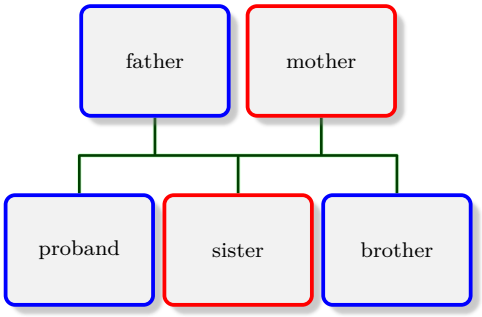
This is an externalized version of `genealogypicture` using the `external` library of the package `tcolorbox` [3]. The picture is drawn by automatic compilation of an external file denoted by ⟨*name*⟩ (usually prefixed by a directory or string). Afterwards, the created pdf image is included into the main document. As long as the ⟨*tree content*⟩ and the ⟨*options*⟩ are not changed, the external file is not compiled again which saves overall compilation time. The process can be controlled by ⟨*externalization options*⟩, see [3].
For a detailed application example, see Section 2.6 on page 49.

3.2 Using Tree Options

`\gtrset{`⟨*options*⟩`}`

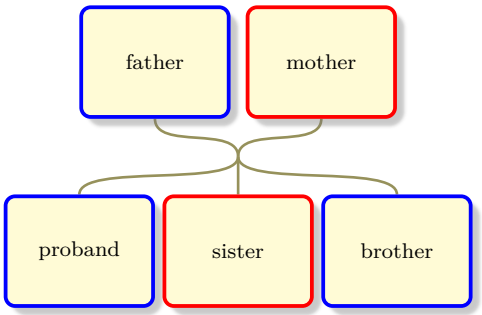
Sets ⟨*options*⟩ for every following `\genealogytree` → P. 55 inside the current TEX group. These ⟨*options*⟩ are `pgf` keys with the key tree path `/gtr/` and they are described in the following.

```
% Setting options for the following
\gtrset{template=signpost}

\begin{tikzpicture}
\genealogytree
{
  parent{
    g[male]{proband}
    c[female]{sister}
    c[male]{brother}
    p[male]{father}
    p[female]{mother}
  }
}
\end{tikzpicture}
```



Another important field of application for `\gtrset` is to create own styles for later usage.

```
% Setting options for the following
\gtrset{mytree/.style={
  template=signpost,
  box={colback=yellow!20},
  edges={swing,no background,
    foreground=yellow!50!black},
  }}

\begin{tikzpicture}
\genealogytree[mytree]% own style
{
  parent{
    g[male]{proband}
    c[female]{sister}
    c[male]{brother}
    p[male]{father}
    p[female]{mother}
  }
}
\end{tikzpicture}
```



`\gtrkeysappto{`⟨*hook*⟩`}{`⟨*key list*⟩`}`

Auxiliary macro which appends a ⟨*key list*⟩ (options) to a ⟨*hook*⟩ macro which may already contain a key list.

`\gtrkeysgappto{`⟨*hook*⟩`}{`⟨*key list*⟩`}`

Auxiliary macro which globally appends a ⟨*key list*⟩ (options) to a ⟨*hook*⟩ macro which may already contain a key list.

3.3 Accessing Information inside Nodes

Inside the node content, there are several processing informations available which can be used for debugging or steering the output. Also see Section 11.4 on page 243 for displaying these values.

\gtrnodetype
Holds the node type g, p, or c.

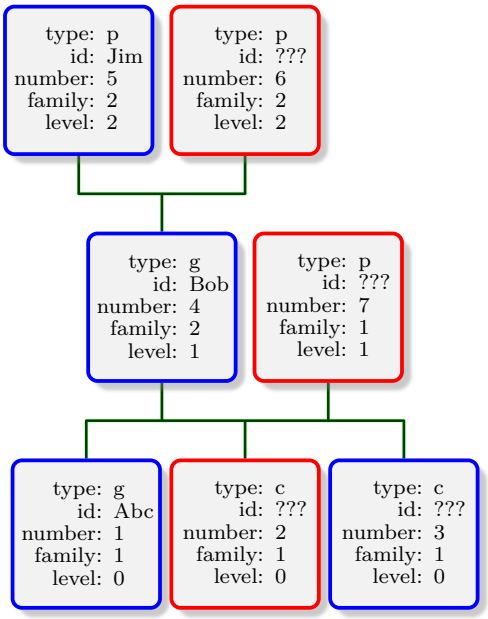
\gtrnodeid
Holds the /gtr/id → P. 90 value of the node.

\gtrnodenumber
Holds the internal node number.

\gtrnodefamily
Holds the internal family number this node belongs to.

\gtrnodelevel
Holds the tree level number this node belongs to.

```
\begin{tikzpicture}
\genealogytree[
  template=signpost,
  level size=2cm,
  content interpreter content={
    \begin{tabular}{@{}r@{: }l@{}}
    type   & \gtrnodetype\\
    id     & \gtrnodeid\\
    number & \gtrnodenumber\\
    family & \gtrnodefamily\\
    level  & \gtrnodelevel\\
    \end{tabular}}
]{
  parent{
    g[male,id=Abc]{}
    c[female]{}
    c[male]{}
    parent{
      g[male,id=Bob]{}
      p[male,id=Jim]{}
      p[female]{}
    }
    p[female]{}
  }
}
\end{tikzpicture}
```



\gtrifnodeid{⟨true⟩}{⟨false⟩}
Expands to ⟨true⟩, if /gtr/id → P. 90 was set, and to ⟨false⟩ otherwise.

\gtrifgnode{⟨true⟩}{⟨false⟩}
Expands to ⟨true⟩, if the node type is g, and to ⟨false⟩ otherwise.

\gtrifcnode{⟨true⟩}{⟨false⟩}
Expands to ⟨true⟩, if the node type is c, and to ⟨false⟩ otherwise.

\gtrifpnode{⟨true⟩}{⟨false⟩}
Expands to ⟨true⟩, if the node type is p, and to ⟨false⟩ otherwise.

`\gtrifroot{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if the node is the root node of a `parent` tree or of a `child` tree, and to ⟨*false*⟩ otherwise. For a `sandclock` tree, it expands always to ⟨*false*⟩.

`\gtrifleaf{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if the node type is `c` or `p` or if the node is the root node of a `parent` tree or of a `child` tree, and to ⟨*false*⟩ otherwise. Note that ⟨*false*⟩ is set for all `g` nodes with the root node as an exception, even if the node does not have a parent or a child. Also note that a root node is intentionally considered to be a leaf also.

`\gtrifchild{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if the node type is `c` or is `g` in a `parent` family or is `g` but not root in a `child` family, and to ⟨*false*⟩ otherwise.

`\gtrifparent{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if the node type is `p` or is `g` in a `child` family or is `g` but not root in a `parent` family, and to ⟨*false*⟩ otherwise.

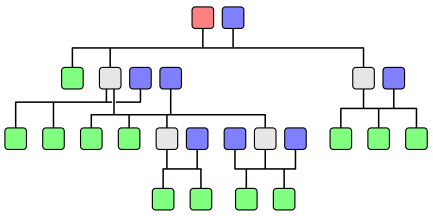
`\gtrifleafchild{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if `\gtrifleaf` and `\gtrifchild` are both true, and to ⟨*false*⟩ otherwise.

`\gtrifleafparent{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if `\gtrifleaf` and `\gtrifparent` are both true, and to ⟨*false*⟩ otherwise.

```
\begin{tikzpicture}
\genealogytree[
template=tiny boxes,
box={code={%
  \gtrifroot{\tcbset{colback=red!50}}{%
    \gtrifleafparent{\tcbset{colback=blue!50}}{%
       \gtrifleafchild{\tcbset{colback=green!50}}{}%
    }%
  }%
}}
]{
  child{
    g-p-c-
    child{g-p-c-c-
      union{p-c-c-
        child{g-p-c-c-}
        child{p-g-p-c-c-}
      }
    }
    child{g-p-c-c-c-}
  }
}
\end{tikzpicture}
```

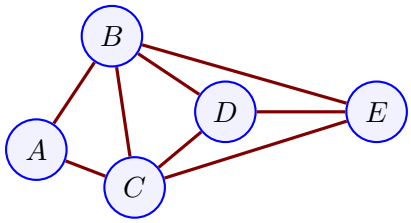


4

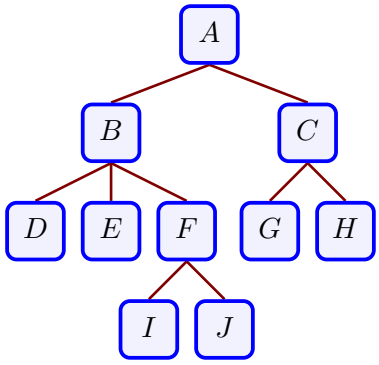
Graph Grammar

4.1 Graph Structure

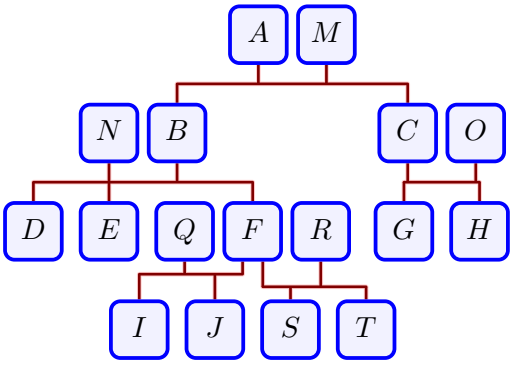
In graph theory, a *graph* is defined by a set of vertices (or nodes) and a set of edges connecting these vertices. A general graph structure would certainly allow to depict genealogy data, but building and displaying such a general graph is not supported by this LaTeX package.



An ordinary *tree* structure is a specialized (directed) graph which has a *root* node as starting point. Every node may have one or more descendant nodes. In this relationship, the first node is called *parent* and the second is called *child*. Such tree structures are heavily used for many applications. Also, there exist excellent LaTeX packages to display such structures. Such *tree* structures can also be used for several kinds of genealogy type diagrams, but, by definition, they miss the core element of genealogy: the *family* consisting of two parents and several childs.



The graph structure used by the `genealogytree` package is intended to put the *family* as a set of *parent* and *child* nodes in the foreground. Every *family* is allowed to have more than one *parent* and more than one *child*. The interconnection between the *parent* and *child* nodes of a family is not considered to be bilateral between pairs of nodes, but to be multilateral between all nodes of the family. From the idea, a node is connected not to another node, but to one or more families. Still, there apply strong restrictions on the set of possible graphs, because graphs have to be reasonable processable and presentable. The restrictions are realized by the following graph grammar. In the following, the resulting graphs are called *genealogy trees*.



A *family* is a set of *parent* and *child* nodes which has to contain exactly one *genealogy* node (`g` node). Therefore, a family can have three types of nodes:

- `c` nodes are *child* nodes to a family, see Section 4.6 on page 71,
- `p` nodes are *parent* nodes to a family, see Section 4.7 on page 71,
- `g` nodes are usually *child* nodes to one family and *parent* nodes another family or even several families, see Section 4.8 on page 71.

A genealogy tree can have following types of families:

- `parent`: A *parent* family may contain other `parent` families. Trees with this construction grow into *ancestor direction*, see Section 4.2 on page 63,
- `child`: A *child* family may contain other `child` families or `union` families. Trees with this construction grow into *descendant direction*, see Section 4.3 on page 65,
- `union`: A *union* ties a second child-type family to a `g` node as parent of this family, see Section 4.4 on page 67,
- `sandclock`: A *sandclock* connects ancestors and descendants starting from a single proband, see Section 4.5 on page 69.

As will be documented on the following pages, the graph input data is strongly hierarchically organized. Each element is allowed to have specific sub-elements. The starting point is the *root* element which is the top element inside `\genealogytree`[→ P. 55]. The *root* of a parsable graph is one of the following:

- a `parent` (for ancestor graphs), see Section 4.2 on the facing page,
- a `child` (for descendant graphs), see Section 4.3 on page 65,
- a `sandclock` (for mixed ancestor/descendant graphs), see Section 4.5 on page 69.

4.2 Subgraph 'parent'

A `parent` subgraph is a family where the `g` node acts as a child. This family may have arbitrary child and parent leaves. Also, this family may have arbitrary `parent` subgraphs.

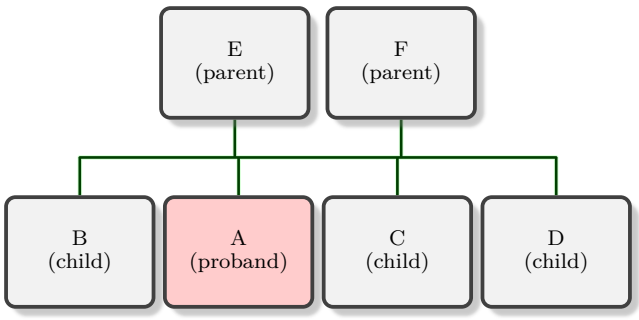
Syntax for a 'parent' subgraph

```
parent[⟨parent options⟩]{
  g[⟨node options⟩]{⟨node content⟩}          mandatory; exactly once
  c[⟨node options⟩]{⟨node content⟩}          optional; zero or many times
  p[⟨node options⟩]{⟨node content⟩}          optional; zero or many times
  parent[⟨parent options⟩]{⟨subtree content⟩} optional; zero or many times
  input{⟨file name⟩}                         optional; zero or many times
  insert{⟨csname⟩}                           optional; zero or many times
}
```

'g', 'c', 'p', 'parent', 'input' may appear in arbitrary order.

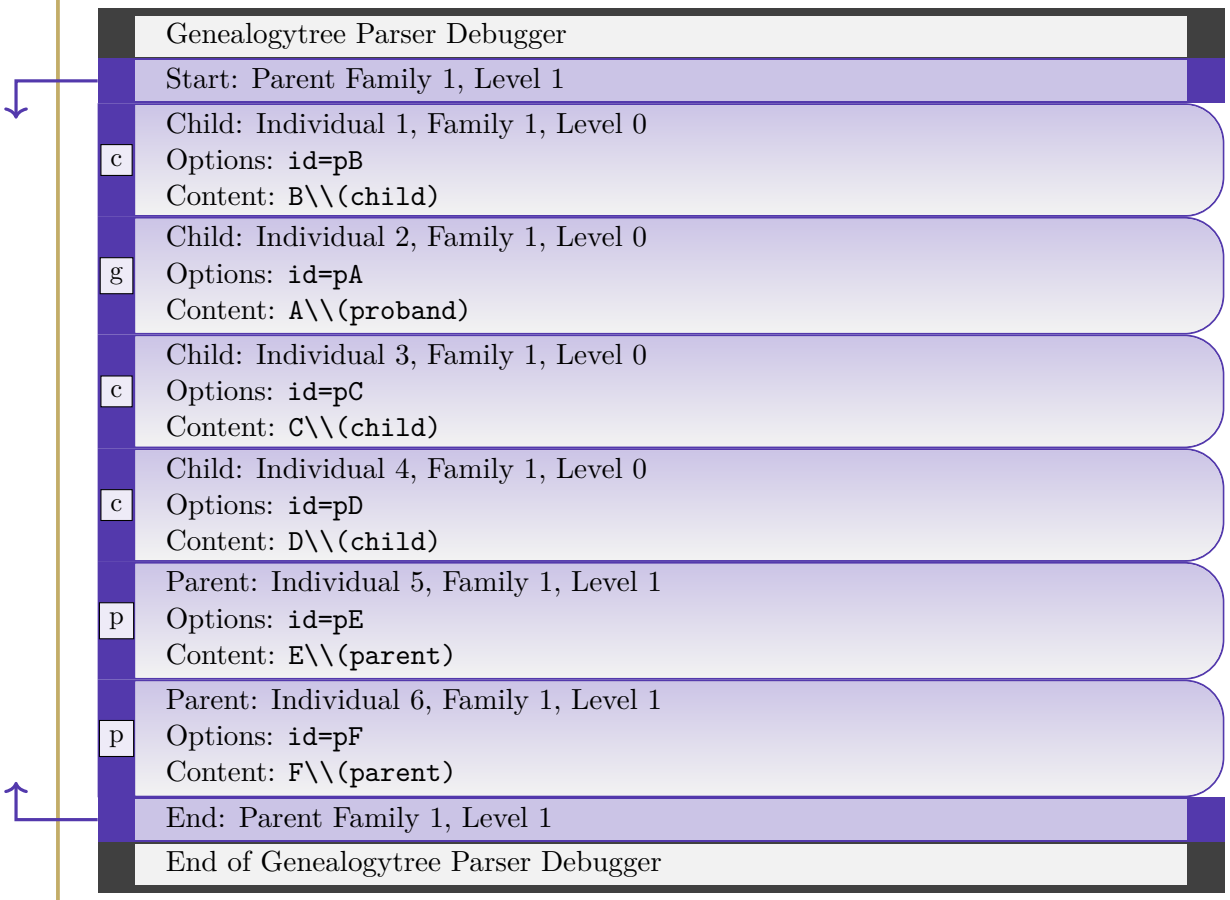
The optional ⟨*parent options*⟩ can be settings for the current family or the whole subgraph. See Chapter 5 on page 75 and especially Section 5.6 on page 102 and Section 5.7 on page 105 for feasible options.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for node={pA}{box={colback=red!20!white}}]
{
  parent{
    c[id=pB]{B\\(child)}
    g[id=pA]{A\\(proband)}
    c[id=pC]{C\\(child)}
    c[id=pD]{D\\(child)}
    p[id=pE]{E\\(parent)}
    p[id=pF]{F\\(parent)}
  }
}
\end{tikzpicture}
```



\gtrparserdebug → P. 228 can help to detect structural errors. Here, we get:

```
\gtrparserdebug{
  parent{
    c[id=pB]{B\\(child)}
    g[id=pA]{A\\(proband)}
    c[id=pC]{C\\(child)}
    c[id=pD]{D\\(child)}
    p[id=pE]{E\\(parent)}
    p[id=pF]{F\\(parent)}
  }
}
```



4.3 Subgraph 'child'

A `child` subgraph is a family where the `g` node acts as a parent. This family may have arbitrary child and parent leaves. Also, this family may have arbitrary `child` and `union` subgraphs.

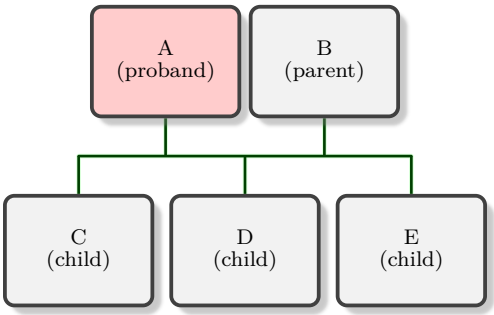
Syntax for a 'child' subgraph

`child[`⟨*child options*⟩`]{`	
`  g[`⟨*node options*⟩`]{`⟨*node content*⟩`}`	*mandatory; exactly once*
`  c[`⟨*node options*⟩`]{`⟨*node content*⟩`}`	*optional; zero or many times*
`  p[`⟨*node options*⟩`]{`⟨*node content*⟩`}`	*optional; zero or many times*
`  child[`⟨*child options*⟩`]{`⟨*subtree content*⟩`}`	*optional; zero or many times*
`  union[`⟨*union options*⟩`]{`⟨*subtree content*⟩`}`	*optional; zero or many times*
`  input{`⟨*file name*⟩`}`	*optional; zero or many times*
`  insert{`⟨*csname*⟩`}`	*optional; zero or many times*
`}`	

'g', 'c', 'p', 'child', 'union', 'input' may appear in arbitrary order.

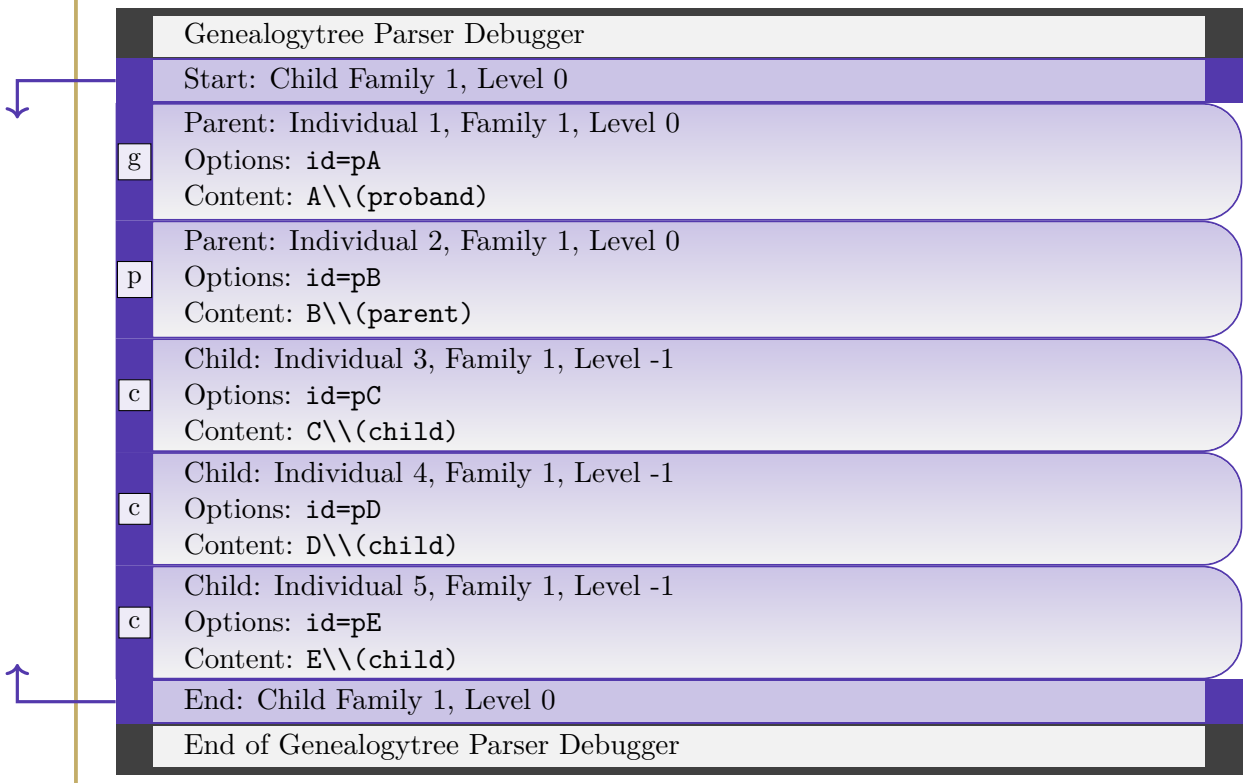
The optional ⟨*child options*⟩ can be settings for the current family or the whole subgraph. See Chapter 5 on page 75 and especially Section 5.6 on page 102 and Section 5.7 on page 105 for feasible options.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for node={pA}{box={colback=red!20!white}}]
{
  child{
    g[id=pA]{A\\(proband)}
    p[id=pB]{B\\(parent)}
    c[id=pC]{C\\(child)}
    c[id=pD]{D\\(child)}
    c[id=pE]{E\\(child)}
  }
}
\end{tikzpicture}
```



\gtrparserdebug → P. 228 can help to detect structural errors. Here, we get:

```
\gtrparserdebug{
  child{
    g[id=pA]{A\\(proband)}
    p[id=pB]{B\\(parent)}
    c[id=pC]{C\\(child)}
    c[id=pD]{D\\(child)}
    c[id=pE]{E\\(child)}
  }
}
```



4.4 Subgraph 'union'

A `union` subgraph is a family without a `g` node. The `g` node (parent) is inherited from an embedding `child` family. A `union` family may have arbitrary child and parent leaves. Also, this family may have arbitrary `child` subgraphs.

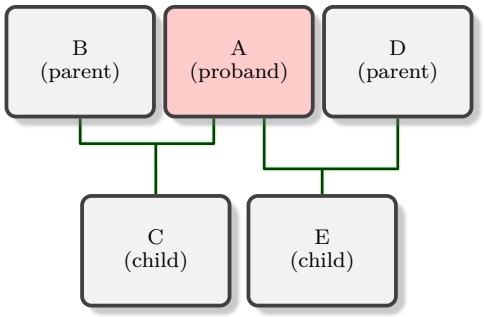
Syntax for a 'union' subgraph

`union[`⟨*union options*⟩`]{`	
`  c[`⟨*node options*⟩`]{`⟨*node content*⟩`}`	*optional; zero or many times*
`  p[`⟨*node options*⟩`]{`⟨*node content*⟩`}`	*optional; zero or many times*
`  child[`⟨*child options*⟩`]{`⟨*subtree content*⟩`}`	*optional; zero or many times*
`  input{`⟨*file name*⟩`}`	*optional; zero or many times*
`  insert{`⟨*csname*⟩`}`	*optional; zero or many times*
`}`	

'c', 'p', 'child', 'input' may appear in arbitrary order.

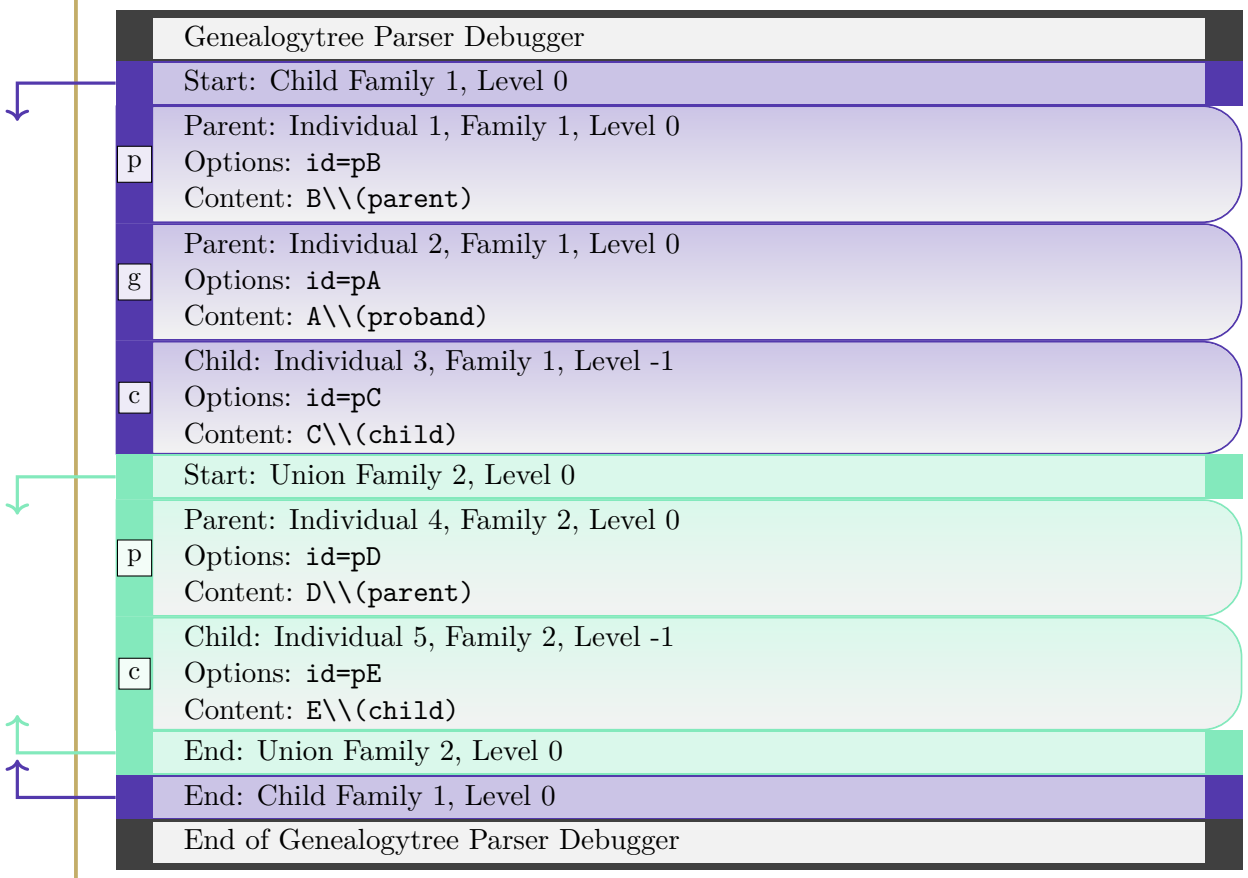
The optional ⟨*child options*⟩ can be settings for the current family or the whole subgraph. See Chapter 5 on page 75 and especially Section 5.6 on page 102 and Section 5.7 on page 105 for feasible options. As a special case for unions, note that the `g` node of the embedding `child` family will not be affected by these options.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for node={pA}{box={colback=red!20!white}}]
{
  child{
    p[id=pB]{B\\(parent)}
    g[id=pA]{A\\(proband)}
    c[id=pC]{C\\(child)}
    union{
      p[id=pD]{D\\(parent)}
      c[id=pE]{E\\(child)}
    }
  }
}
\end{tikzpicture}
```



\gtrparserdebug → P. 228 can help to detect structural errors. Here, we get:

```
\gtrparserdebug{
  child{
    p[id=pB]{B\\(parent)}
    g[id=pA]{A\\(proband)}
    c[id=pC]{C\\(child)}
    union{
      p[id=pD]{D\\(parent)}
      c[id=pE]{E\\(child)}
    }
  }
}
```



4.5 Subgraph 'sandclock'

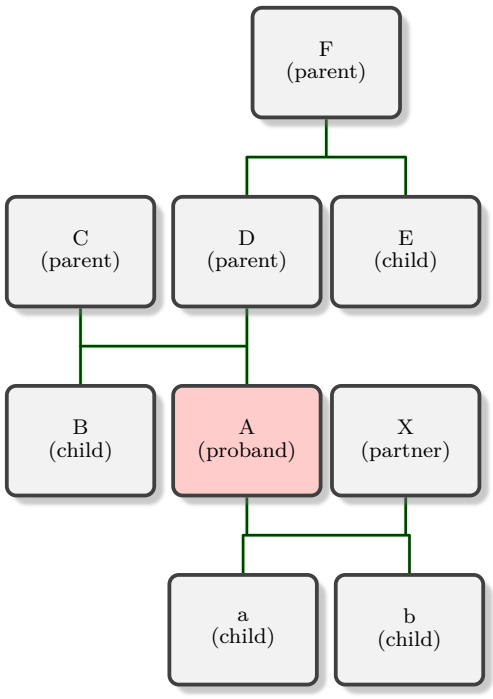
A `sandclock` subgraph is a family without a `g` node. The `g` node (child) is inherited from an embedded `child` family. A `sandclock` family may have arbitrary child and parent leaves. Also, this family must have at least one `child` subgraph and may have arbitrary `parent` subgraphs.

Syntax for a 'sandclock' subgraph

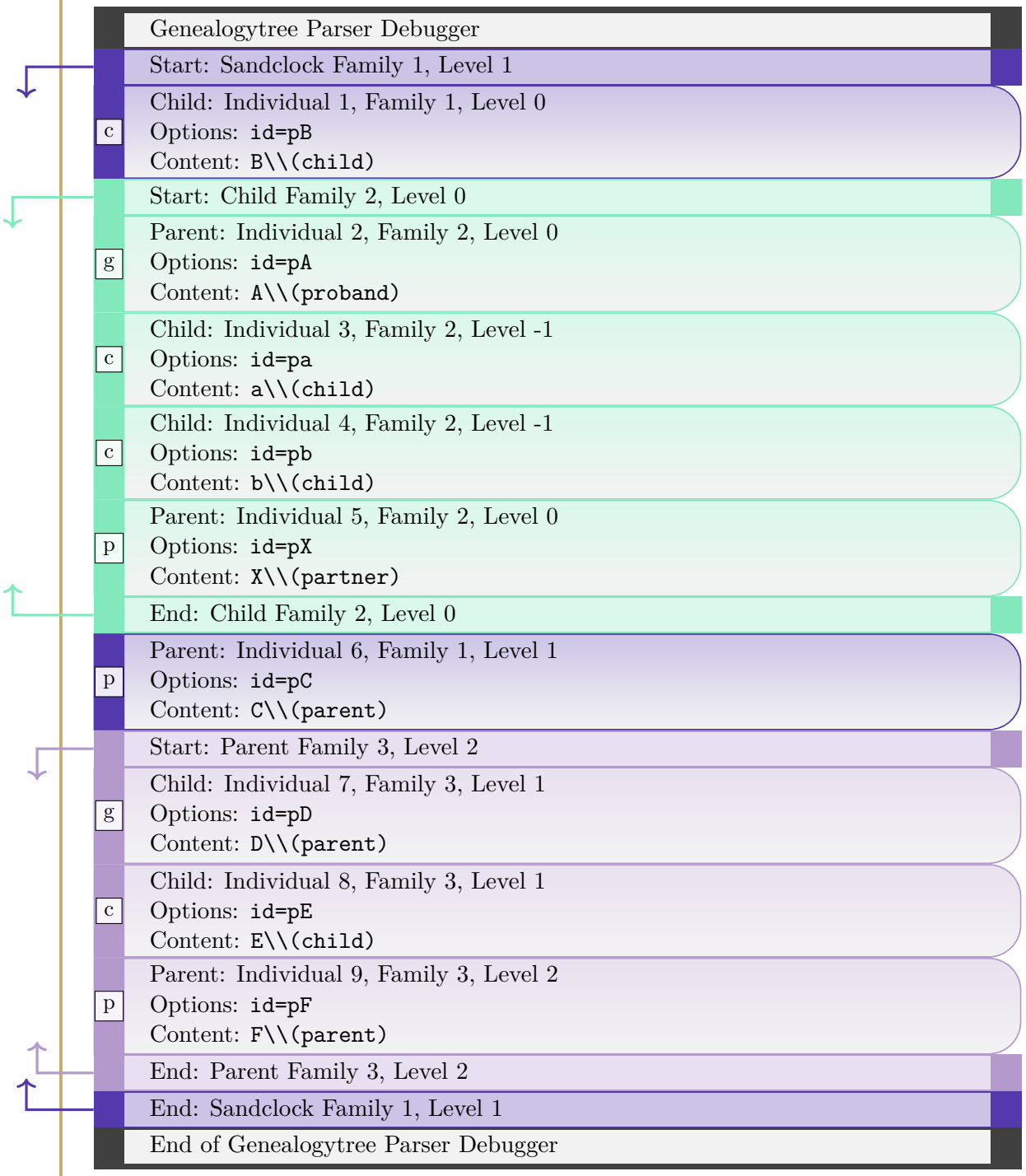
```
sandclock[⟨sandclock options⟩]{
  c[⟨node options⟩]{⟨node content⟩}              optional; zero or many times
  p[⟨node options⟩]{⟨node content⟩}              optional; zero or many times
  child[⟨child options⟩]{⟨subtree content⟩}      mandatory; one or many times
  parent[⟨parent options⟩]{⟨subtree content⟩}    optional; zero or many times
  input{⟨file name⟩}                            optional; zero or many times
  insert{⟨csname⟩}                              optional; zero or many times
}
```

'c', 'p', 'child', 'parent', 'input' may appear in arbitrary order.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for node={pA}{box={colback=red!20!white}}]
{
  sandclock{
    c[id=pB]{B\\(child)}
    child
    {
      g[id=pA]{A\\(proband)}    c[id=pa]{a\\(child)}
      c[id=pb]{b\\(child)}      p[id=pX]{X\\(partner)}
    }
    p[id=pC]{C\\(parent)}
    parent{
      g[id=pD]{D\\(parent)}     c[id=pE]{E\\(child)}     p[id=pF]{F\\(parent)}
    }
  }
}
\end{tikzpicture}
```



```
\gtrparserdebug{
  sandclock{
    c[id=pB]{B\\(child)}
    child
    {
      g[id=pA]{A\\(proband)}    c[id=pa]{a\\(child)}
      c[id=pb]{b\\(child)}      p[id=pX]{X\\(partner)}
    }
    p[id=pC]{C\\(parent)}
    parent{
      g[id=pD]{D\\(parent)}     c[id=pE]{E\\(child)}     p[id=pF]{F\\(parent)}
    }
  }
}
```



4.6 Node 'c'

The c (child) node is a leaf node which is child to a family.

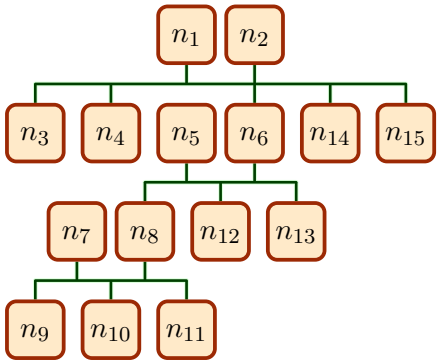
Syntax for a 'c' node

c[⟨*node options*⟩]{⟨*node content*⟩}

For the optional ⟨*node options*⟩, see Chapter 5 on page 75 and especially Section 5.5 on page 93.

The ⟨*node content*⟩ can be any text to be displayed inside the node. This ⟨*node content*⟩ can also be processed before displaying, see Chapter 6 on page 127 and especially Chapter 7 on page 151 for database processing. Also, the ⟨*node content*⟩ can be completely ignored for processing. In this case, one can use c{} or even shorter c⟨*token*⟩ for the node.

```
\begin{genealogypicture}[
  template=formal graph,
  content interpreter content=
    {n_{\gtrnodenumber}},
]
  child{ g-p-c-c-
    child{ p-g-
      child{ p-g-c-c-c- }
      c-c-
    }
    c-c-
  }
\end{genealogypicture}
```



4.7 Node 'p'

The p (parent) node is a leaf node which is parent to a family.

Syntax for a 'p' node

p[⟨*node options*⟩]{⟨*node content*⟩}

For the optional ⟨*node options*⟩, see Chapter 5 on page 75 and especially Section 5.5 on page 93. For ⟨*node content*⟩, see Section 4.6 on page 71.

4.8 Node 'g'

The g (genealogy) node is an interconnecting individual which is member of at least two families. For one family it is a child, for another one it is a parent.

Syntax for a 'g' node

g[⟨*node options*⟩]{⟨*node content*⟩}

For the optional ⟨*node options*⟩, see Chapter 5 on page 75 and especially Section 5.5 on page 93. For ⟨*node content*⟩, see Section 4.6 on page 71.

4.9 Data 'input'

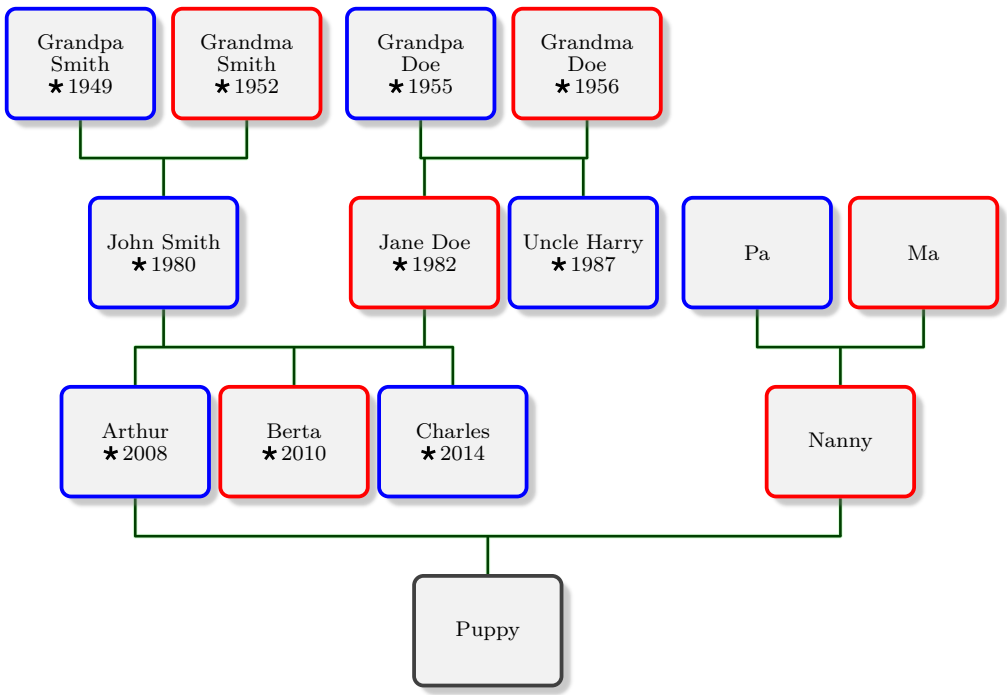
Feasible subgraphs may be read from external files using the `input` command at places where such subgraphs are expected.

Syntax for data 'input'

`input{`⟨*file name*⟩`}`

The following example reads a `parent` subgraph from a file. See Section 14.1 on page 287 for the file contents.

```
\begin{tikzpicture}
\genealogytree[template=signpost]
{
  parent{
    g{Puppy}
    input{example.option.graph}
    parent
    {
      g[female]{Nanny}
      p[male]{Pa}
      p[female]{Ma}
    }
  }
}
\end{tikzpicture}
```



4.10 Control Sequence 'insert'

Feasible subgraphs may be inserted from control sequences using the `insert` command at places where such subgraphs are expected.

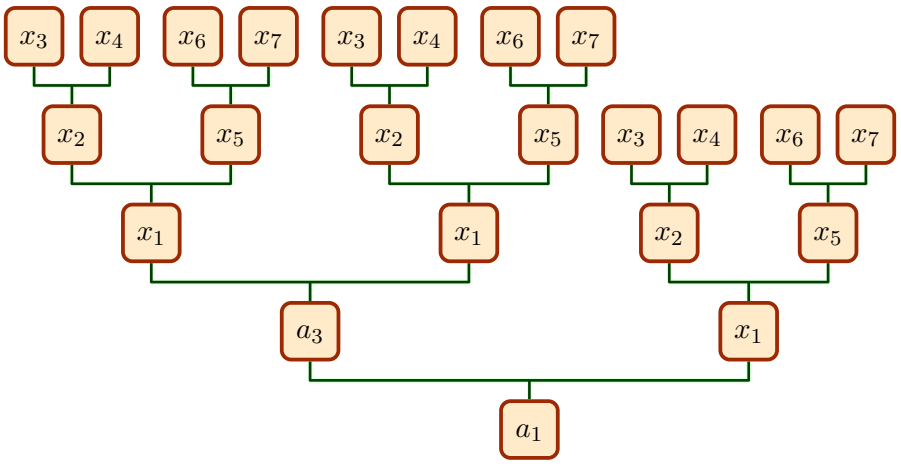
Syntax for data 'insert'

`insert{`⟨*csname*⟩`}`

⟨*csname*⟩ is the name of a control sequence without the leading backslash '\'. This control sequence has to be a parameterless macro whose replacement text is a feasible subgraph.

The following example creates such a macro `\mytest`:

```
\newcommand{\mytest}{
  parent{ g{x_1}
    parent{ g{x_2} p{x_3} p{x_4} }
    parent{ g{x_5} p{x_6} p{x_7} }
}}

\begin{tikzpicture}
\genealogytree[template=formal graph]
{
  parent{
    g{a_1}
    parent{
      g{a_3}
      insert{mytest}
      insert{mytest}
    }
    insert{mytest}
  }
}
\end{tikzpicture}
```



5

Option Setting

For the ⟨*options*⟩ in \genealogytree→ P. 55, genealogypicture→ P. 57, and \gtrset→ P. 58, keys with pgf syntax can be applied as documented in the following. The key tree path /gtr/ is not to be used inside these macros. It is easy to add your own style keys using the syntax for pgf keys, see [4].

Some of the following examples use a standard graph file which is documented in Section 14.1 on page 287.

5.1 Option Priorities

This section can be skipped on first reading. Option priorities are more or less natural. This section can be consulted later in case of doubt.

Options for the graph drawing can be set at several spots inside the code using the pgf key-value syntax:

- as parameter of \gtrset→ P. 58 for setting (global) options,
- as optional parameter of \genealogytree→ P. 55 or genealogypicture→ P. 57,
- as optional parameter of a family identifier like parent or child,
- as optional parameter of a node identifier like g, p, or c.

Depending on where the options are given, they are treated with different priority. If an option is given several times with the same priority, the last one wins.

- For options like /gtr/pivot→ P. 95, an option setting with higher priority overwrites an option setting with lower priority.
- For options like /gtr/box→ P. 96, an option setting with higher appends to an option setting with lower priority. Thus, /gtr/box→ P. 96 options which are not overwritten, stay active.

5.1.1 Option Priorities for Nodes

Example: Priorities for setting `box` options to a node with `id=A`

```
\gtrset{
  ... % priorities identical to options for \genealogytree
}
\genealogytree[
  options for node={A}{box={...}},                  % priority (1) highest
  options for family={fam_a}{box={...}},            % priority (5)
  options for subtree={fam_a}{box={...}},           % priority (9)
  level 2/.style={
    node={box={...}},                               % priority (3)
    family={box={...}},                             % priority (7)
    subtree={box={...}}},                           % priority (11)
  level/.code={\ifnum#1=2\relax\gtrset{
    node={box={...}},                               % priority (4)
    family={box={...}},                             % priority (8)
    subtree={box={...}}}\fi},                       % priority (12)
  box={...},                                        % priority (13) lowest
  ...
  ]%
{
  ...
  parent[  id=fam_a,     % family with id 'fam_a'
           family={box={...}},                      % priority (6)
           subtree={box={...}},                     % priority (10)
       ]
    {
      ...
      p[id=A,           % node with id 'A'
        box={...}]{A}                               % priority (2)
      ...
    }
  ...
}
```

The priorities for options regarding nodes

1. `/gtr/options for node` → P. 93 has the highest priority. The node has to be identified by a given `/gtr/id` → P. 90. `/gtr/options for node` → P. 93 should be given using `\gtrset` → P. 58 or as option of `\genealogytree` → P. 55.
2. Optional parameter of a node identifier like `g`, `p`, or `c`.
3. Option `/gtr/node` → P. 94 inside `/gtr/level n` → P. 108.
4. Option `/gtr/node` → P. 94 inside `/gtr/level` → P. 107.
5. `/gtr/options for family` → P. 102; the family has to be identified by a given `/gtr/id` → P. 90.
6. `/gtr/family` → P. 103 as optional parameter of a family identifier like `parent` or `child`.
7. Option `/gtr/family` → P. 103 inside `/gtr/level n` → P. 108.
8. Option `/gtr/family` → P. 103 inside `/gtr/level` → P. 107.
9. `/gtr/options for subtree` → P. 105; the subtree has to be identified by a given `/gtr/id` → P. 90.
10. `/gtr/subtree` → P. 106 as optional parameter of a family identifier like `parent` or `child`.
11. Option `/gtr/subtree` → P. 106 inside `/gtr/level n` → P. 108.
12. Option `/gtr/subtree` → P. 106 inside `/gtr/level` → P. 107.
13. Setting as parameter of `\genealogytree` → P. 55 or `\gtrset` → P. 58 has the lowest priority.

5.1.2 Option Priorities for Families

Example: Priorities for setting `edges` options to a family with `id=fam_a`

```
\gtrset{
  ... % priorities identical to options for \genealogytree
}
\genealogytree[
  options for family={fam_a}{edges={...}},                % priority (1) highest
  options for subtree={fam_a}{edges={...}},               % priority (5)
  level 2/.style={family={edges={...}}                    % priority (3)
                  subtree={edges={...}}},                 % priority (7)
  level/.code={\ifnum#1=2\relax%
    \gtrset{family={edges={...}},                         % priority (4)
            subtree={edges={...}}}                        % priority (8)
    \fi},
  edges={...},                                            % priority (9) lowest
  ...
  ]%
{
  ...
  parent[  id=fam_a,     % family with id 'fam_a'
           family={edges={...}},                          % priority (2)
           subtree={edges={...}},                         % priority (6)
        ]
    {
      ...
    }
  ...
}
```

The priorities for options regarding families

1. `/gtr/options for family` → P. 102 has the highest priority. The family has to be identified by a given `/gtr/id` → P. 90. `/gtr/options for family` → P. 102 should be given using `\gtrset` → P. 58 or as option of `\genealogytree` → P. 55.
2. Optional `/gtr/family` → P. 103 parameter of a family identifier like `parent` or `child`.
3. Option `/gtr/family` → P. 103 inside `/gtr/level n` → P. 108.
4. Option `/gtr/family` → P. 103 inside `/gtr/level` → P. 107.
5. `/gtr/options for subtree` → P. 105; the subtree has to be identified by a given `/gtr/id` → P. 90.
6. Optional `/gtr/subtree` → P. 106 parameter of a family identifier like `parent` or `child`.
7. Option `/gtr/subtree` → P. 106 inside `/gtr/level n` → P. 108.
8. Option `/gtr/subtree` → P. 106 inside `/gtr/level` → P. 107.
9. Setting as parameter of `\genealogytree` → P. 55 or `\gtrset` → P. 58 has the lowest priority.

5.2 Graph Growth Setting (Time Flow)

A genealogy tree may grow in one of four directions. This /gtr/timeflow setting is valid for the whole graph, but two graphs with different growth setting may be joined together.

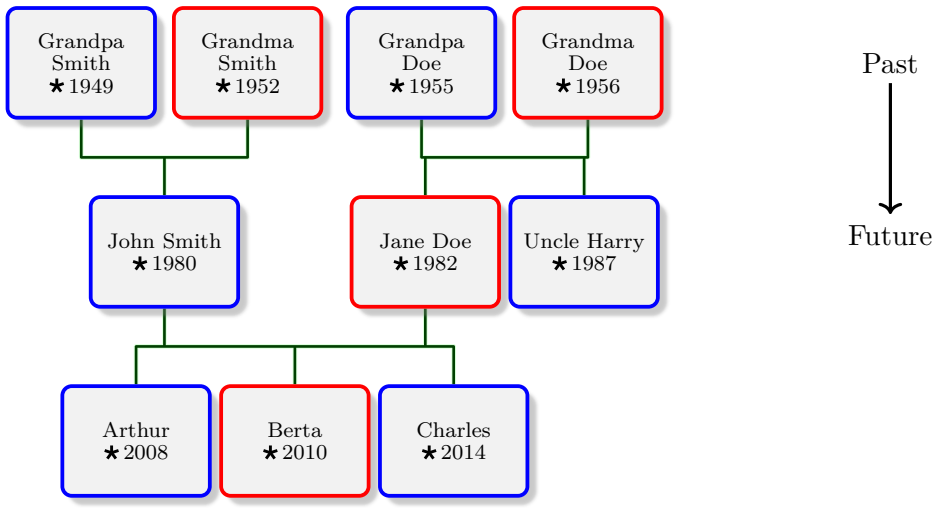
/gtr/timeflow=⟨*direction*⟩ (no default, initially down)

The /gtr/timeflow key controls the growing direction of a given graph. It is always used to place the generations according to this value. If the ⟨*direction*⟩ is set to down, a child graph will grow down, but a parent graph will grow up. Feasible values are:

- down
- up
- left
- right

timeflow=down

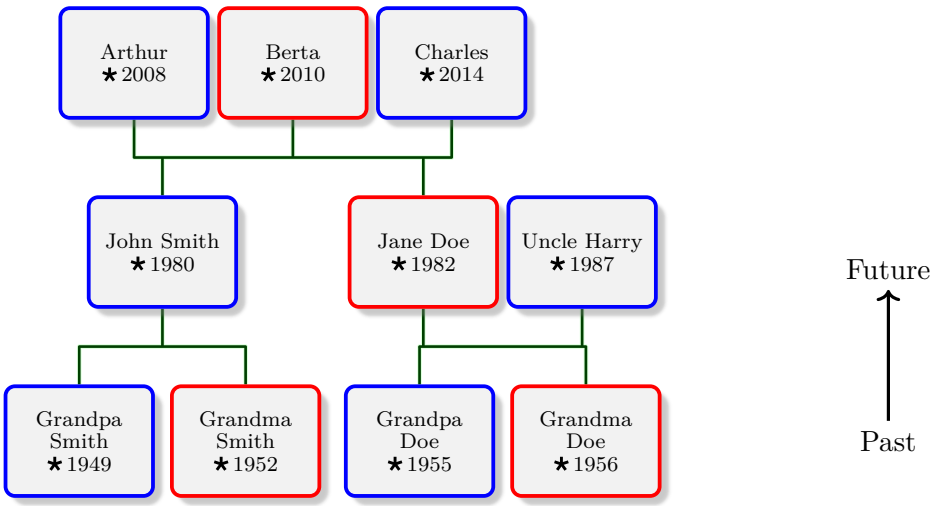
```
\begin{tikzpicture}
\genealogytree[template=signpost,timeflow=down]
  {input{example.option.graph}}
\node at ([xshift=4cm]GmDo1956) (past) {Past};
\draw[very thick,->] (past) -- +(0,-2) node[below] {Future};
\end{tikzpicture}
```



See Section 14.1 on page 287 for the included example graph file.

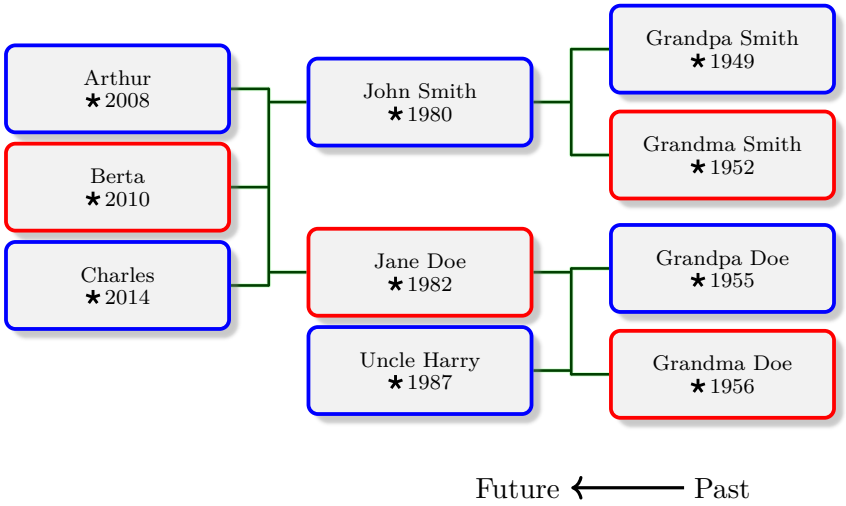
timeflow=up

```
\begin{tikzpicture}
\genealogytree[template=signpost,timeflow=up]
  {input{example.option.graph}}
\node at ([xshift=4cm]GmDo1956) (past) {Past};
\draw[very thick,->] (past) -- +(0,2) node[above] {Future};
\end{tikzpicture}
```



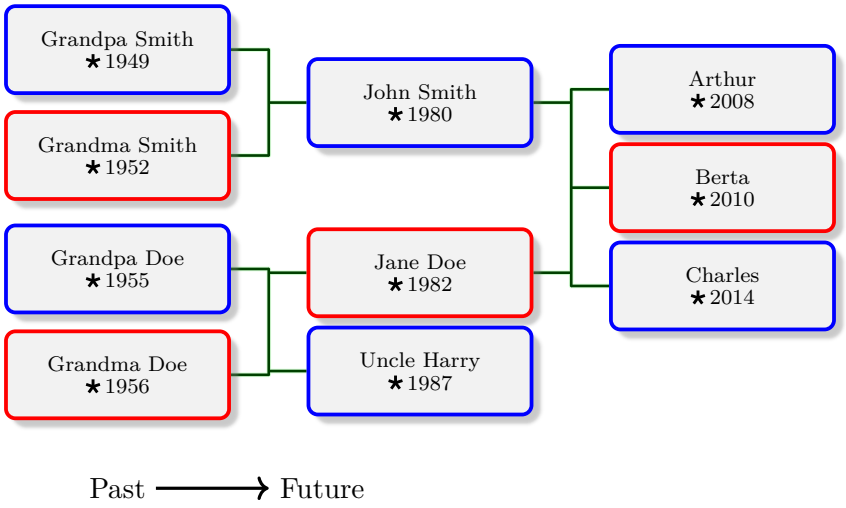
timeflow=left

```
\begin{tikzpicture}
\genealogytree[template=signpost,timeflow=left,node size=1.2cm,level size=3cm]
  {input{example.option.graph}}
\node at ([yshift=-1.5cm]GmDo1956) (past) {Past};
\draw[very thick,->] (past) -- +(-2,0) node[left] {Future};
\end{tikzpicture}
```



timeflow=right

```
\begin{tikzpicture}
\genealogytree[template=signpost,timeflow=right,node size=1.2cm,level size=3cm]
  {input{example.option.graph}}
\node at ([yshift=-1.5cm]GmDo1956) (past) {Past};
\draw[very thick,->] (past) -- +(2,0) node[right] {Future};
\end{tikzpicture}
```

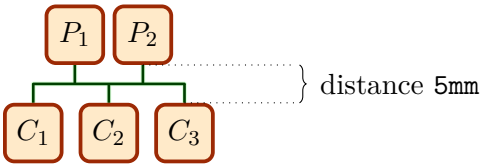


5.3 Graph Geometry

The following geometry settings are usually set for the whole graph, but they can be set for every `/gtr/level` → P. 107 separately. Inside a level, they are fixed.

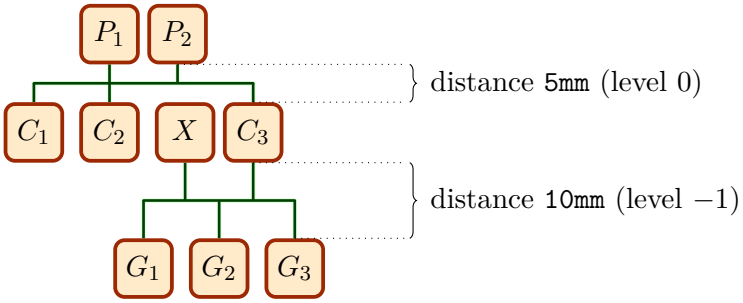
`/gtr/level distance`=⟨*length*⟩ (no default, initially `5mm`)

The given ⟨*length*⟩ defines the distance between two following generations. This distance can be set in dependency of the `/gtr/level` → P. 107.



The `/gtr/level distance` can be specified for individual level numbers, e.g.

```
\gtrset{
  level 0/.style={level distance=5mm},
  level -1/.style={level distance=10mm}
}
...
```

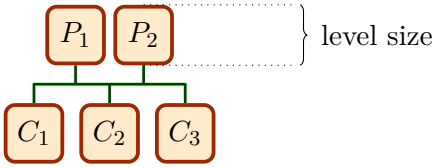


`/gtr/level size`=⟨*length*⟩ (no default, initially `3.5cm`)

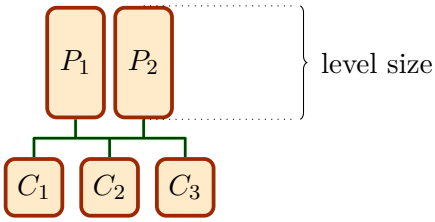
The given ⟨*length*⟩ defines one dimension of a node.

- If `/gtr/timeflow` → P. 78 is `up` or `down`, then `/gtr/level size` sets the height of a node.
- If `/gtr/timeflow` → P. 78 is `left` or `right`, then `/gtr/level size` sets the width of a node.

The `/gtr/level size` be set in dependency of the `/gtr/level` → P. 107.




```
\gtrset{
  level 0/.style={level size=15mm},
}
...
```



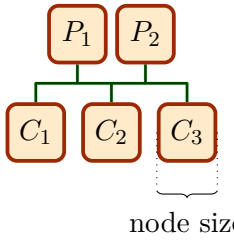
Some actual node implementations may not respect the given `/gtr/level size`. Note that the placement algorithm ignores deviations and assumes that the restrictions hold.

`/gtr/node size`=⟨*length*⟩ (no default, initially `2.5cm`)

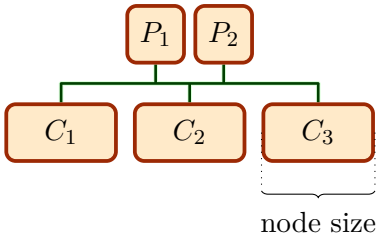
The given ⟨*length*⟩ defines one dimension of a node.

- If `/gtr/timeflow` → P. 78 is `up` or `down`, then `/gtr/level size` → P. 82 sets the width of a node.
- If `/gtr/timeflow` → P. 78 is `left` or `right`, then `/gtr/level size` → P. 82 sets the height of a node.

The `/gtr/node size` can be set in dependency of the `/gtr/level` → P. 107. Note that the `/gtr/node size` may be ignored by nodes boxes which set the width individually or depending from the content width.

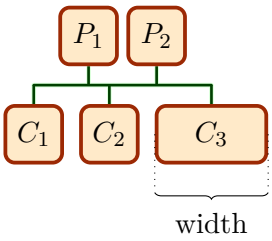



```
\gtrset{
  level 0/.style={level size=15mm},
}
...
```



If the size should be changed for an individual node, use `/gtr/box` → P. 96 instead of `/gtr/node size`:

```
...
    c[id=A,box={width=15mm}]{C_3}
...
```

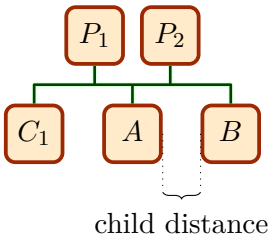


Some actual node implementations may not respect the given `/gtr/node size`. The placement algorithm accepts deviations and calculates positions accordingly.

`/gtr/node size from`=⟨*minlength*⟩ `to` ⟨*maxlength*⟩ (no default, initially `2.5cm to 2.5cm`)
Sets the `/gtr/node size` → P. 83 in a flexible way ranging from ⟨*minlength*⟩ to ⟨*maxlength*⟩. The actual size of a node is determined by the node content. A node will be enlarged up to ⟨*maxlength*⟩ before the content font size is allowed to shrink. Note that the `/gtr/node size from` may be ignored by nodes boxes which set the width individually or depending from the content width.

`/gtr/child distance in parent graph`=⟨*length*⟩ (no default, initially `1mm`)
The given ⟨*length*⟩ defines the minimum distance of two children of a family in a `parent` graph. The `/gtr/child distance in parent graph` can be set in dependency of the `/gtr/level` → P. 107. For an individual node, this distance can be overruled by setting `/gtr/distance` → P. 94.

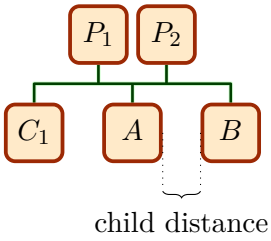
```
...
\genealogytree[
  ...
  child distance in parent graph=5mm]
{
  parent{
    p{P_1}
    p{P_2}
    g{C_1}
    c[id=A]{A}
    c[id=B]{B}
  }
}
...
```



`/gtr/child distance in child graph`=⟨*length*⟩ (no default, initially `2mm`)

The given ⟨*length*⟩ defines the minimum distance of two children of a family in a `child` graph. The `/gtr/child distance in child graph` can be set in dependency of the `/gtr/level` → P. 107. For an individual node, this distance can be overruled by setting `/gtr/distance` → P. 94.

```
...
\genealogytree[
  ...
  child distance in child graph=5mm]
{
  child{
    g{P_1}
    p{P_2}
    c{C_1}
    c[id=A]{A}
    c[id=B]{B}
  }
}
...
```



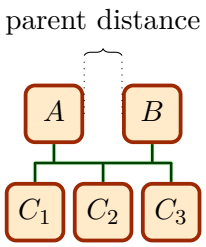
`/gtr/child distance`=⟨*length*⟩ (no default, style)

This is an abbreviation for setting `/gtr/child distance in parent graph` → P. 84 and `/gtr/child distance in child graph` to the same ⟨*length*⟩.

`/gtr/parent distance in parent graph`=⟨*length*⟩ (no default, initially `2mm`)

The given ⟨*length*⟩ defines the minimum distance of two parents of a family in a `parent` graph. The `/gtr/parent distance in parent graph` can be set in dependency of the `/gtr/level` → P. 107. For an individual node, this distance can be overruled by setting `/gtr/distance` → P. 94.

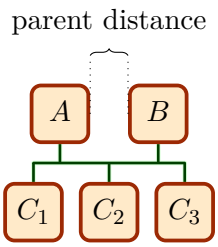
```
...
\genealogytree[
  ...
  parent distance in parent graph=5mm]
{
  parent{
    p[id=A]{A}
    p[id=B]{B}
    g{C_1}
    c{C_2}
    c{C_3}
  }
}
...
```



`/gtr/parent distance in child graph`=⟨*length*⟩ (no default, initially `1mm`)

The given ⟨*length*⟩ defines the minimum distance of two parents of a family in a `child` graph. The `/gtr/parent distance in parent graph` → P. 86 can be set in dependency of the `/gtr/level` → P. 107. For an individual node, this distance can be overruled by setting `/gtr/distance` → P. 94.

```
...
\genealogytree[
  ...
  parent distance in child graph=5mm]
{
  child{
    g[id=A]{A}
    p[id=B]{B}
    c{C_1}
    c{C_2}
    c{C_3}
  }
}
...
```



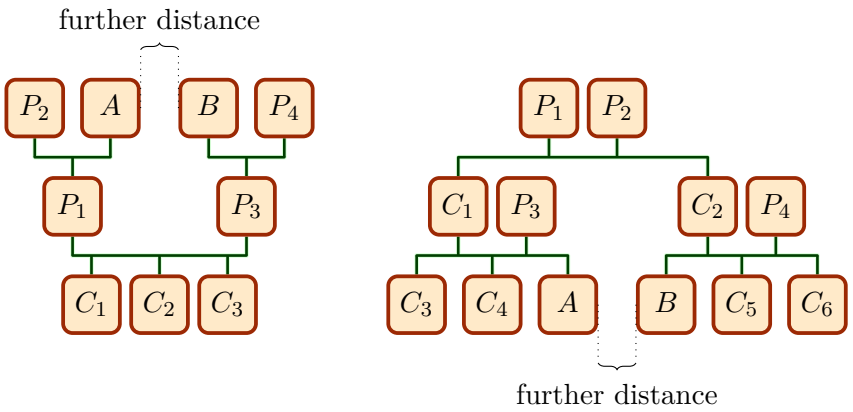
`/gtr/parent distance`=⟨*length*⟩ (no default, style)

This is an abbreviation for setting `/gtr/parent distance in parent graph` → P. 86 and `/gtr/parent distance in child graph` to the same ⟨*length*⟩.

`/gtr/further distance`=⟨*length*⟩ (no default, initially `3mm`)

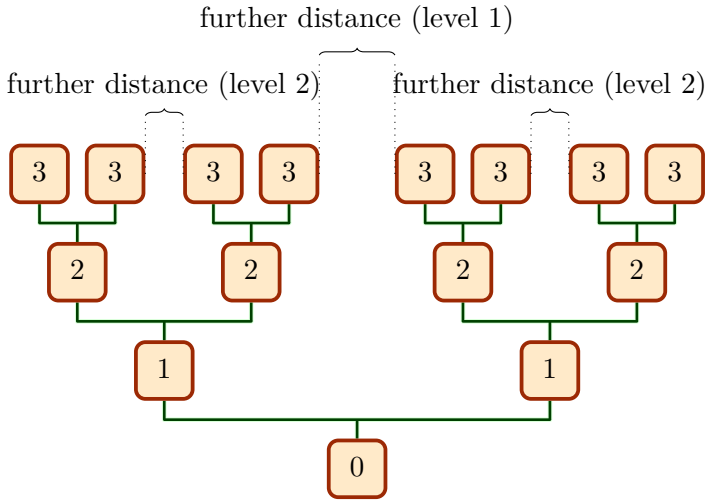
The given ⟨*length*⟩ defines the minimum distance of two nodes which are not parents or children of the same family. The `/gtr/further distance` can be set in dependency of the `/gtr/level` → P. 107. For an individual node, this distance can be overruled by setting `/gtr/distance` → P. 94.

```
...
\gtrset{
  further distance=5mm
}
...
```



If /gtr/further distance → P. 88 is set in level dependency, it is worth to note that this distance is not used for the nodes on the specified layer but for joining the subtrees on the specified layer. In the following example, the distances set on layer 1 and on layer 2 influence different nodes on layer 3.

```
...
\gtrset{
  level 1/.style={further distance=10mm},
  level 2/.style={further distance=5mm},
}
...
```



5.4 Identifiers

Identifiers play an important role for semi-automatic processing of graph data. Every node and every family can be marked by an `/gtr/id` for later reference. If the graph data is exported or generated by a tool, all nodes and families *should* be marked. This allows to manipulate the graph without editing the generated data.

`/gtr/id`=⟨*name*⟩ (no default, initially empty)

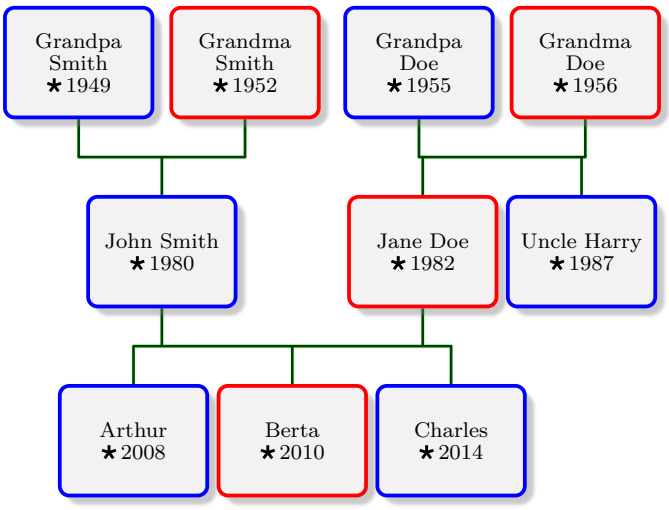
Every node and every family can be marked by a ⟨*name*⟩ using this option. This ⟨*name*⟩ is used by `/gtr/options for node` → P. 93, `/gtr/options for family` → P. 102, etc, to set options for the specified node or family.

- The ⟨*name*⟩ should be unique inside the `tikzpicture` environment.
- A Ti*k*Z node ⟨*name*⟩ is automatically created for later usage.

```
...
child[id=family_a]{     % family with id 'family_a'
  p[id=A]{Father}       % node with id 'A'
  p[id=B]{Mother}       % node with id 'B'
  c[id=C]{Child}        % node with id 'C'
}
...
```

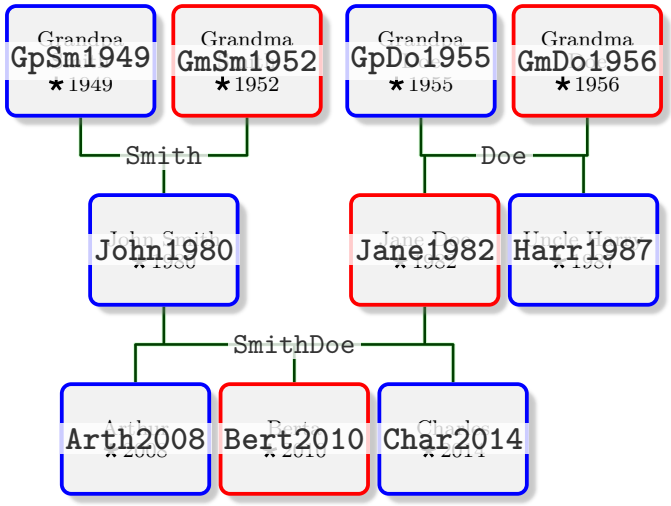
For example, let us consider the Smith-Doe graph used many times in this document, see Section 14.1 on page 287. Using the identifiers, Jane Doe should be emphasized strongly. Without specific manipulation, the graph data is depicted as follows:

```
\begin{tikzpicture}
\genealogytree[template=signpost]
  {input{example.option.graph}}
\end{tikzpicture}
```



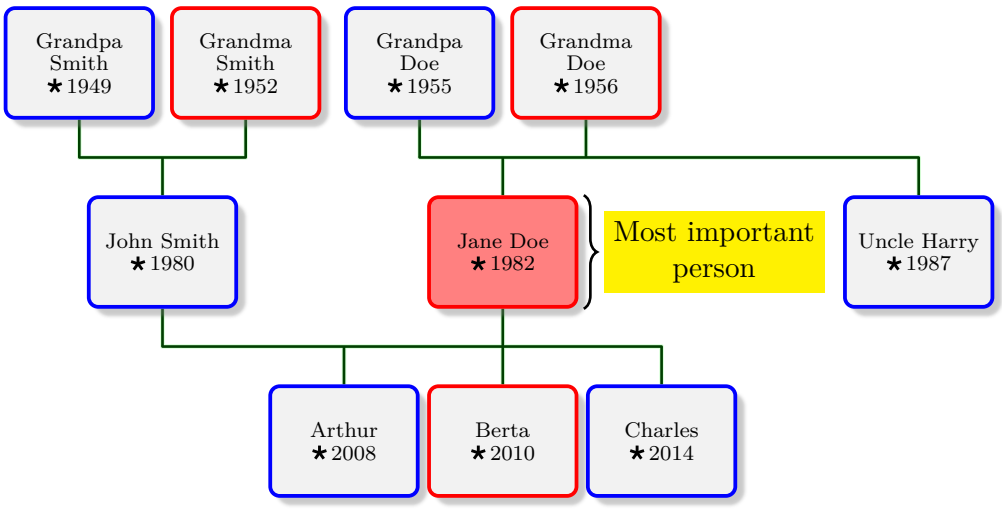
One could inspect the source code in Section 14.1 on page 287 to see the given identifiers. For a large dataset, this may become inconvenient. A good alternative is to use `/gtr/show id` → P. 243 to overlay the depicted graph with all given `/gtr/id` → P. 90 values.

```
\begin{tikzpicture}
\genealogytree[template=signpost,show id]
  {input{example.option.graph}}
\end{tikzpicture}
```



Now, Jane Doe can be emphasized. Note that the id value `Jane1982` is also a TikZ node and can be used such.

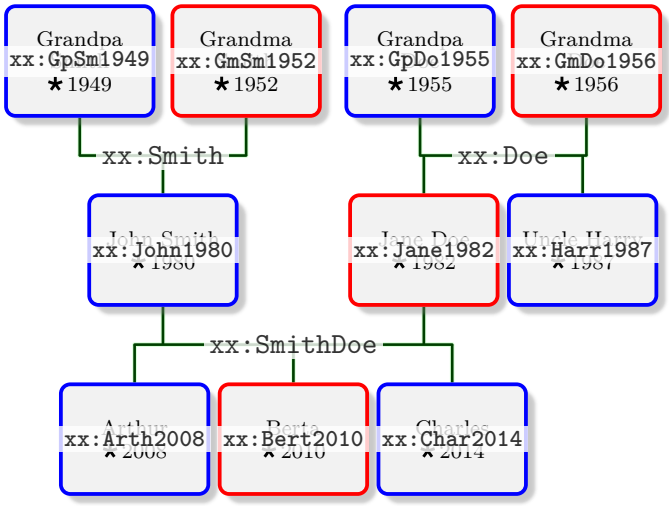
```
\begin{tikzpicture}
\genealogytree[template=signpost,
    options for node={Jane1982}{box={colback=red!50},pivot},
    options for node={Harr1987}{distance=3.5cm}   ]
  {input{example.option.graph}}
\draw [decorate,decoration={brace,amplitude=5pt,mirror,raise=2pt},
  line width=1pt,yshift=0pt] (Jane1982.south east) -- (Jane1982.north east)
  node [align=center,right=9pt,midway,fill=yellow] {Most important\\ person};
\end{tikzpicture}
```



/gtr/id prefix=⟨*text*⟩ (no default, initially empty) N 2015-06-09

The given ⟨*text*⟩ is prefixed to every /gtr/id → P. 90. This option is intended to be used as part of an option list for a \genealogytree → P. 55 or genealogypicture → P. 57. If not used there, it will be set to empty by \genealogytree → P. 55.

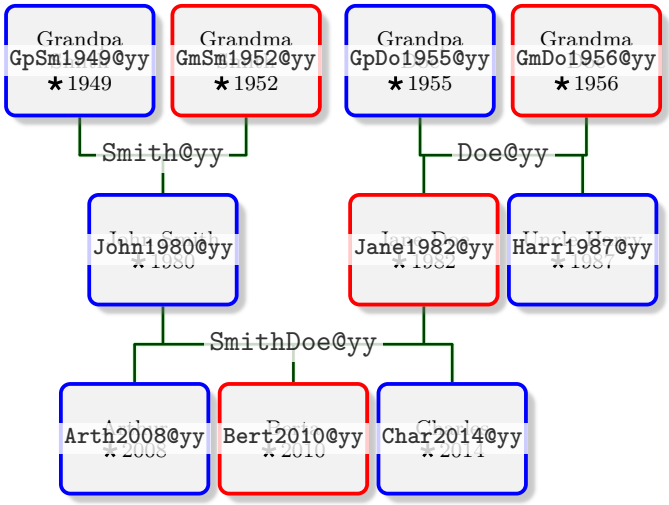
```
\begin{tikzpicture}
\genealogytree[template=signpost,id prefix=xx:,show id]
  {input{example.option.graph}}
\end{tikzpicture}
```



/gtr/id suffix=⟨*text*⟩ (no default, initially empty) N 2015-06-09

The given ⟨*text*⟩ is suffixed to every /gtr/id → P. 90. This option is intended to be used as part of an option list for a \genealogytree → P. 55 or genealogypicture → P. 57. If not used there, it will be set to empty by \genealogytree → P. 55.

```
\begin{tikzpicture}
\genealogytree[template=signpost,id suffix=@yy,show id]
  {input{example.option.graph}}
\end{tikzpicture}
```

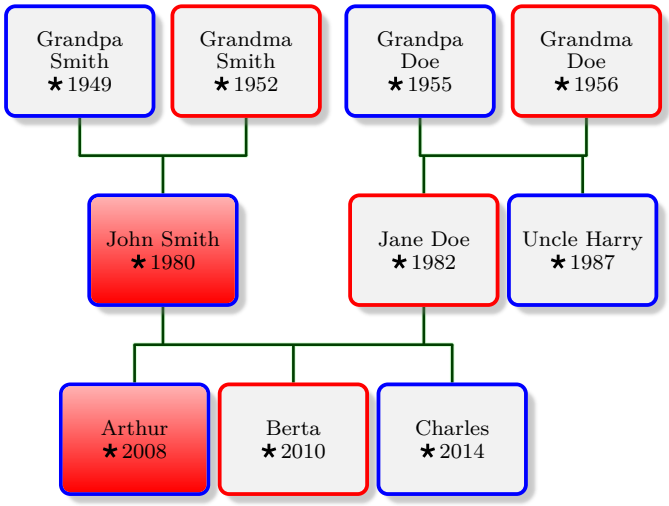


5.5 Node Options

`/gtr/options for node={`⟨*id list*⟩`}{`⟨*options*⟩`}` (style, no default)

The given ⟨*options*⟩ are set for all nodes with `/gtr/id` → P. 90 values from the given ⟨*id list*⟩. If an `/gtr/id` → P. 90 value is not existing, the setting is silently ignored. The intended spot for using `/gtr/options for node` is before `\genealogytree` → P. 55 or inside its option list. Also see Section 5.1.1 on page 76.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for node={Arth2008,John1980}{%
    box={interior style={top color=red!30,bottom color=red}}
  }]
{input{example.option.graph}}
\end{tikzpicture}
```



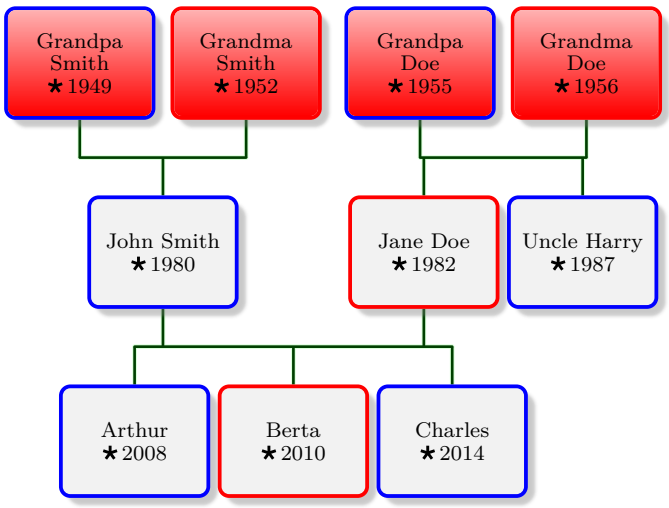
`\gtrsetoptionsfornode{`⟨*id list*⟩`}{`⟨*options*⟩`}`

Identical to using `/gtr/options for node`.

`/gtr/node={`⟨*options*⟩`}` (style, no default)

The given ⟨*options*⟩ are set for all nodes within the current scope. This scope is primarily intended to be a `/gtr/level` → P. 107 or `/gtr/level n` → P. 108 definition. For other spots, where `/gtr/node` is not needed, it may be ignored or directly replaced by its content. Also see Section 5.1.1 on page 76.

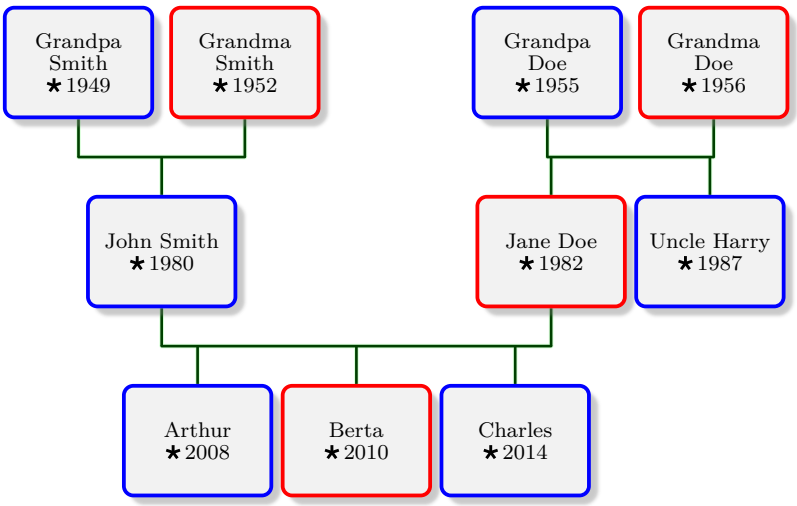
```
\begin{tikzpicture}
\genealogytree[template=signpost,
  level 2/.style={
    node={box={interior style={top color=red!30,bottom color=red}}}
  }]
{input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/distance=`⟨*length*⟩ (no default, initially `-1sp`)

A non-negative ⟨*length*⟩ replaces the default minimum distance to the preceeding sibling. The default settings are given by `/gtr/child distance in parent graph` → P. 84 etc.

```
\begin{tikzpicture}
\genealogytree[template=signpost,options for node={GpDo1955}{distance=2cm}]
  {input{example.option.graph}}
\end{tikzpicture}
```



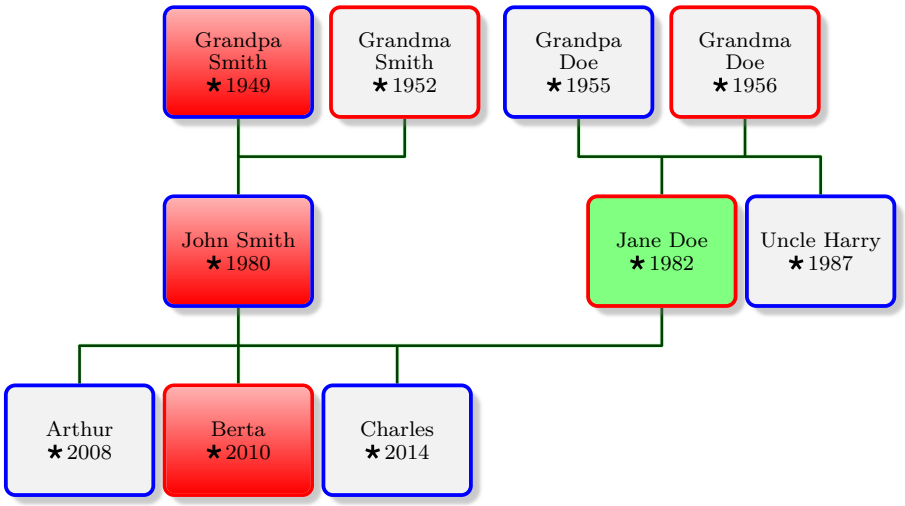
`/gtr/pivot=`⟨*value*⟩ (default `both`, initially `none`)

Using this option, a node can gain a pivot role in the graph construction.
Feasible values are

- `none`: no special treatment.
- `child`: pivot role as a child of a family. The node will be placed centered according to its parents or its pivot parent.
- `parent`: pivot role as a parent of a family. The node will be placed centered according to its children or its pivot child.
- `both`: pivot role as a child and as a parent.

A sequence of `/gtr/pivot` settings for ancestors or descendants can be used to emphasize a certain lineage. In the following example, the nodes marked in red form such a lineage. The green node is a pivot as a child.

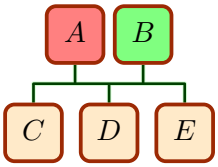
```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for node={Bert2010,John1980,GpSm1949}{pivot,
    box={interior style={top color=red!30,bottom color=red}}},
  options for node={Jane1982}{pivot=child,box={colback=green!50}} ]
{input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/box={⟨options⟩}` (no default)

Passes the given ⟨*options*⟩ to the underlying `/gtr/node processor` → P. 128. Depending on the selected processor, the ⟨*options*⟩ are usually `tcolorbox` options which describe how a node box is drawn. If a processor is not based on the `tcolorbox` package, the ⟨*options*⟩ can be different, e.g. Ti*k*Z options.

```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  options for node={node_B}{box={colback=green!50}},
]{%
  child{
    g[box={colback=red!50}]{A}
    p[id=node_B]{B}
    c{C}   c{D}  c{E}
  }
}
\end{tikzpicture}
```



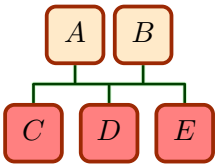
`/gtr/box clear` (no value)

`/gtr/box` settings are additive. To clear all box settings, use this option.

`/gtr/node box=⟨options⟩` (no default)

This is an abbreviation for placing `/gtr/box` inside `/gtr/node` → P. 94.

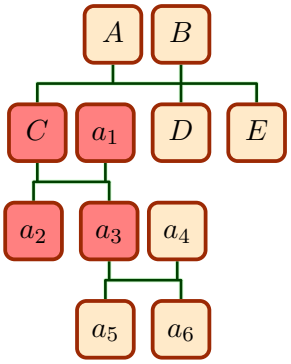
```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  level -1/.style={node box={colback=red!50}},
]{%
  child{
    g{A}   p{B}
    c{C}   c{D}  c{E}
  }
}
\end{tikzpicture}
```



`/gtr/family box=`⟨*options*⟩ (no default)

This is an abbreviation for placing `/gtr/box` → P. 96 inside `/gtr/family` → P. 103.

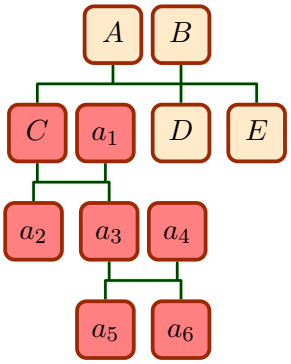
```
\begin{tikzpicture}
\genealogytree[template=formal graph
]{%
  child{
    g{A}   p{B}
    child[family box={colback=red!50}]{
      g{C} p{a_1} c{a_2}
      child{ g{a_3} p{a_4} c{a_5} c{a_6} }
    }
    c{D}   c{E}
  }
}
\end{tikzpicture}
```



`/gtr/subtree box=`⟨*options*⟩ (no default)

This is an abbreviation for placing `/gtr/box` → P. 96 inside `/gtr/subtree` → P. 106.

```
\begin{tikzpicture}
\genealogytree[template=formal graph
]{%
  child{
    g{A}   p{B}
    child[subtree box={colback=red!50}]{
      g{C} p{a_1} c{a_2}
      child{ g{a_3} p{a_4} c{a_5} c{a_6} }
    }
    c{D}   c{E}
  }
}
\end{tikzpicture}
```



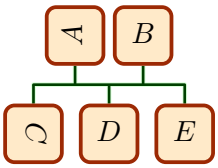
`/gtr/turn=`⟨*option*⟩ (default `right`, initially `off`)

This is a special `/gtr/box` → P. 96 style to rotate the content a node. Typically, all nodes of a `/gtr/level n` → P. 108 may be rotated together.

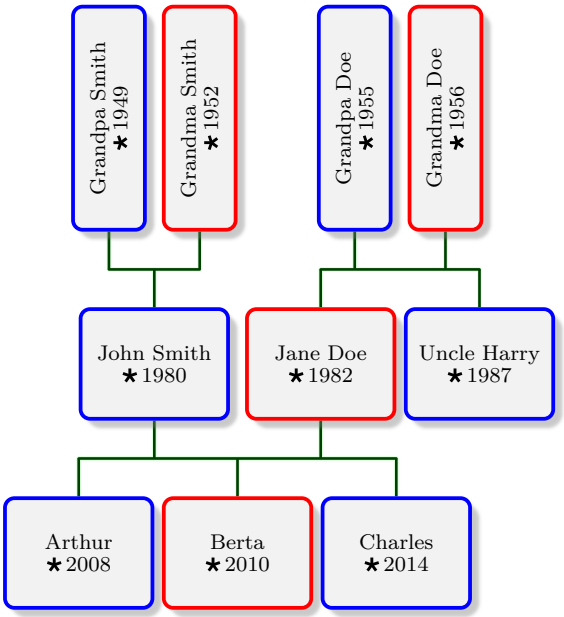
Feasible ⟨*option*⟩ values are:

- `off`: no rotation.
- `right`: rotation by 90 degrees.
- `upsidedown`: rotation by 180 degrees.
- `left`: rotation by 270 degrees.

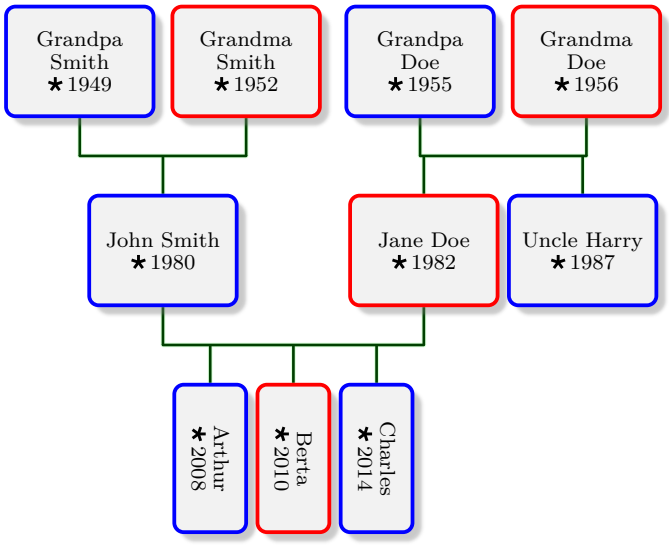
```
\begin{tikzpicture}
\genealogytree[template=formal graph]{%
  child{
    g[turn]{A}
    p{B}
    c[turn=left]{C}   c{D}  c{E}
  }
}
\end{tikzpicture}
```




```
\begin{tikzpicture}
\genealogytree[template=signpost,
  level 2/.style={level size=3cm,node size from=1cm to 2cm,
    node={turn,box={no shadow,drop fuzzy shadow southwest}}}]
  {input{example.option.graph}}
\end{tikzpicture}
```



```
\begin{tikzpicture}
\genealogytree[template=signpost,
  level 0/.style={level size=2cm,node size from=1cm to 2cm,
    node={turn=left,box={no shadow,drop fuzzy shadow northeast}}}]
  {input{example.option.graph}}
\end{tikzpicture}
```



The following three options are `tcolorbox` options which are declared by the `genealogytree` package. They can be redefined for customization.

`/tcb/male` (style, no value)

A `tcolorbox` option defined as

```
\tcbset{male/.style={colframe=blue}}
```

`/tcb/female` (style, no value)

A `tcolorbox` option defined as

```
\tcbset{female/.style={colframe=red}}
```

`/tcb/neuter` (style, no value)

A `tcolorbox` option defined as

```
\tcbset{neuter/.style={}}
```

The following three options are `genealogytree` options which are shortcuts for setting the three options above inside a `/gtr/box` → P. 96.

`/gtr/male` (style, no value, initially unset)

This is an abbreviation for placing `/tcb/male` inside `/gtr/box` → P. 96.

`/gtr/female` (style, no value, initially unset)

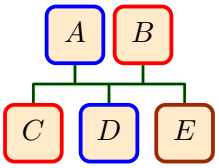
This is an abbreviation for placing `/tcb/female` inside `/gtr/box` → P. 96.

`/gtr/neuter` (style, no value, initially unset)

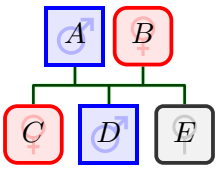
This is an abbreviation for placing `/tcb/neuter` inside `/gtr/box` → P. 96.

Also see `/gtr/database/sex` → P. 155.

```
\begin{tikzpicture}
\genealogytree[template=formal graph,
]{
  child{
    g[male]{A}
    p[female]{B}
    c[female]{C}   c[male]{D}  c[neuter]{E}
  }
}
\end{tikzpicture}
```



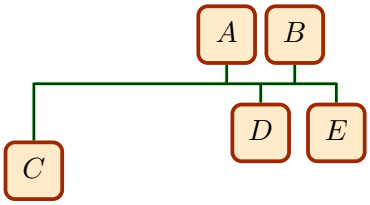

```
\tcbset{
  male/.style={sharp corners,colframe=blue,colback=blue!10,
    watermark text=\gtrSymbolsSetDraw{blue!30}\gtrsymMale},
  female/.style={arc=4pt,colframe=red,colback=red!10,
    watermark text=\gtrSymbolsSetDraw{red!30}\gtrsymFemale},
  neuter/.style={arc=2pt,colframe=black!80!white,colback=black!5,
    watermark text=\gtrSymbolsSetDraw{black!20}\gtrsymNeuter},
}%
\begin{tikzpicture}
\genealogytree[template=formal graph,
]{%
  child{
    g[male]{A}
    p[female]{B}
    c[female]{C}   c[male]{D}  c[neuter]{E}
  }
}
\end{tikzpicture}
```



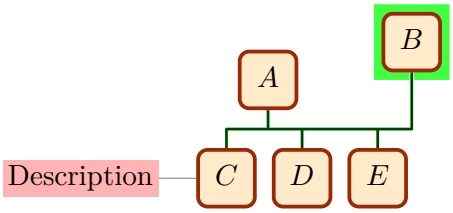
N 2016-07-29 **/gtr/tikz=**⟨*options*⟩ (style, initially empty)

Applys extra TikZ ⟨*options*⟩ for drawing a node. These ⟨*options*⟩ are *not* used for the node box content, even if /gtr/processing → P. 128 is set to **tikznode**. They are used in the drawing process to reserve a node space for the later placement of the box content.
The most interesting usage is to move the node from its computed position. Note that the auto-layout algorithm is *not* aware of such movements, but edge drawing will follow the new positioning.

```
\begin{tikzpicture}
\genealogytree[template=formal graph,
]{%
  child{
    g{A}   p{B}
    c[tikz={xshift=-20mm,yshift=-5mm}]{C}
    c{D}  c{E}
  }
}
\end{tikzpicture}
```




```
\begin{tikzpicture}
\genealogytree[template=formal graph,
]{%
  child{
    g{A}
    p[tikz={xshift=10mm,yshift=5mm,draw=green!75,line width=2mm}]{B}
    c[tikz={
        pin={[fill=red!30,inner sep=2pt]left:Description}
    }]{C}
    c{D}  c{E}
  }
}
\end{tikzpicture}
```



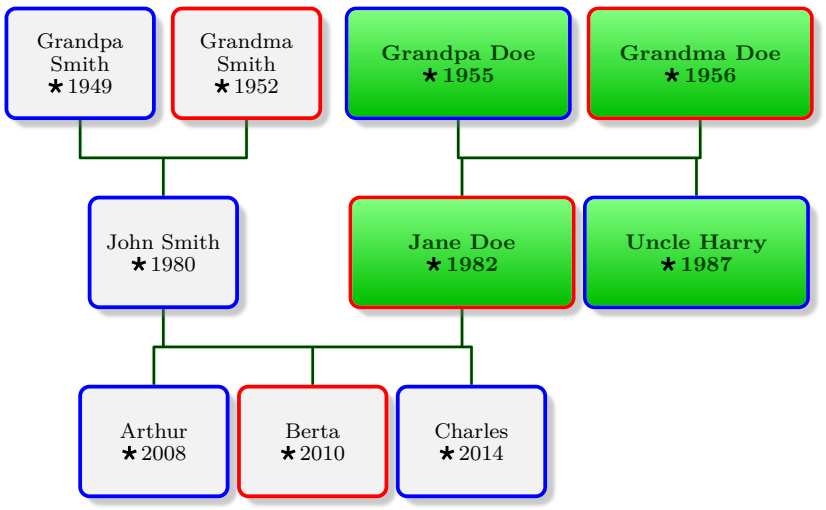
For node movements which influence the auto-layout algorithm, see /gtr/distance → P. 94, /gtr/pivot → P. 95, and /gtr/pivot shift → P. 104. For node content option settings like coloring, see /gtr/box → P. 96.

5.6 Family Options

`/gtr/options for family`={⟨*id list*⟩}{⟨*options*⟩} (style, no default)

The given ⟨*options*⟩ are set for all families with `/gtr/id` → P. 90 values from the given ⟨*id list*⟩. If an `/gtr/id` → P. 90 value is not existing, the setting is silently ignored. The intended spot for using `/gtr/options for family` is before `\genealogytree` → P. 55 or inside its option list. Also see Section 5.1.1 on page 76 and Section 5.1.2 on page 77.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for family={Doe}{box={
    coltext=green!25!black,fontupper=\bfseries,width=3cm,
    interior style={top color=green!50!white,bottom color=green!75!black}
  }}]
{input{example.option.graph}}
\end{tikzpicture}
```



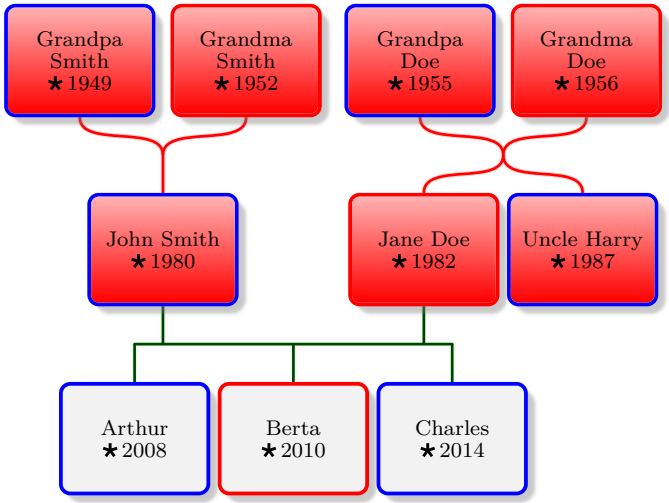
`\gtrsetoptionsforfamily`{⟨*id list*⟩}{⟨*options*⟩}

Identical to using `/gtr/options for family`.

`/gtr/family={⟨options⟩}` (style, no default)

The given ⟨*options*⟩ are set for all nodes and edges within the current scope. This scope is intended to be a `/gtr/level` → P. 107 or `/gtr/level n` → P. 108 definition or an option of a family identifier like `parent` or `child`. Also see Section 5.1.1 on page 76 and Section 5.1.2 on page 77.

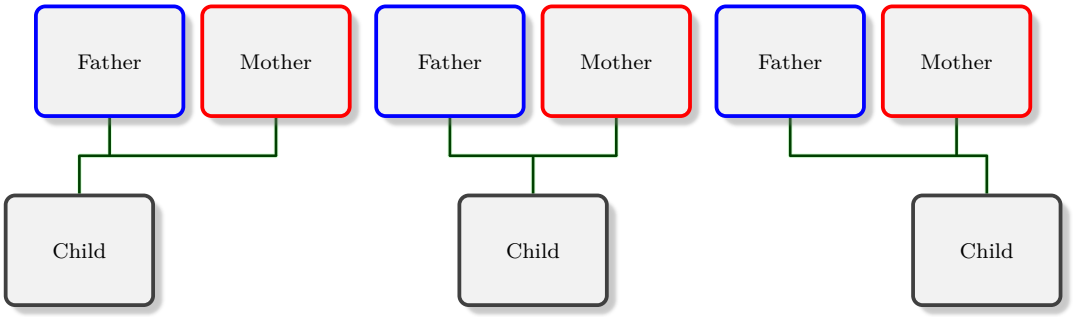
```
\begin{tikzpicture}
\genealogytree[template=signpost,
  level 2/.style={
    family={
      edges={swing,foreground=red,background=red!20},
      box={interior style={top color=red!30,bottom color=red}}}
  }]
{input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/pivot shift=`⟨*length*⟩ (no default, initially `0pt`)

For a family, there is a parent pivot point (typically centered between the parents) and a child pivot point (typically centered between the children). Normally, the auto-layout algorithms brings both points in congruence. Using a `/gtr/pivot shift`, there is a shift of the given ⟨*length*⟩ between these two points. Note that this works for `child`, `parent`, and `sandclock`, but not for `union`.

```
\begin{tikzpicture}
\genealogytree[template=signpost]
{
  parent[pivot shift=-1.5cm]{
    g{Child}
    p[male]{Father}
    p[female]{Mother}
  }
}
\genealogytree[tree offset=4.5cm]{
  parent{
    g{Child}
    p[male]{Father}
    p[female]{Mother}
  }
}
\genealogytree[tree offset=9cm]{
  parent[pivot shift=1.5cm]{
    g{Child}
    p[male]{Father}
    p[female]{Mother}
  }
}
\end{tikzpicture}
```

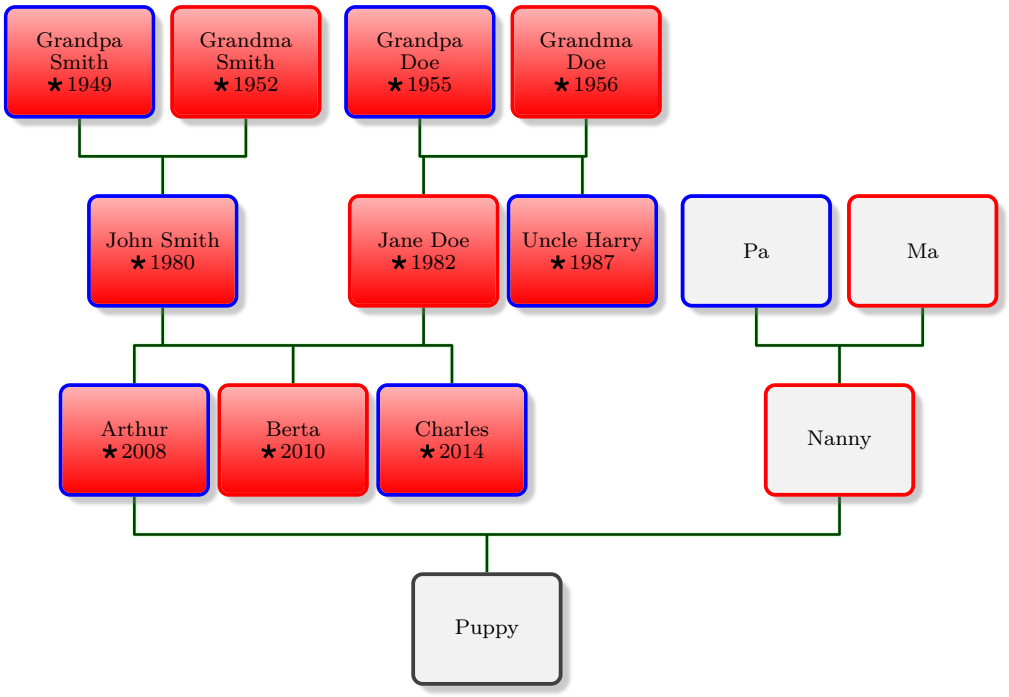


5.7 Subtree Options

`/gtr/options for subtree={⟨id list⟩}{⟨options⟩}` (style, no default)

The given ⟨*options*⟩ are set for all subtrees with `/gtr/id` → P. 90 values from the given ⟨*id list*⟩. Subtrees are identified by the `/gtr/id` → P. 90 of the root family of the subtree. If an `/gtr/id` → P. 90 value is not existing, the setting is silently ignored. The intended spot for using `/gtr/options for subtree` is before `\genealogytree` → P. 55 or inside its option list. Also see Section 5.1.1 on page 76 and Section 5.1.2 on page 77.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for subtree={SmithDoe}{%
    box={interior style={top color=red!30,bottom color=red}}}
  ]
{
  parent{
    g{Puppy}
    input{example.option.graph}
    parent
    {
      g[female]{Nanny}
      p[male]{Pa}
      p[female]{Ma}
    }
  }
}
\end{tikzpicture}
```



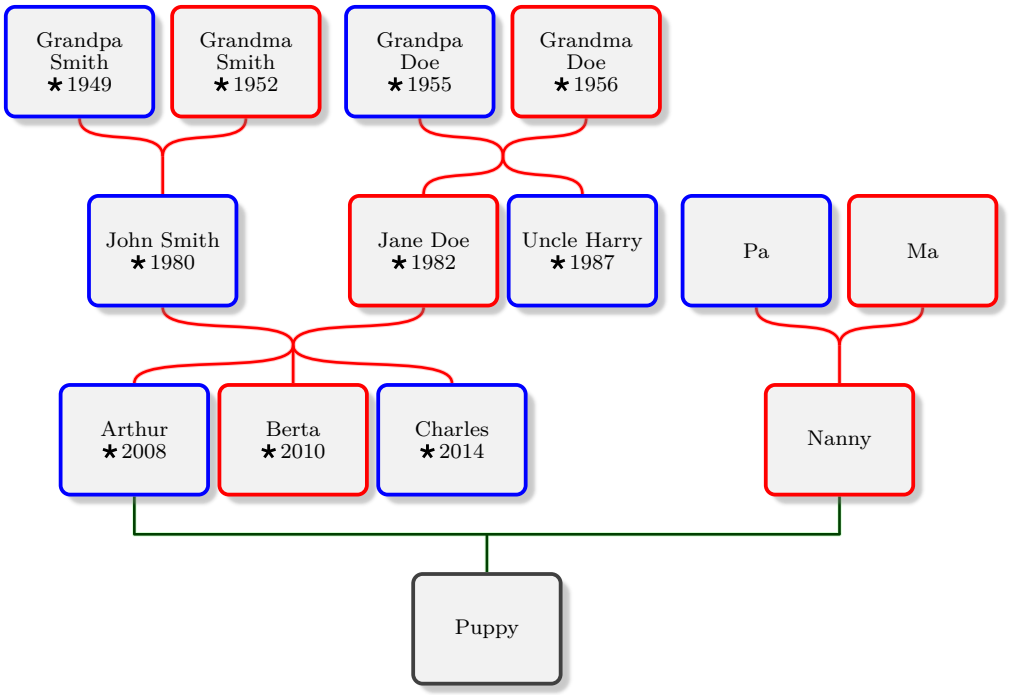
`\gtrsetoptionsforsubtree{⟨id list⟩}{⟨options⟩}`

Identical to using `/gtr/options for subtree`.

`/gtr/subtree={⟨options⟩}` (style, no default)

The given ⟨*options*⟩ are set for all families and their nodes and edges within the current scope. This scope is intended to be a `/gtr/level` → P. 107 or `/gtr/level n` → P. 108 definition or an option of a family identifier like `parent` or `child`. Also see Section 5.1.1 on page 76 and Section 5.1.2 on page 77.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  level 2/.style={%
    subtree={edges={swing,foreground=red,background=red!20}}}
  ]
{
  parent{
    g{Puppy}
    input{example.option.graph}
    parent
    {
      g[female]{Nanny}
      p[male]{Pa}
      p[female]{Ma}
    }
  }
}
\end{tikzpicture}
```



5.8 Level Options

With `/gtr/level` and `/gtr/level n` → P. 108 options can be set for individual levels of the graph. Inside the key list of these styles, the following options can be used:

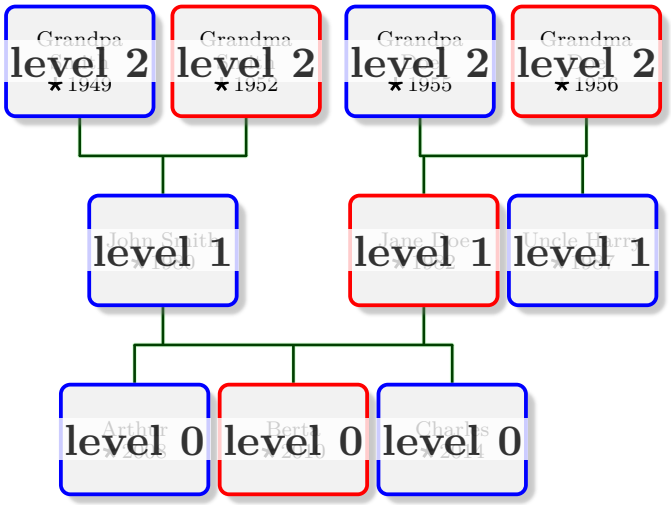
- All geometry options, see Section 5.3 on page 81.
- `/gtr/node` → P. 94 to set options for nodes.
- `/gtr/family` → P. 103 to set options for families.
- `/gtr/subtree` → P. 106 to set options for subtrees.
- Also see `/gtr/ignore` → P. 115 and `/gtr/ignore level` → P. 117.

Also see Section 5.1.1 on page 76 and Section 5.1.2 on page 77.

`/gtr/level`=⟨*number*⟩ (style, initially empty)

An initially empty style which as applied at each level with the level ⟨*number*⟩ as parameter. This style can be redefined.

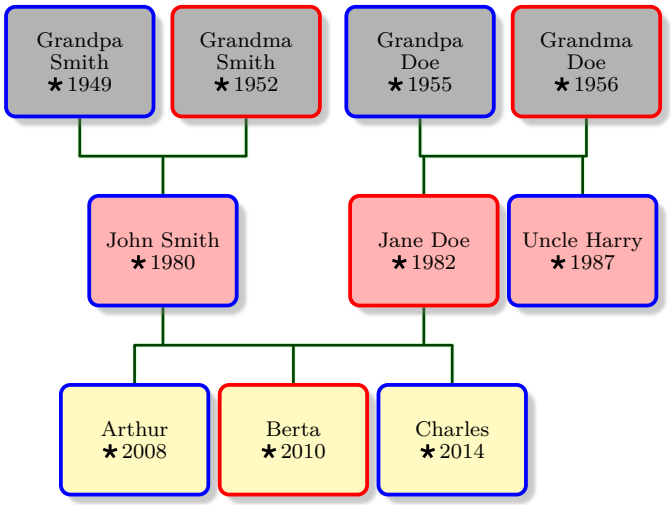
```
\begin{tikzpicture}
\genealogytree[template=signpost,
    level/.style={node={show=level #1}} ]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/level n` (style, initially unset)

At each level with the level number n this style is applied after `/gtr/level` → P. 107. This style can be (re-)defined.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  level 2/.style={node box={colback=black!30}},
  level 1/.style={node box={colback=red!30}},
  level 0/.style={node box={colback=yellow!30}},
]
{input{example.option.graph}}
\end{tikzpicture}
```



5.9 Tree Positioning Options

`/gtr/proband level`=⟨*number*⟩ (no default, initially `0`)

Sets the level number of the proband to ⟨*number*⟩. All level numbers inside the given tree will be adapted accordingly. This is useful in connection with `/gtr/level` → P. 107 dependent settings, especially when two trees are connected.

`/gtr/tree offset`=⟨*length*⟩ (no default, initially `0pt`)

Sets the offset value of the root family to ⟨*length*⟩. Depending on the given `/gtr/timeflow` → P. 78, this means a shift in horizontal or vertical direction in reference of the `tikzpicture` coordinate system.

`/gtr/after parser`=⟨*code*⟩ (no default, initially empty)

Adds ⟨*code*⟩ to a list of code which is executed after the tree content is parsed and before the parsed data is drawn. This is used internally by other options and may not be needed by a normal user.

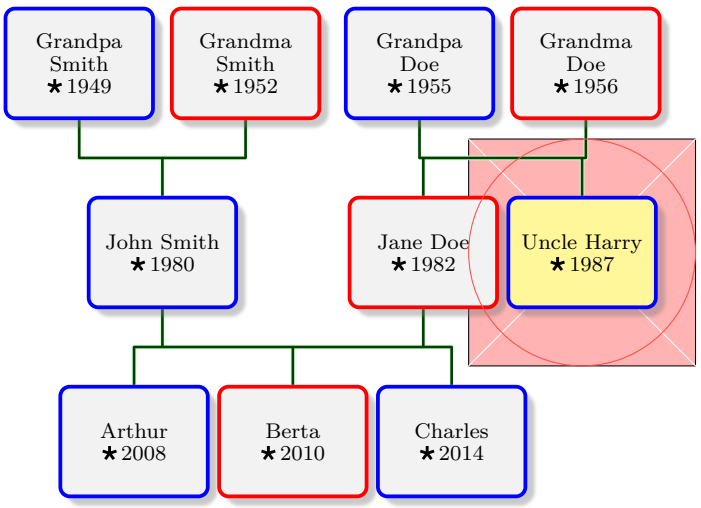
The following options allow to shift the whole tree such that a specific node is placed at a specific position.

- `/gtr/set position`: place a node centered at a position.
- `/gtr/adjust position` → P. 110: place a node relative to a position (respecting the node dimensions).
- `/gtr/adjust node` → P. 111: place a node relative to another node (respecting both node dimensions).

`/gtr/set position`=⟨*node*⟩ `at` ⟨*position*⟩ (style, no default)

Adjusts the current graph such that a ⟨*node*⟩ of the graph is placed at the given ⟨*position*⟩. If the ⟨*position*⟩ is given by coordinates, one has to use curly brackets to enclose ⟨*position*⟩, e.g. `{2,3}`. The ⟨*node*⟩ is identified by a `/gtr/id` → P. 90.

```
\begin{tikzpicture}
\node[draw,fill=red!30,minimum size=3cm] (X) at (0,0) {};
\draw[white] (X.south west)--(X.north east) (X.north west)--(X.south east);

\genealogytree[template=signpost,
  set position=Harr1987 at X,
  options for node={Harr1987}{box={colback=yellow!50}}]
  {input{example.option.graph}}

\draw[red!70] (Harr1987) circle (1.5cm);
\end{tikzpicture}
```

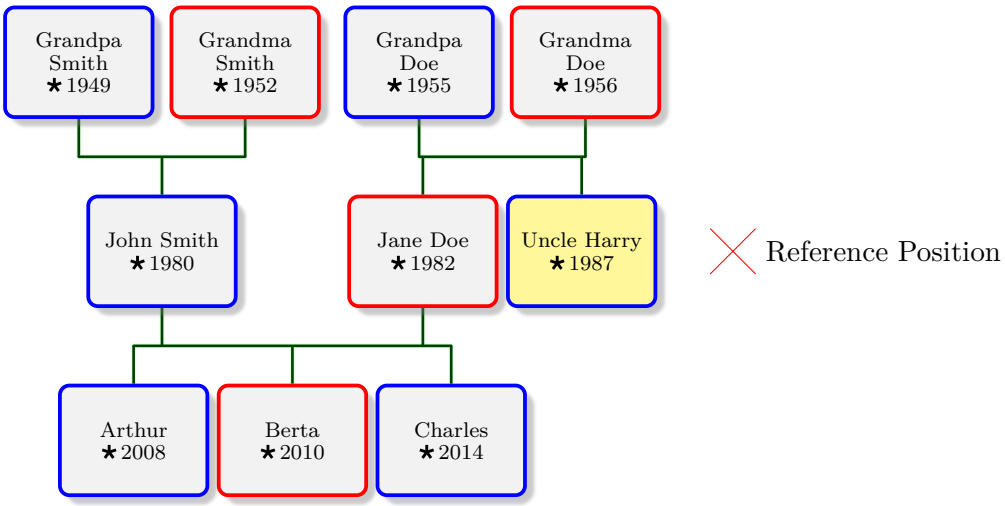


`/gtr/adjust position`=⟨*node*⟩ ⟨*direction*⟩ `of` ⟨*position*⟩ `distance` ⟨*distance*⟩ `shift` ⟨*shift*⟩ (style, no default)

Adjusts the current graph such that a ⟨*node*⟩ of the graph is placed in the given ⟨*direction*⟩ relative to the given ⟨*position*⟩ with a given ⟨*distance*⟩ in this direction and an optional ⟨*shift*⟩ orthogonal to the direction. The ⟨*node*⟩ is identified by a `/gtr/id` → P. 90.
Feasible values for the ⟨*direction*⟩ are

- `right`
- `left`
- `above`
- `below`

```
\begin{tikzpicture}
\draw[red] (-0.3,-0.3)--++(0.6,0.6) (-0.3,0.3)--++(0.6,-0.6);
\node[right=3mm] at (0,0) {Reference Position};

\genealogytree[template=signpost,
  adjust position=Harr1987 left of {0,0} distance 1cm,
  options for node={Harr1987}{box={colback=yellow!50}}]
  {input{example.option.graph}}
\end{tikzpicture}
```



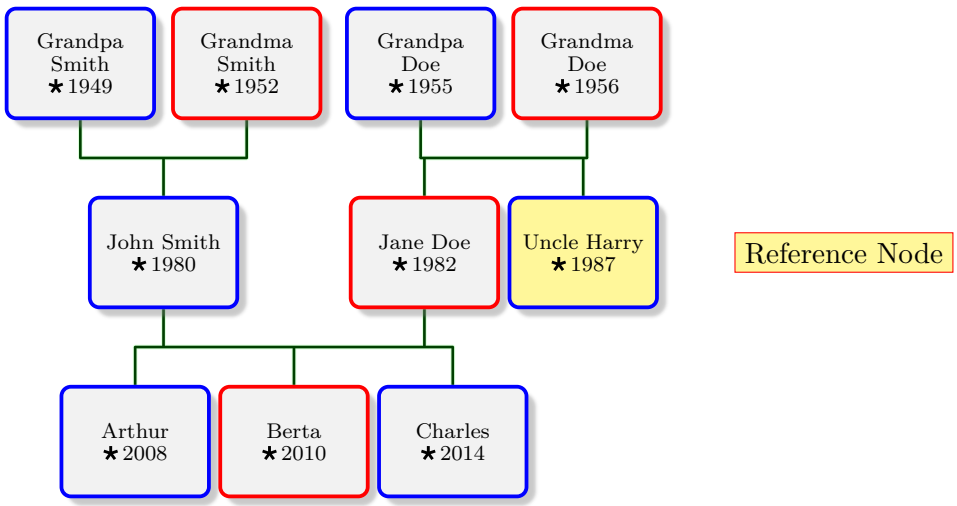
`/gtr/adjust node=`⟨*node*⟩ ⟨*direction*⟩ `of` ⟨*reference node*⟩ (style, no default)
`distance` ⟨*distance*⟩ `shift` ⟨*shift*⟩

Adjusts the current graph such that a ⟨*node*⟩ of the graph is placed in the given ⟨*direction*⟩ relative to the given ⟨*reference node*⟩ (a TikZ node) with a given ⟨*distance*⟩ in this direction and an optional ⟨*shift*⟩ orthogonal to the direction. The ⟨*node*⟩ is identified by a `/gtr/id` → P. 90.

Feasible values for the ⟨*direction*⟩ are

- `right` (right of ⟨*reference node*⟩`.east`)
- `left` (left of ⟨*reference node*⟩`.west`)
- `above` (above of ⟨*reference node*⟩`.north`)
- `below` (below of ⟨*reference node*⟩`.south`)

```
\begin{tikzpicture}
\node[fill=yellow!50,draw=red] (R) {Reference Node};

\genealogytree[template=signpost,
  adjust node=Harr1987 left of R distance 1cm,
  options for node={Harr1987}{box={colback=yellow!50}}]
  {input{example.option.graph}}
\end{tikzpicture}
```



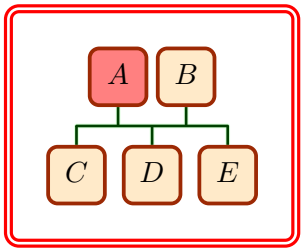
5.10 TikZ and Tcolorbox Options

Also see /gtr/tikz → P. 101.

/gtr/tikzpicture={⟨*tikz options*⟩} (no default, initially empty)

Used to insert ⟨*tikz options*⟩ to the `tikzpicture` environment inside `genealogypicture` → P. 57. **This option is ignored by `\genealogytree` → P. 55!**

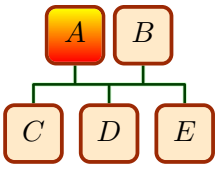
```
\begin{genealogypicture}[template=formal graph,
  tikzpicture={execute at end picture={
    \path[draw=red,double,double distance=1pt,very thick,rounded corners]
      ([xshift=-5mm,yshift=-5mm]current bounding box.south west) rectangle
      ([xshift=5mm,yshift=5mm]current bounding box.north east);}}  ]
  child{
    g[box={colback=red!50}]{A}
    p{B}
    c{C}   c{D}  c{E}
  }
\end{genealogypicture}
```



/gtr/tikzset={⟨*tikz options*⟩} (no default, initially empty)

Used to insert ⟨*tikz options*⟩ before the tree is drawn by `\genealogytree` → P. 55 or `genealogypicture` → P. 57. In contrast to /gtr/tikzpicture, one can use /gtr/tikzset also for `\genealogytree` → P. 55, but some some settings may need to be given in the argument of `tikzpicture` (see *The TikZ and PGF Packages* [4]).
Note that `\genealogytree` → P. 55 does not limit the scope of these settings.

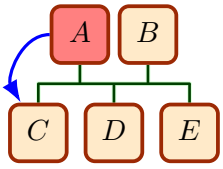
```
\begin{genealogypicture}[template=formal graph,
  tikzset={myfill/.style={top color=yellow,bottom color=red}}  ]
  child{
    g[box={interior style=myfill}]{A}
    p{B}
    c{C}   c{D}  c{E}
  }
\end{genealogypicture}
```



/gtr/after tree={⟨*tikz code*⟩} (no default, initially empty)

Used to insert ⟨*tikz code*⟩ after the tree is drawn by \genealogytree → P. 55 or genealogypicture → P. 57. This is also used internally by other options.

```
\begin{genealogypicture}[template=formal graph,
  after tree={ \draw[very thick,blue,-Latex] (node_A) to[out=180,in=120] (node_C);
    }]
  child{
    g[box={colback=red!50},id=node_A]{A}
    p{B}
    c[id=node_C]{C}   c{D}  c{E}
  }
\end{genealogypicture}
```

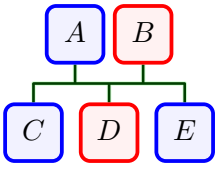


/gtr/tcbset={⟨*tcolorbox options*⟩} (no default, initially empty)

Used to insert ⟨*tcolorbox options*⟩ before the tree is drawn by \genealogytree → P. 55 or genealogypicture → P. 57.

Note that \genealogytree → P. 55 does not limit the scope of these settings.

```
\begin{genealogypicture}[template=formal graph,
  tcbset={
    male/.style={colframe=blue,colback=blue!5},
    female/.style={colframe=red,colback=red!5}
  }
]
  child{
    g[male]{A}
    p[female]{B}
    c[male]{C}   c[female]{D}  c[male]{E}
  }
\end{genealogypicture}
```



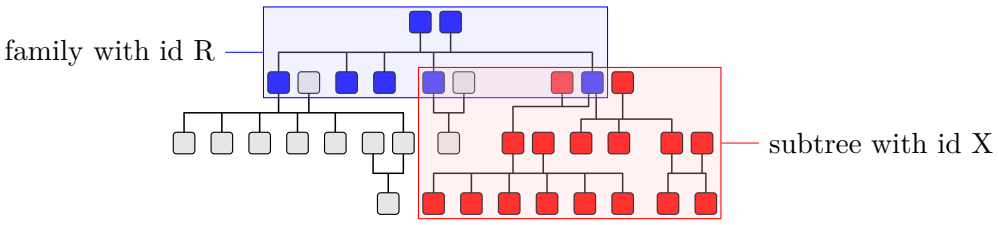
`/tikz/fit to family`=⟨*id*⟩ (style, no default)

This is an extension to the `fit` library of Ti*k*Z. This option must be given to a `node` path command. The ⟨*id*⟩ has to be an `/gtr/id` → P. 90 value of a family. All nodes of this family are given to the `fit` option of a Ti*k*Z node which is sized to frame all family members.

`/tikz/fit to subtree`=⟨*id*⟩ (style, no default)

Like `/tikz/fit to family`, this is an extension to the `fit` library of Ti*k*Z. All nodes of the subtree identified by ⟨*id*⟩ are given to the `fit` option of a Ti*k*Z node which is sized to frame the whole subtree.

```
\begin{tikzpicture}
\genealogytree[template=tiny boxes]
{
  child[id=R,family box={colback=blue}]{
    g-p-
    child{
      g-p-c-c-c-c-c-
      child{ p-g-c- }
    }
    c-c-
    child{ g-p-c- }
    child[id=X,subtree box={colback=red}]{
      p-g-
      child{ g-p-c-c-c-c-c-c- }
      union{
        p-c-c-
        child{ g-p-c-c- }
      }
    }
  }
}

\node[draw=blue,fill=blue!20,fill opacity=0.25,inner sep=0.5mm,
  pin={[pin edge=blue]left:family with id R},
  fit to family=R] {};
\node[draw=red,fill=red!20,fill opacity=0.25,inner sep=0.5mm,
  pin={[pin edge=red]right:subtree with id X},
  fit to subtree=X] {};
\end{tikzpicture}
```



5.11 Ignoring Input

The following options allow to ignore some parts of the input data. Note that debugging using the methods from Chapter 11 on page 227 will usually ignore the ignore settings. Also, if some counters are incremented by node or family options, these increments may not be undone by ignoring the particular node or family.

`/gtr/ignore=true|false` (default `true`, initially `false`)

The `/gtr/ignore` option can be used inside the option list for any node or family specifier. Child `c` and parent `p` leaf nodes are simply ignored, if this option is used. An error will arise, if a `g` node is ignored and there is no other `g` node for the family.

- Using `/gtr/ignore` inside a node ignores this node.
- Using `/gtr/ignore` inside a family means that the whole subtree becomes ignored.
- Using `/gtr/ignore` inside `/gtr/level n` → P. 108 means that all families on this level are ignored. Since families span two levels, the effect may not be restricted to the target level. Leaf nodes on the target level are not affected. Also see `/gtr/ignore level` → P. 117.

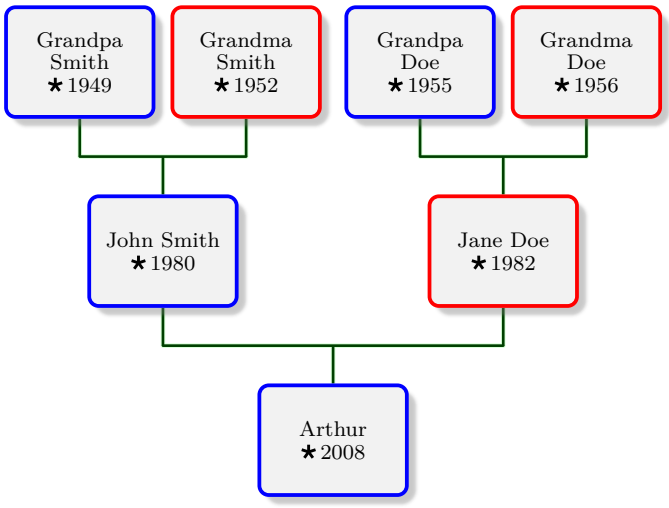
To ignore a node or subtree by its `/gtr/id` → P. 90, use `/gtr/ignore node` → P. 116 or `/gtr/ignore subtree` → P. 116.

```
\begin{tikzpicture}
\genealogytree[template=signpost,timeflow=left,level size=3cm]
{
  parent[id=DoeJones]{
    g[id=Deir2012,female]{Deirdre\\\gtrsymBorn\,2012}
    parent[id=Jones]{
      g[id=Mary1988,female]{Aunt Mary\\\gtrsymBorn\,1988}
      p[id=JimJ1944,male]{Jim Jones\\\gtrsymBorn\,1944}
      % the following node is going to be ignored
      p[ignore,id=Jenn1949,female]{Jenny Jones\\\gtrsymBorn\,1949}
    }
  }
}
\end{tikzpicture}
```

`/gtr/ignore node={⟨id list⟩}` (style, no default)

All nodes with `/gtr/id` → P. 90 values from the given ⟨*id list*⟩ are ignored. If an `/gtr/id` → P. 90 value is not existing, the setting is silently ignored.

```
\begin{tikzpicture}
\genealogytree[template=signpost,ignore node={Bert2010,Char2014,Harr1987}]
  {input{example.option.graph}}
\end{tikzpicture}
```



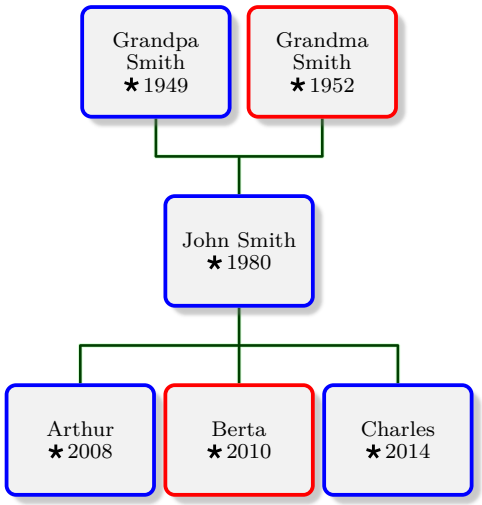
`\gtrignorenode{⟨id list⟩}`

Identical to using `/gtr/ignore node`.

`/gtr/ignore subtree={⟨id list⟩}` (style, no default)

All subtrees with `/gtr/id` → P. 90 values from the given ⟨*id list*⟩ are ignored. If an `/gtr/id` → P. 90 value is not existing, the setting is silently ignored.

```
\begin{tikzpicture}
\genealogytree[template=signpost,ignore subtree={Doe}]
  {input{example.option.graph}}
\end{tikzpicture}
```



`\gtrignoresubtree{⟨id list⟩}`

Identical to using `/gtr/ignore subtree`.

`/gtr/ignore level=`⟨*number*⟩ (style, no default)

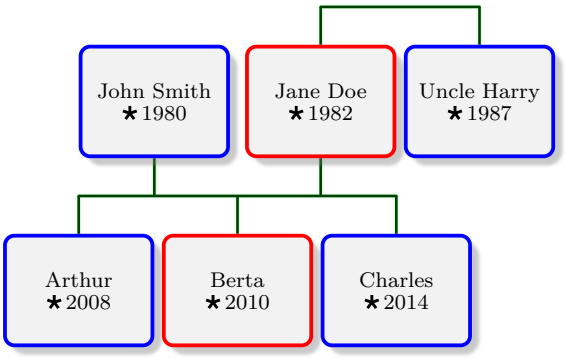
The level with the given ⟨*number*⟩ is ignored. This also removes unconnected nodes and families. **Note that `/gtr/ignore level` should never be used, if `/gtr/proband level`** → P. 109 **was set!**

This style sets `/gtr/level n` → P. 108 options to remove all unwanted nodes and families. Depending on the algebraic sign of ⟨*number*⟩ the implementation differs. Zero has no effect.

```
\gtrset{ignore level=4}
    % is equal to
\gtrset{level 4/.style={node=ignore},level 5/.style={ignore}}
```

```
\gtrset{ignore level=-4}
    % is equal to
\gtrset{level -4/.style={ignore,node=ignore}}
```

```
\begin{tikzpicture}
\genealogytree[template=signpost,ignore level=2]
  {input{example.option.graph}}
\end{tikzpicture}
```



5.12 Inserting Input

The following options allow to insert parsable data into the input. This is a powerful feature with the risk to corrupt the structure of the resulting graph. Note that grammar checks are not so strictly applied at the insertion points and occurring errors may be difficult to detect.

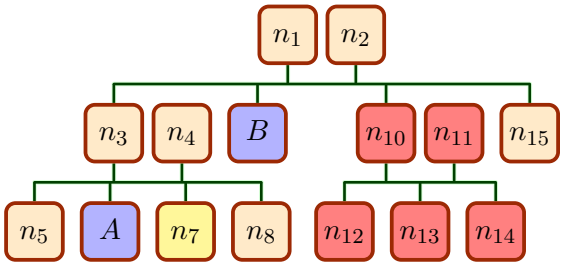
`\gtrparserdebug` → P. 228 ignores inserting options, while `\gtrprocessordebug` → P. 230 uses these options.

Recursive insertion is possible, i.e. inserting into already inserted data, but should be handled with care. Especially, using `/gtr/insert after node` and `/gtr/insert after family` → P. 119 should never be used to insert data after the root element of an inserted node or family.

`/gtr/insert after node`={⟨*id*⟩}{⟨*input data*⟩} (style, no default)

Inserts ⟨*input data*⟩ into the graph data right after the node with the given ⟨*id*⟩ was processed. If no node with ⟨*id*⟩ exists, this setting is silently ignored. If more than one insert command was given for a specific node, following insert commands for this node are ignored. Note that grammar checks are not so strictly applied at the insertion point, i.e. one has to be more careful to obey the rules to avoid mess.

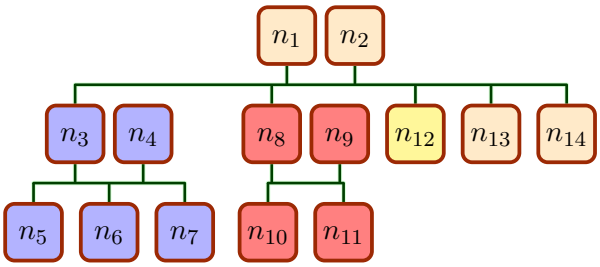
```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  content interpreter content={\gtrifnodeid{\gtrnodeid}{n_{\gtrnodenumber}}},
  insert after node={A}{  c[box={colback=yellow!50}]-  },
  insert after node={B}{  child[subtree box={colback=red!50}]{g-p-c-c-c-}  },
]
{
  child{ g-p-
    child{ g-p-c-c[box={colback=blue!30},id=A]-c- }
    c[box={colback=blue!30},id=B]-c-
  }
}
\end{tikzpicture}
```



`/gtr/insert after family={⟨id⟩}{⟨input data⟩}` (style, no default)

Inserts ⟨*input data*⟩ into the graph data right after the family with the given ⟨*id*⟩ was processed. If no family with ⟨*id*⟩ exists, this setting is silently ignored. There should be not more than one `/gtr/insert after family` command for a specific family; using it twice may give unpredictable results. Note that grammar checks are not so strictly applied at the insertion point, i.e. one has to be more careful to obey the rules to avoid mess. Especially, never use `/gtr/insert after family` for the root family!

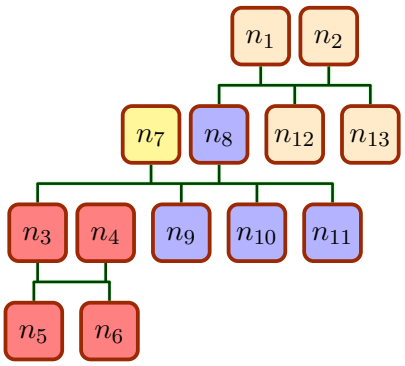
```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  content interpreter content={\gtrifnodeid{\gtrnodeid}{n_{\gtrnodenumber}}},
  insert after family={fam_a}{
    child[subtree box={colback=red!50}]{g-p-c-c-}
    c[box={colback=yellow!50}]-
  },
]
{
  child[id=root]{ g-p-
    child[subtree box={colback=blue!30},id=fam_a]{ g-p-c-c-c- }
    c-c-
  }
}
\end{tikzpicture}
```



`/gtr/insert at begin family={⟨id⟩}{⟨input data⟩}` (style, no default)

Inserts ⟨*input data*⟩ into the graph data of the family with the given ⟨*id*⟩, before the content of the family is processed. If no family with ⟨*id*⟩ exists, this setting is silently ignored. There should be not more than one `/gtr/insert at begin family` command for a specific family; using it twice may give unpredictable results. Note that grammar checks are not so strictly applied at the insertion point, i.e. one has to be more careful to obey the rules to avoid mess.

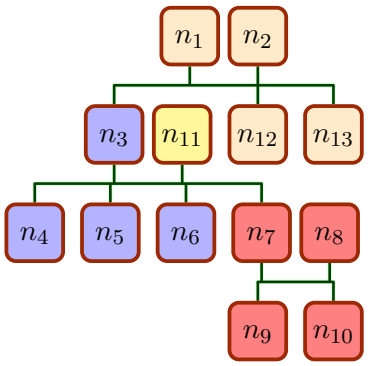
```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  content interpreter content={\gtrifnodeid{\gtrnodeid}{n_{\gtrnodenumber}}},
  insert at begin family={fam_a}{
    child[subtree box={colback=red!50}]{g-p-c-c-}
    p[box={colback=yellow!50}]-
  },
]
{
  child[id=root]{ g-p-
    child[subtree box={colback=blue!30},id=fam_a]{ g-c-c-c- }
    c-c-
  }
}
\end{tikzpicture}
```



`/gtr/insert at end family={⟨id⟩}{⟨input data⟩}` (style, no default)

Inserts ⟨*input data*⟩ into the graph data of the family with the given ⟨*id*⟩, after the content of the family is processed. If no family with ⟨*id*⟩ exists, this setting is silently ignored. There should be not more than one `/gtr/insert at end family` command for a specific family; using it twice may give unpredictable results. Note that grammar checks are not so strictly applied at the insertion point, i.e. one has to be more careful to obey the rules to avoid mess.

```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  content interpreter content={\gtrifnodeid{\gtrnodeid}{n_{\gtrnodenumber}}},
  insert at end family={fam_a}{
    child[subtree box={colback=red!50}]{g-p-c-c-}
    p[box={colback=yellow!50}]-
  },
]
{
  child[id=root]{ g-p-
    child[subtree box={colback=blue!30},id=fam_a]{ g-c-c-c- }
    c-c-
  }
}
\end{tikzpicture}
```

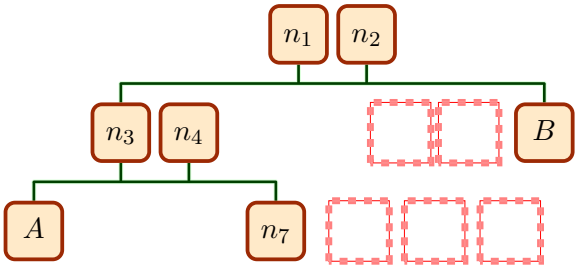


5.13 Phantom Nodes and Subtrees

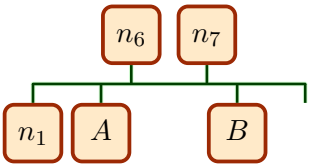
`/gtr/phantom=`⟨*length*⟩ (style, default `/gtr/node size` → P. 83) U 2015-06-08

A `/gtr/subtree` → P. 106 style which makes the whole current subtree invisible. This style can also be applied for single nodes. If a ⟨*length*⟩ value is used, the `/gtr/node size` → P. 83 for all nodes of the subtree is replaced by ⟨*length*⟩ (width for vertical time flow and height for horizontal time flow).

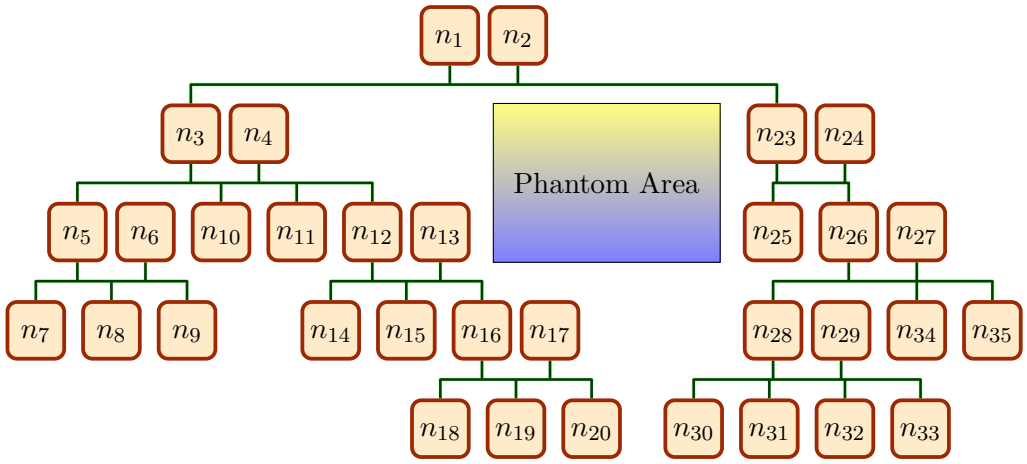
```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  content interpreter content={\gtrifnodeid{\gtrnodeid}{n_{\gtrnodenumber}}},
]
{
  child{ g-p-
    child{ g-p-
      c[id=A]-
      % invisible phantom
      c[phantom=2cm]-
      c-
    }
    % phantom; borders made visible
    child[phantom,subtree box={show bounding box}]{
      g-p-c-c-c-
    }
    c[id=B]-
  }
}
\end{tikzpicture}
```




```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  content interpreter content={\gtrifnodeid{\gtrnodeid}{n_{\gtrnodenumber}}},
  insert after node={A}{ c[phantom]- },
  insert after node={B}{ c[phantom*]- },
]
{
  parent{
    g-c[id=A]-c[id=B]-
    p-p-
  }
}
\end{tikzpicture}
```



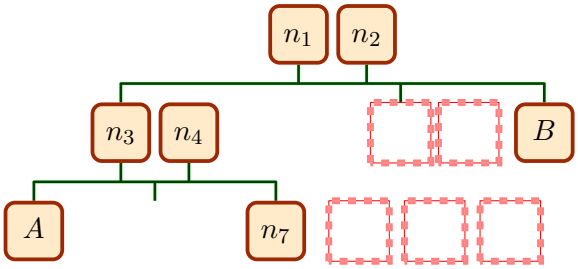
```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  content interpreter content={\gtrifnodeid{\gtrnodeid}{n_{\gtrnodenumber}}},
]
{
  child{ g-p-
    child{ g-p-
      child{ g-p-c-c-c-}
      c-c-
      child{ g-p-c-c-
        child{ g-p-c-c-c-}
      }
    }
    child[phantom=3cm]{g[id=P1]-c[id=P2]-}
    child{ g-p-c-
      child{ g-p-
        child{ g-p-c-c-c-c-}
        c-c-
      }
    }
  }
}
\path[draw,top color=yellow!50,bottom color=blue!50]
  (P2.south west) rectangle node {Phantom Area} (P1.north east);
\end{tikzpicture}
```



`/gtr/phantom*=`⟨*length*⟩ (style, default `/gtr/node size` → P. 83) U 2015-06-08

Identical to `/gtr/phantom` → P. 122, but the phantom subtree is connected by an edge with its embedding family.

```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  content interpreter content={\gtrifnodeid{\gtrnodeid}{n_{\gtrnodenumber}}},
]
{
  child{ g-p-
    child{ g-p-
      c[id=A]-
      % invisible phantom
      c[phantom*=2cm]-
      c-
    }
    % phantom; borders made visible
    child[phantom*,subtree box={show bounding box}]{
      g-p-c-c-c-
    }
    c[id=B]-
  }
}
\end{tikzpicture}
```



5.14 Special and Auxiliary Options

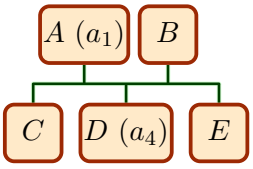
`/gtr/reset` (no value)

Resets all options to their default values.

`/gtr/code=`⟨*code*⟩ (no default)

The given ⟨*code*⟩ is executed immediately. This option is useful to place some arbitrary code into an option list.

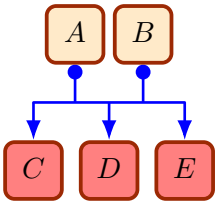
```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  code={\newcommand{\mycom}{(a_{\gtrnodenumber})}},
]{
  child{
    g{A~\mycom}
    p{B}
    c{C}   c{D~\mycom}  c{E}
  }
}
\end{tikzpicture}
```



`/gtr/keysfrom=`⟨*macro*⟩ (no default)

The given ⟨*macro*⟩ (without parameters) is supposed to contain an option list. The keys from the list are applied.

```
\newcommand{\mylist}{
  level distance=1cm,
  level -1/.style={node box={colback=red!50}},
  edges={no background,foreground={blue,Circle-Latex}},
}
%
\begin{tikzpicture}
\genealogytree[template=formal graph, keysfrom=\mylist
]{
  child{
    g{A}
    p{B}
    c{C}   c{D}  c{E}
  }
}
\end{tikzpicture}
```



6

Node Data (Content) Processing

Every node in a `\genealogytree` → P. 55 graph is drawn inside a rectangular box. These boxes are arranged by the auto-layout algorithm to build the entire graph.

The interior of a node box is created by an element called `/gtr/node processor` → P. 128. Several customizable node processors are predefined by the package to choose from. Further, an own node processor can be added easily.

The node data may be used *as-is* or changed in some way before the node processor displays it. This is done by an element called `/gtr/content interpreter` → P. 145. Again, several content interpreters are predefined by the package to choose from and own interpreters can be added.

The combination of node interpreter and node processor is called *node data processing* in the following.

Two classes of node processings can be distinguished:

- Non-interpreting node data processings take their content text *as-is* and just format it with colors, fonts, frames, etc; see Section 6.2 on page 129.
- Interpreting node data processings use some `/gtr/content interpreter` → P. 145 to possibly change the content.
  - The most prominent processing is database node processing where the node content is interpreted as organized data. Some representation of the data will form the visual output; see Chapter 7 on page 151.
  - Further interpreters are documented in Section 6.4 on page 145.

6.1 Setting a Node Data Processing and Processor

In this context, there is a small difference between *node data processing* and a *node data processors*. The *processing* is the combination of an *node data interpreter* and a *node data processors*. If the interpreter does not change the node data, the difference vanishes.

`/gtr/node processor`=⟨*macro*⟩ (no default)

Sets a ⟨*macro*⟩ for processing the content of a node. This ⟨*macro*⟩ has to be defined without parameters. It should display the node content which is stored in `\gtrBoxContent` → P. 144.

```
\newcommand{\myprocessor}{%
  \tikz\node[outer sep=0pt]{\gtrBoxContent};%
}

\gtrset{node processor=\myprocessor}
```

This option is useful for authors who wish to implement some very specific node processing (drawing) which is not covered by the standard mechanisms. See `/gtr/processing` for the standard processors. Since the standard processors are highly customizable, there may be no need to create a specific processor for most use cases.

A predefined `/gtr/node processor` is set by using `/gtr/processing` which also sets a `/gtr/content interpreter` → P. 145.

`/gtr/processing`=⟨*processing*⟩ (no default, initially `fit`)

Defines the base procedure for processing the content of a node. Feasible values for ⟨*processing*⟩ are

- `fit`: The content is set *as-is* inside a `\tcboxfit` macro from the `tcolorbox` package, see Section 6.2.1 on page 129.
- `tcolorbox`: The content is set *as-is* inside a `tcolorbox` environment from the `tcolorbox` package, see Section 6.2.2 on page 133.
- `tcbox`: The content is set *as-is* inside a `\tcbox` macro from the `tcolorbox` package, see Section 6.2.3 on page 136.
- `tcbox*`: As a variant to `tcbox`, the content is also set *as-is* inside a `\tcbox` macro from the `tcolorbox` package, see Section 6.2.4 on page 139.
- `tikznode`: The content is set *as-is* inside a `\node` macro from the `tikz` package, see Section 6.2.5 on page 142.
- `database`: The content is interpreted as database key-value pairs. The processed content is set inside a `\tcboxfit` macro from the `tcolorbox` package, see Chapter 7 on page 151.

Values given to `/gtr/box` → P. 96 will be interpreted according to the definded ⟨*processing*⟩. For `tcolorbox`, the values have to be `tcolorbox` settings; for `tikznode`, the values have to be `tikz` settings.

6.2 Predefined Non-Interpreting Processings

6.2.1 fit

`/gtr/processing`→P. 128`=fit`

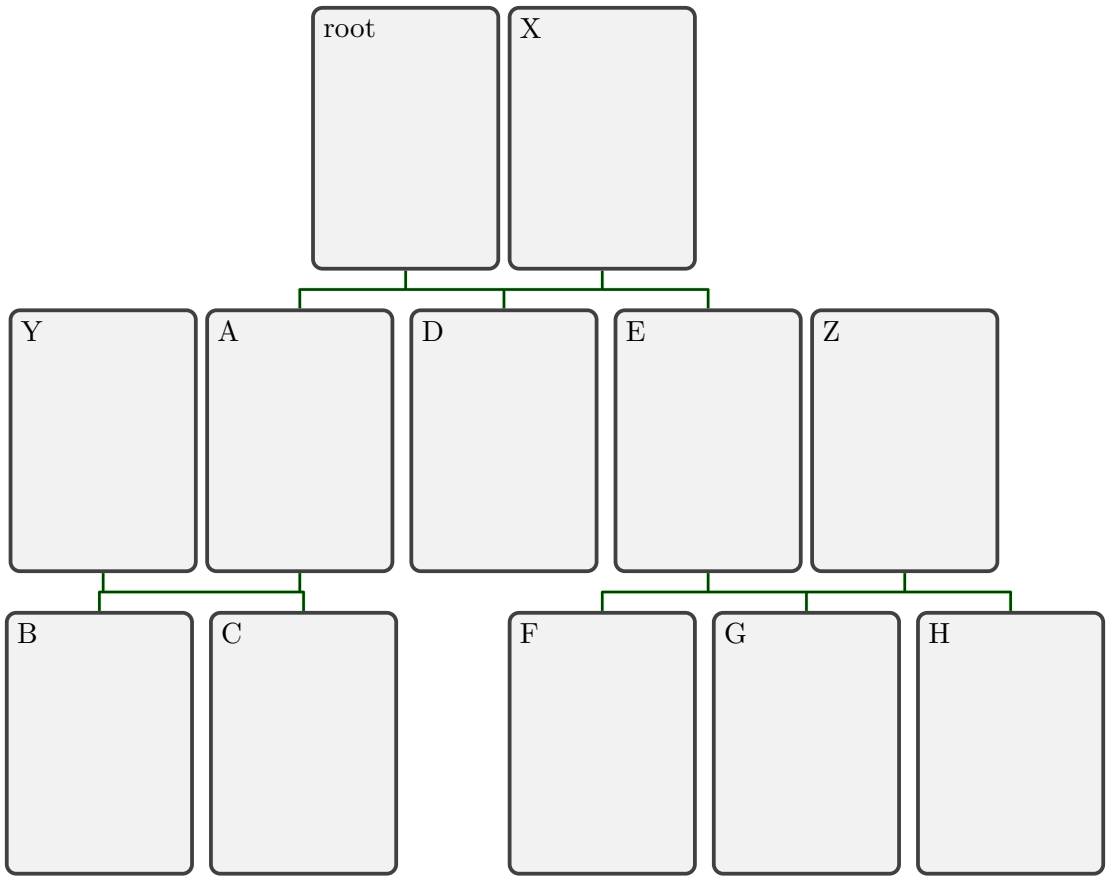
The preset processing is based on `\tcboxfit` of the `tcolorbox` package [3]. Options given to `/gtr/box`→P. 96 have to be `tcolorbox` options which are used by `\tcboxfit`.

The `/gtr/no content interpreter`→P. 147 is used. The main characteristics of the applied node data processor are:

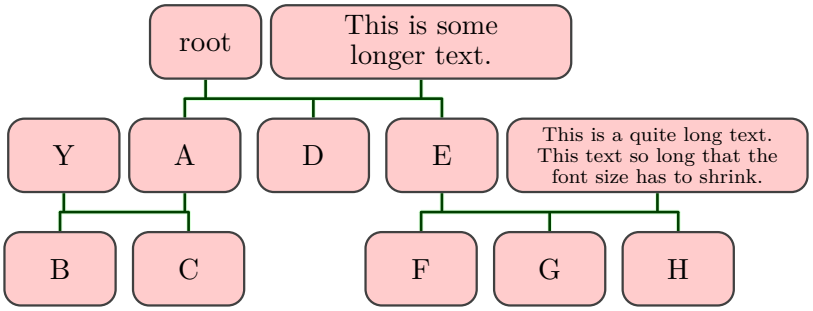
- Full observance of `/gtr/level size`→P. 82, `/gtr/node size`→P. 83, and `/gtr/node size from`→P. 84. These options can be used without restriction.
- The node content is set inside a `minipage`. The text size of the content and the margins are shrunk automatically, if needed. The used font should be freely scalable for this.
- Due to the fit algorithm, this node processing will consume more compilation time than other ones.
- To observe node and level settings as far as possible, the dimensions can be set by `/tcb/gtrNodeDimensions` or `/tcb/gtrNodeDimensionsLandscape`. `/tcb/gtrNodeDimensions` is initially set.

This processor is also used for database processing, see Chapter 7 on page 151.

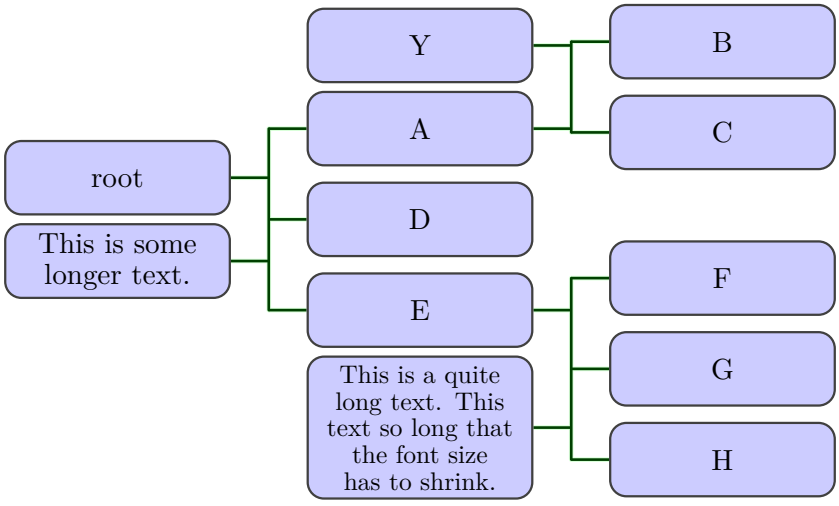
```
\begin{genealogypicture}[processing=fit]
  child{ g{root} p{X}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```



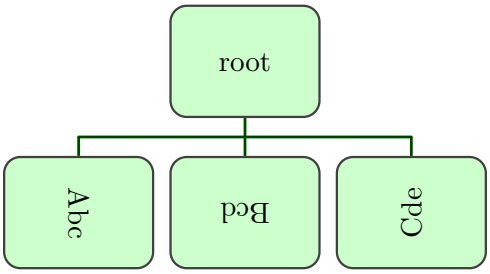
```
\begin{genealogypicture}[processing=fit,
      level size=1cm,node size from=1.5cm to 4cm,
      box={halign=center,valign=center,size=small,arc=2mm,colback=red!20}]
  child{ g{root} p{This is some longer text.}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E}
          p{This is a quite long text. This text so long that
            the font size has to shrink.}
          c{F} c{G} c{H} }
  }
\end{genealogypicture}
```




```
\begin{genealogypicture}[processing=fit,
      timeflow=right,
      level size=3cm,level distance=10mm,
      node size from=1cm to 2cm,
      box={halign=center,valign=center,size=small,arc=2mm,colback=blue!20}]
%
  child{ g{root} p{This is some longer text.}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E}
          p{This is a quite long text. This text so long that
            the font size has to shrink.}
          c{F} c{G} c{H} }
  }
\end{genealogypicture}
```



```
\begin{genealogypicture}[processing=fit,
      level size=1.5cm,level distance=5mm,node size=2cm,
      box={halign=center,valign=center,size=small,arc=2mm,colback=green!20}]
%
  child{ g{root}
    c[turn=left]{Abc}
    c[turn=upsidedown]{Bcd}
    c[turn=right]{Cde}
  }
\end{genealogypicture}
```



6.2.2 tcolorbox

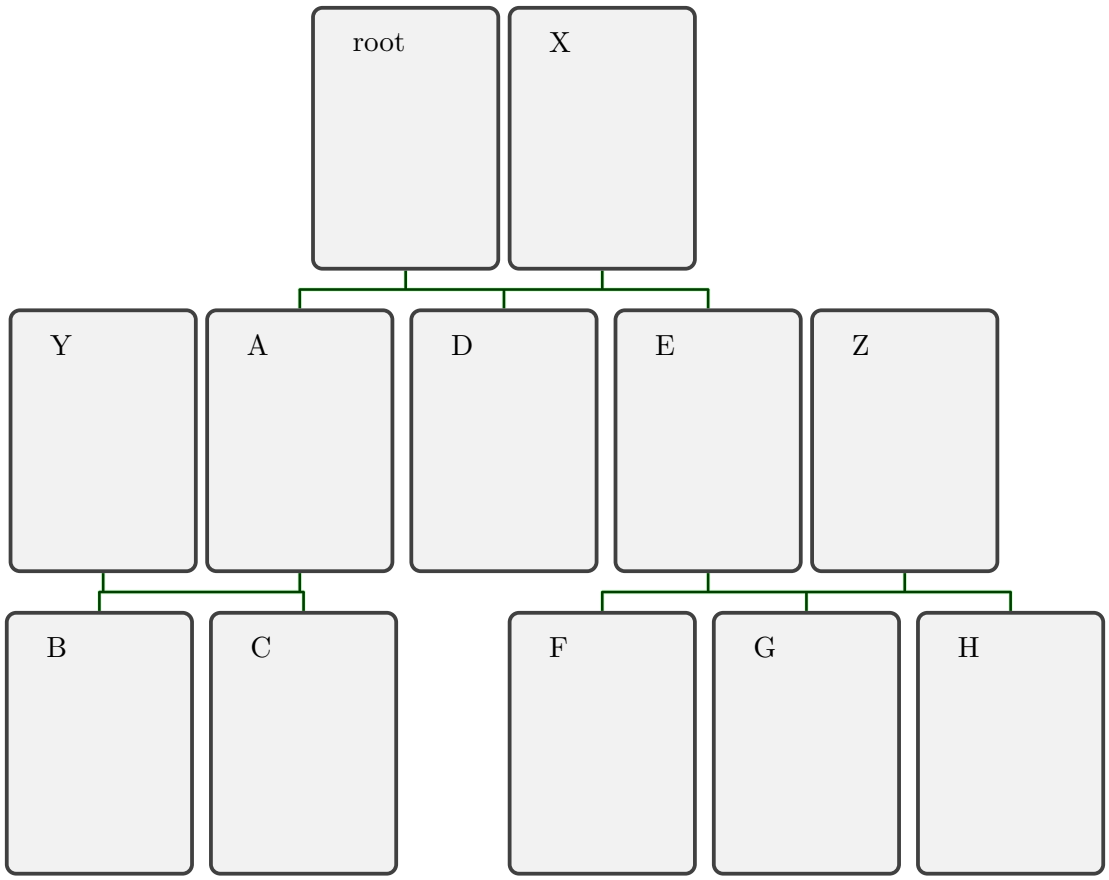
/gtr/processing→P. 128=tcolorbox

This processing is based on the tcolorbox environment of the tcolorbox package [3]. Options given to /gtr/box→P. 96 have to be tcolorbox options.

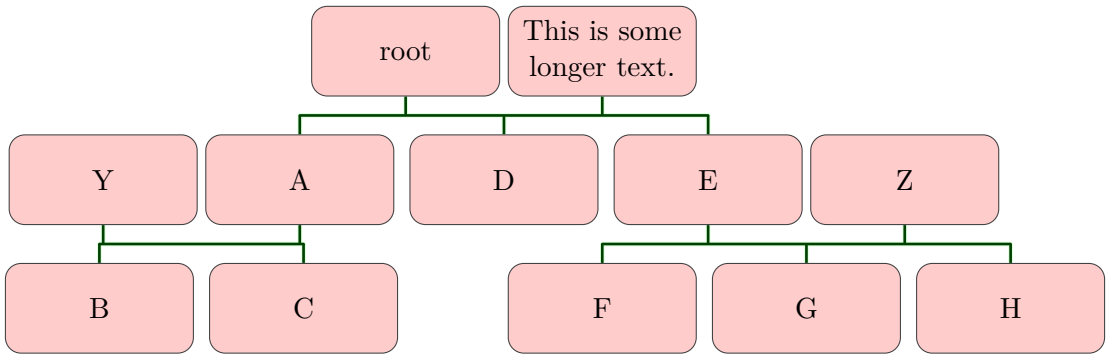
The /gtr/no content interpreter→P. 147 is used. The main characteristics of the applied node data processor are:

- For /gtr/timeflow→P. 78 settings up and down, full observance of /gtr/node size→P. 83, but no observance of /gtr/node size from→P. 84. The /gtr/level size→P. 82 is observed, but content which is too large may overflow.
- For /gtr/timeflow→P. 78 settings left and right, full observance of /gtr/level size→P. 82. /gtr/node size→P. 83 and /gtr/node size from→P. 84 are both observed, but content which is too large may overflow.
- Using the option natural height, the height of a node box can be freely adapted to its content. This may be especially useful for /gtr/timeflow→P. 78 settings left and right, but with some limited use for /gtr/timeflow→P. 78 settings up and down.
- Extremely customizable using options.
- To observe node and level settings as far as possible, the dimensions can be set by /tcb/gtrNodeDimensions or /tcb/gtrNodeDimensionsLandscape. /tcb/gtrNodeDimensions is initially set.

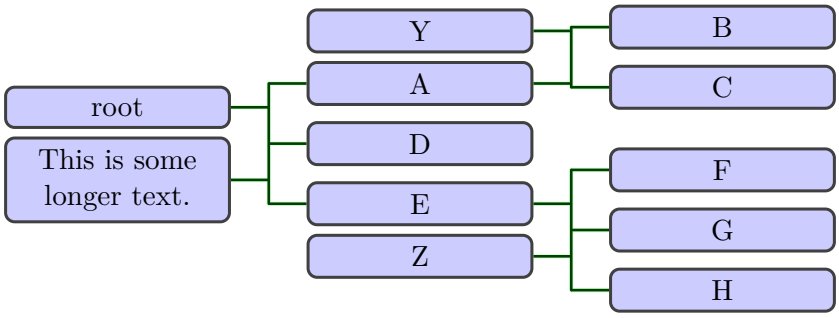
```
\begin{genealogypicture}[processing=tcolorbox]
  child{ g{root} p{X}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```



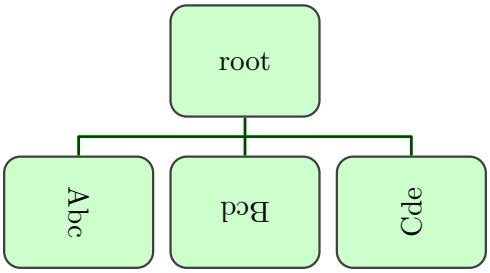

```
\begin{genealogypicture}[processing=tcolorbox,
    level size=1.2cm,node size=2.5cm,
    box={halign=center,valign=center,size=fbox,arc=2mm,colback=red!20} ]
  child{ g{root} p{This is some longer text.}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```



```
\begin{genealogypicture}[processing=tcolorbox,
    timeflow=right,
    level size=3cm,level distance=10mm,
    box={halign=center,natural height,size=title,arc=1mm,colback=blue!20} ]
  child{ g{root} p{This is some longer text.}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```




```
\begin{genealogypicture}[processing=tcolorbox,
      level size=1.5cm,level distance=5mm,node size=2cm,
      box={halign=center,valign=center,size=small,arc=2mm,colback=green!20}]
%
  child{ g{root}
    c[turn=left]{Abc}
    c[turn=upsidedown]{Bcd}
    c[turn=right]{Cde}
  }
\end{genealogypicture}
```



6.2.3 tcbox

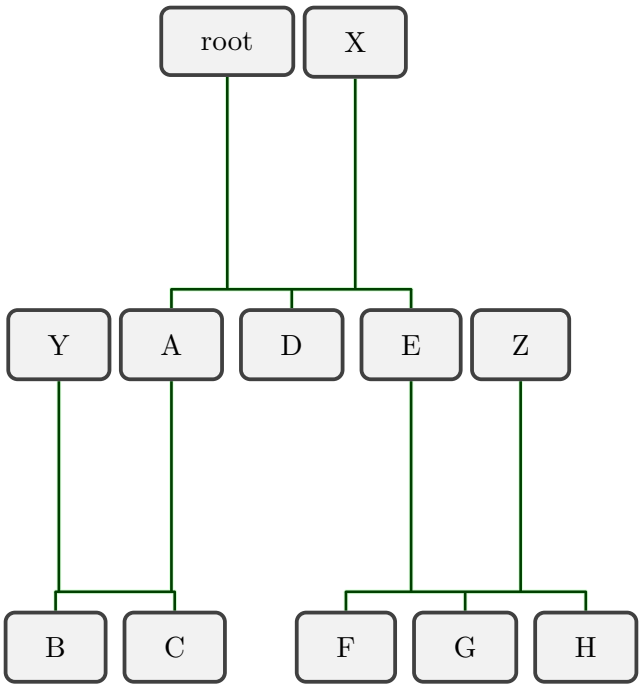
`/gtr/processing`→P. 128`=tcbox`

This processing is based on `\tcbox` of the `tcolorbox` package [3]. Options given to `/gtr/box`→P. 96 have to be `tcolorbox` options which are used by `\tcbox`.

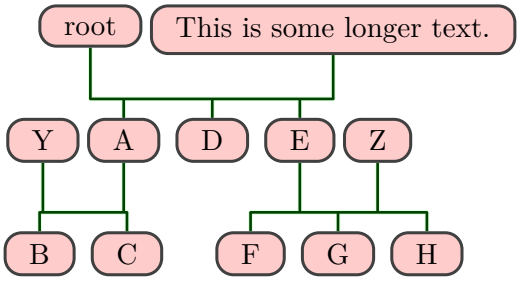
The `/gtr/no content interpreter`→P. 147 is used. The main characteristics of the applied node data processor are:

- For `/gtr/timeflow`→P. 78 settings `up` and `down`, no observance of `/gtr/node size`→P. 83 and `/gtr/node size from`→P. 84, but full observance of `/gtr/level size`→P. 82, if `/tcb/gtrNodeDimensions` is set.
- For `/gtr/timeflow`→P. 78 settings `left` and `right`, no observance of `/gtr/level size`→P. 82. `/gtr/node size`→P. 83 and `/gtr/node size from`→P. 84 are both observed, if `/tcb/gtrNodeDimensions` is set.
- If not specified otherwise by options, the content is set as a single line and the box is sized according to its content.
- To observe node and level settings as far as possible, the dimensions can be set by `/tcb/gtrNodeDimensions` or `/tcb/gtrNodeDimensionsLandscape`. `/tcb/gtrNodeDimensions` is initially *not* set, but `/gtr/turn`→P. 98 will switch dimensions settings on.

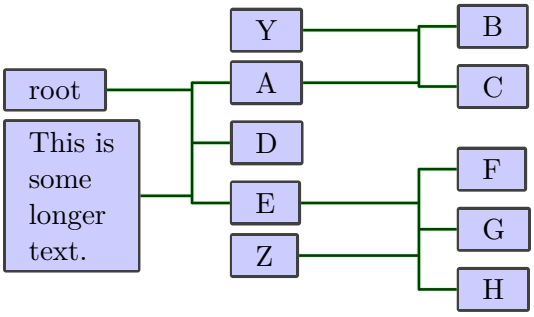
```
\begin{genealogypicture}[processing=tcbox]
  child{ g{root} p{X}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```



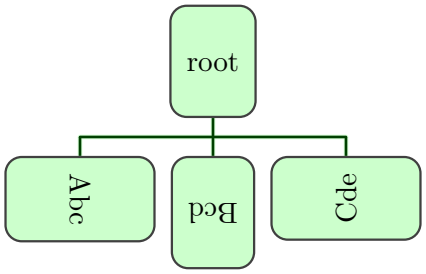
```
\begin{genealogypicture}[processing=tcbox,
    level size=1cm,
    box={valign=center,size=title,arc=2mm,colback=red!20} ]
  child{ g{root} p{This is some longer text.}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```



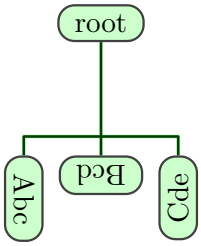

```
\begin{genealogypicture}[processing=tcbox,
    timeflow=right,
    level size=2cm,level distance=10mm,
    box={size=title,natural height,arc=0mm,colback=blue!20} ]
  child{ g{root} p[box={varwidth upper=\gtrNodeMaxWidth}]{This is some longer text.}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```



```
\begin{genealogypicture}[processing=tcbox,
      level size=1.5cm,level distance=5mm,node size=2cm,
      box={halign=center,valign=center,size=small,arc=2mm,colback=green!20}]
%
  child{ g[turn=off]{root}
    c[turn=left]{Abc}
    c[turn=upsidedown]{Bcd}
    c[turn=right]{Cde}
  }
\end{genealogypicture}
```




```
\begin{genealogypicture}[processing=tcbox,
      level size=1.5cm,level distance=5mm,node size=2cm,
      box={halign=center,valign=center,size=small,natural height,
           arc=2mm,colback=green!20}]
%
  child{ g{root}
    c[turn=left,box={natural height}]{Abc}
    c[turn=upsidedown,box={natural height}]{Bcd}
    c[turn=right,box={natural height}]{Cde}
  }
\end{genealogypicture}
```



6.2.4 tcbox*

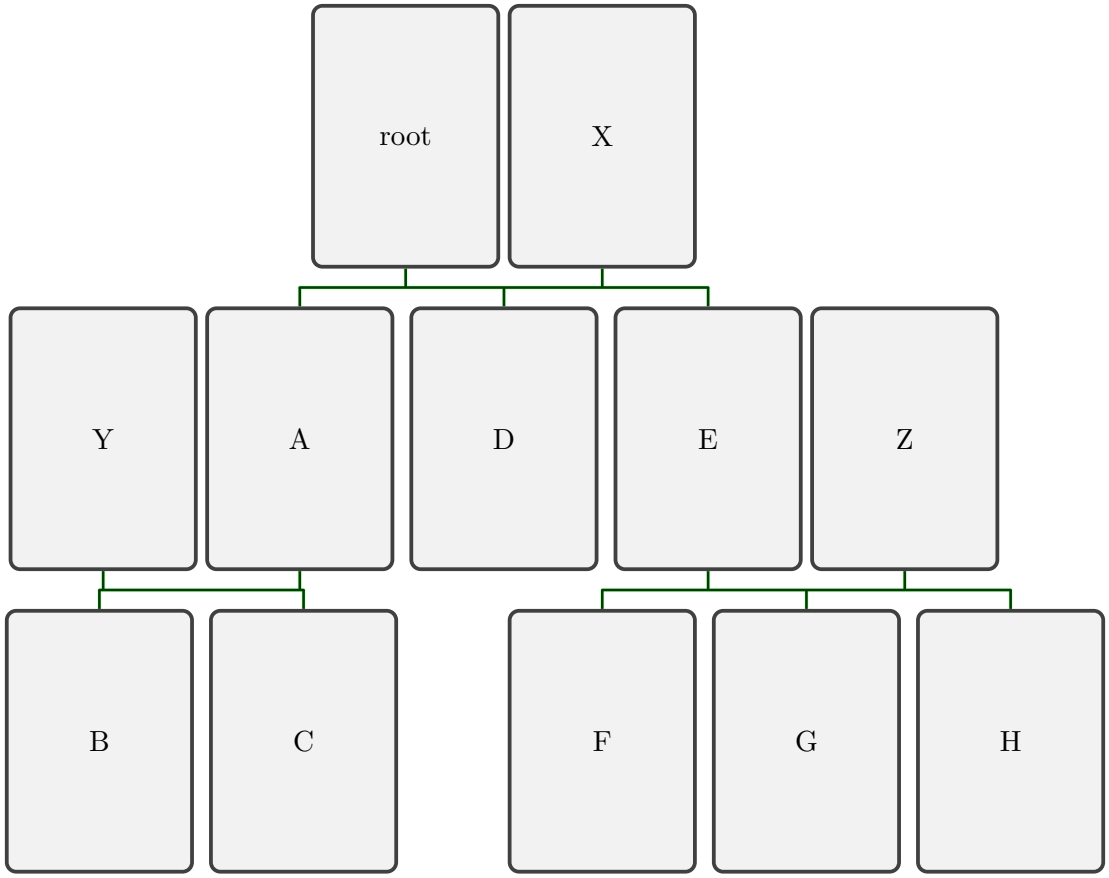
`/gtr/processing`→ P. 128`=tcbox*`

This processing is based on `\tcbox` of the `tcolorbox` package [3]. Options given to `/gtr/box`→ P. 96 have to be `tcolorbox` options which are used by `\tcbox`. This is a variant of Section 6.2.3 on page 136.

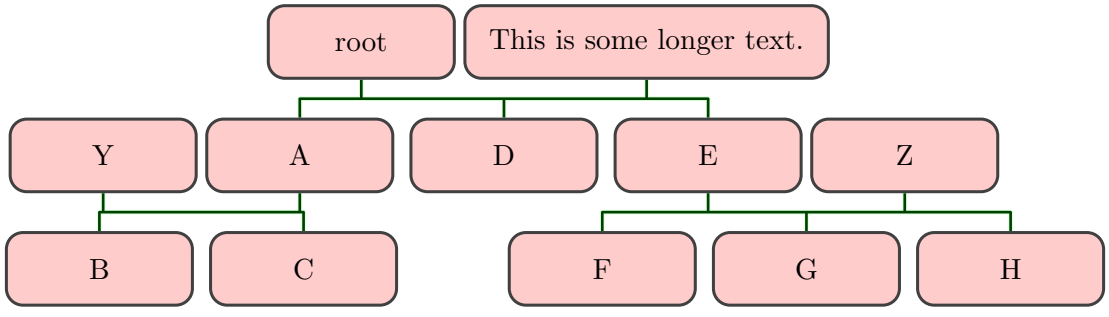
The `/gtr/no content interpreter`→ P. 147 is used. The main characteristics of the applied node data processor are:

- For `/gtr/timeflow`→ P. 78 settings `up` and `down`, observance of `/gtr/node size`→ P. 83 (but width may grow beyond) and full observance of `/gtr/level size`→ P. 82, if `/tcb/gtrNodeDimensions` is set.
- For `/gtr/timeflow`→ P. 78 settings `left` and `right`, some observance of `/gtr/level size`→ P. 82 (but width may grow beyond). `/gtr/node size`→ P. 83 and `/gtr/node size from`→ P. 84 are both observed, if `/tcb/gtrNodeDimensions` is set.
- If not specified otherwise by options, the content is set horizontally and vertically centered as a single line.
- To observe node and level settings as far as possible, the dimensions can be set by `/tcb/gtrNodeDimensions` or `/tcb/gtrNodeDimensionsLandscape`.
  `/tcb/gtrNodeDimensions` is initially set.

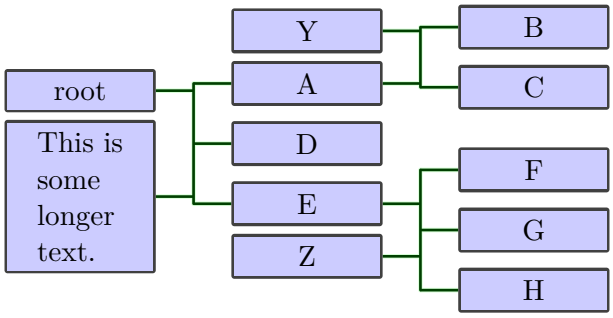
```
\begin{genealogypicture}[processing=tcbox*]
  child{ g{root} p{X}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```



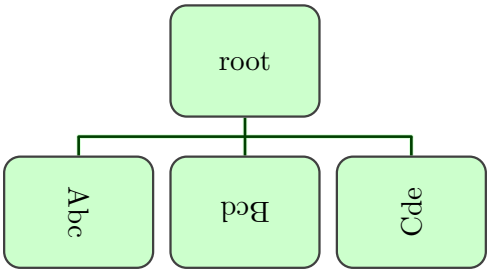

```
\begin{genealogypicture}[processing=tcbox*,
    level size=1cm,
    box={valign=center,size=title,arc=2mm,colback=red!20} ]
  child{ g{root} p{This is some longer text.}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```



```
\begin{genealogypicture}[processing=tcbox*,
    timeflow=right,
    level size=2cm,level distance=10mm,
    box={size=title,natural height,arc=0mm,colback=blue!20} ]
  child{ g{root} p[box={varwidth upper=\gtrNodeMaxWidth}]{This is some longer text.}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```




```
\begin{genealogypicture}[processing=tcbox*,
      level size=1.5cm,level distance=5mm,node size=2cm,
      box={halign=center,valign=center,size=small,arc=2mm,colback=green!20}]
%
  child{ g[turn=off]{root}
    c[turn=left]{Abc}
    c[turn=upsidedown]{Bcd}
    c[turn=right]{Cde}
  }
\end{genealogypicture}
```



6.2.5 tikznode

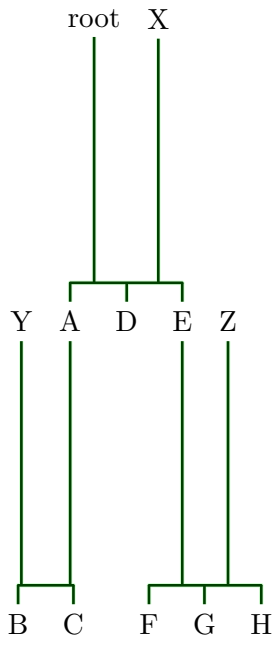
`/gtr/processing`→ P. 128`=tikznode`

This processing is based on `\node` of the `tikz` package [4]. Options given to `/gtr/box`→ P. 96 have to be `tikz` options which are used by `\node`.

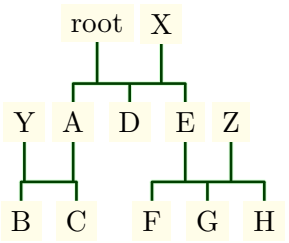
The `/gtr/no content interpreter`→ P. 147 is used. The main characteristics of the applied node data processor are:

- No observance of `/gtr/level size`→ P. 82, `/gtr/node size`→ P. 83, and `/gtr/node size from`→ P. 84.
- Not as customizable as other processors, but full `tikz` options.
- This node processing will consume the smallest compilation time.
- To observe node and level settings as far as possible, the dimensions can be set by `/tikz/gtrNodeDimensions` or `/tikz/gtrNodeDimensionsLandscape`. `/tikz/gtrNodeDimensions` is initially *not* set, but `/gtr/turn`→ P. 98 will switch dimensions settings on.

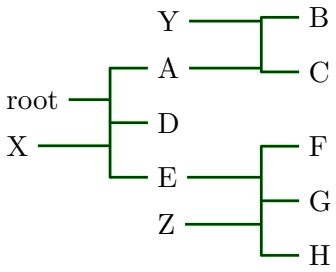
```
\begin{genealogypicture}[processing=tikznode]
  child{ g{root} p{X}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```



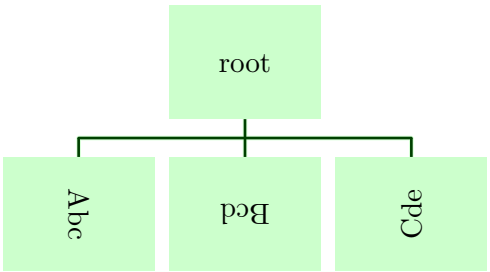
```
\begin{genealogypicture}[processing=tikznode,
    level size=8mm,box={fill=yellow!10}, ]
  child{ g{root} p{X}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```




```
\begin{genealogypicture}[processing=tikznode,
    timeflow=right,
    level size=1cm,level distance=10mm ]
  child{ g{root} p{X}
    child{ p{Y} g{A} c{B} c{C} }
    c{D}
    child{ g{E} p{Z} c{F} c{G} c{H} }
  }
\end{genealogypicture}
```




```
\begin{genealogypicture}[processing=tikznode,
      level size=1.5cm,level distance=5mm,node size=2cm,
      box={fill=green!20,gtrNodeDimensions}]
%
  child{ g{root}
    c[turn=left]{Abc}
    c[turn=upsidedown]{Bcd}
    c[turn=right]{Cde}
  }
\end{genealogypicture}
```



6.3 Creating a Customized Non-Interpreting Processor

For most applications, one if the predefined non-interpreting processings with their processors will suffice, see Section 6.2 on page 129.

But using `/gtr/node processor` → P. 128, also a new node processor can be defined.

```
\gtrset{node processor=\myprocessor}
```

Here, `\myprocessor` is an own macro which has to be defined without parameters. Inside the macro definition, the following can be used.

`\gtrBoxContent`
Contains the (already interpreted or not interpreted) node content.

`\gtrNodeMinWidth`
Contains the current target minimum node width as defined by the various tree settings.

`\gtrNodeMaxWidth`
Contains the current target maximum node width as defined by the various tree settings.

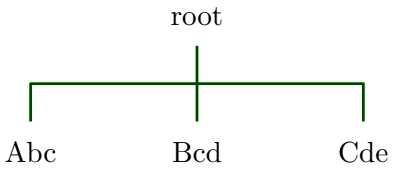
`\gtrNodeMinHeight`
Contains the current target minimum node height as defined by the various tree settings.

`\gtrNodeMaxHeight`
Contains the current target maximum node height as defined by the various tree settings.

`\gtrNodeBoxOptions`
Contains the option settings for the current node. These are the assembled `/gtr/box` → P. 96 settings as comma separated key-value list for the current node.

For demonstration, a simple processor based on the `minipage` environment is constructed:

```
\newcommand{\myprocessor}{%
  \begin{minipage}[c][\gtrNodeMinHeight]{\gtrNodeMinWidth}%
    \begin{center}\gtrBoxContent\end{center}\end{minipage}%
}%
%
\begin{genealogypicture}[node processor=\myprocessor,
      level size=8mm,level distance=10mm,node size=2cm]
%
  child{ g{root}
    c{Abc}
    c{Bcd}
    c{Cde}
  }
\end{genealogypicture}
```



6.4 Content Interpreters

The predefined non-interpreting processings from Section 6.2 on page 129 can easily adapted to become interpreting, if /gtr/content interpreter or /gtr/content interpreter code→ P. 146 is set. The interpreter changes the node content somehow (see Chapter 7 on page 151 for the main example) and gives the changed content to the chosen /gtr/node processor→ P. 128.

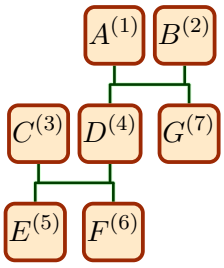
/gtr/content interpreter=⟨*macro*⟩ (no default)

Sets ⟨*macro*⟩ for interpreting the content of a node. This ⟨*macro*⟩ has to take one mandatory parameter (the original box content). It has to define a new parameterless macro \gtrBoxContent→ P. 144 which should store the content which is given to the current /gtr/node processor→ P. 128 for further compilation.

```
\gtrset{content interpreter=\myinterpreter}
```

The most important interpreter is realized by database processing, see Chapter 7 on page 151. This option may be used to implement an own kind of database processing which differs from the package implementation.
Another use case is to replace the node content completely by some automated content like numbering the nodes.

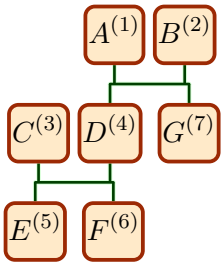
```
\newcommand{\myinterpreter}[1]{\def\gtrBoxContent{#1^{(\gtrnodenumber)}}}

\begin{tikzpicture}
\genealogytree[
  template=formal graph,
  content interpreter=\myinterpreter  ]
{
  child{
    g{A} p{B}
    child{ p{C} g{D} c{E} c{F} }
    c{G}
  }
}
\end{tikzpicture}
```



`/gtr/content interpreter code={⟨code⟩}` (no default)

Sets ⟨*code*⟩ for interpreting the content of a node. This ⟨*code*⟩ can use a parameter `#1` (the original box content) and has to define a new parameterless macro `\gtrBoxContent` → P. 144 which should store the content which is given to the current `/gtr/node processor` → P. 128 for further compilation.

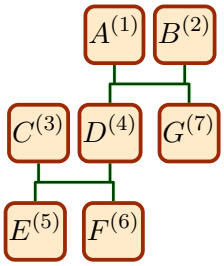
```
\begin{tikzpicture}
\genealogytree[
  template=formal graph,
  content interpreter code={\def\gtrBoxContent{#1^{(\gtrnodenumber)}}}  ]
{
  child{
    g{A} p{B}
    child{ p{C} g{D} c{E} c{F} }
    c{G}
  }
}
\end{tikzpicture}
```



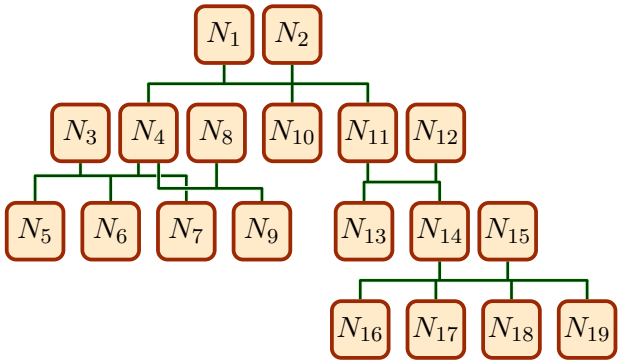
`/gtr/content interpreter content={⟨code⟩}` (no default)

Sets ⟨*code*⟩ for interpreting the content of a node. This ⟨*code*⟩ is the definition for `\gtrBoxContent` → P. 144. The ⟨*code*⟩ can use a parameter `#1` (the original box content).

```
\begin{tikzpicture}
\genealogytree[
  template=formal graph,
  content interpreter content={#1^{(\gtrnodenumber)}}  ]
{
  child{
    g{A} p{B}
    child{ p{C} g{D} c{E} c{F} }
    c{G}
  }
}
\end{tikzpicture}
```



```
\begin{tikzpicture}
\genealogytree[
  template=formal graph,
  content interpreter content={N_{\gtrnodenumber}} ]
{
  child{
    g- p-
    child{ p- g- c- c- c- union{ p- c- } }
    c-
    child{ g- p- c- child{ g- p- c- c- c- c- } }
  }
}
\end{tikzpicture}
```



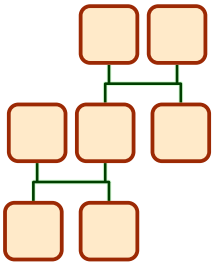
/gtr/no content interpreter (no value, initially set)

Virtually removes any content interpreter. The node content is given directly to the current /gtr/processing → P. 128 for further compilation. Actually, this defines \gtrBoxContent → P. 144 to contain the original box content.

/gtr/deletion content interpreter (no value, initially set)

Deletes any box content. This leads to empty boxes.

```
\begin{tikzpicture}
\genealogytree[
  template=formal graph,
  deletion content interpreter  ]
{
  child{
    g{A} p{B}
    child{ p{C} g{D} c{E} c{F} }
    c{G}
  }
}
\end{tikzpicture}
```



`/gtr/database content interpreter` (no value, initially set)

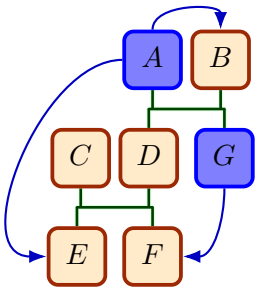
This is the content interpreter for database processing, see Chapter 7 on page 151.

```
% ...  make database processing dependend on tcbox*
  processing=tcbox*,
  database content interpreter,
% ...
```

`/gtr/id content interpreter` (no value, initially set)

The box content is not only used *as-is* but is also set as `/gtr/id` → P. 90 of the node. This implies that no macro code is used inside the nodes.

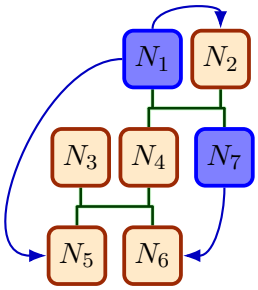
```
\begin{tikzpicture}
\genealogytree[
  template=formal graph,
  id content interpreter,
  options for node={A,G}{box={colback=blue!50,colframe=blue}}  ]
{
  child{
    g{A} p{B}
    child{ p{C} g{D} c{E} c{F} }
    c{G}
  }
}
\draw[-Latex,blue!75!black,thick]
  (A) edge[out=180,in=180] (E)
      edge[out=90,in=90]   (B)
  (G) edge[out=270,in=0]   (F) ;
\end{tikzpicture}
```



`/gtr/content interpreter id and content={⟨id⟩}{⟨code⟩}` (no default)

Sets ⟨*code*⟩ for interpreting the content of a node. This ⟨*code*⟩ is the definition for `\gtrBoxContent` → P. 144. Also, the ⟨*id*⟩ for the node is set. The ⟨*code*⟩ and ⟨*id*⟩ can use a parameter `#1` (the original box content). Note that ⟨*id*⟩ will be fully expanded.

```
\begin{tikzpicture}
\genealogytree[
  template=formal graph,
  content interpreter id and content={n\gtrnodenumber}{N_{\gtrnodenumber}},
  options for node={n1,n7}{box={colback=blue!50,colframe=blue}}  ]
{
  child{
    g-p-
    child{ p-g-c-c- }
    c-
  }
}
\draw[-Latex,blue!75!black,thick]
  (n1) edge[out=180,in=180] (n5)
       edge[out=90,in=90]   (n2)
  (n7) edge[out=270,in=0]   (n6) ;
\end{tikzpicture}
```



7

Database Processing

Database processing is a specialized node data processing, see Chapter 6 on page 127. The node content is interpreted as organized data and some representation of the data will form the visual output.

To switch to database processing, use

```
/gtr/processing→P. 128=database
```

The box content is interpreted as key-value database list. The actual box construction is based on `\tcboxfit` of the `tcolorbox` package [3]. Options given to `/gtr/box`→P. 96 have to be `tcolorbox` options which are used by `\tcboxfit`.

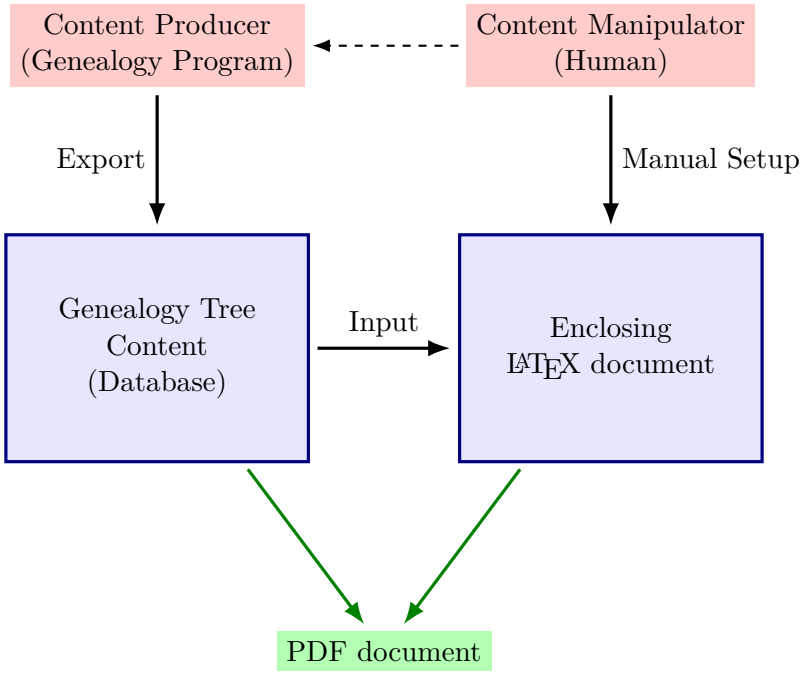
The `/gtr/database content interpreter`→P. 148 is used in combination with the node data processor described in Section 6.2.1 on page 129.

For a quick example-based overview, see the full samples in Section 7.5 on page 162 which use the data given in Section 7.2 on page 153.

7.1 Database Concept

The general idea of this *database* approach is to separate the data content of a node from the formatting. While this is also a common TEX/LATEX idea, the following concept goes somewhat further.

The content producer could be a human person directly, but more presumably a machine like a genealogy program. The node content is written as a comma separated key-value list. This list is processed and its content formatted by a database processor. For a quick survey with an example, see Section 7.2 on page 153.



The content is exported by a program or hand written as key-value list. The format of this list is described in Section 7.3 on page 155. This list is processed by an enclosing LATEX document which is created and manipulated by a human. This enclosing document specifies how the content is displayed. This relieves the exporting program from caring about formatting issues and gives full visual control to a human author. The author is relieved from putting down data by hand which presumably is already data-processed with a genealogy program.

Also, the following methods allow to use the same database for different diagrams with possibly different goals and designs.

7.2 Example Settings

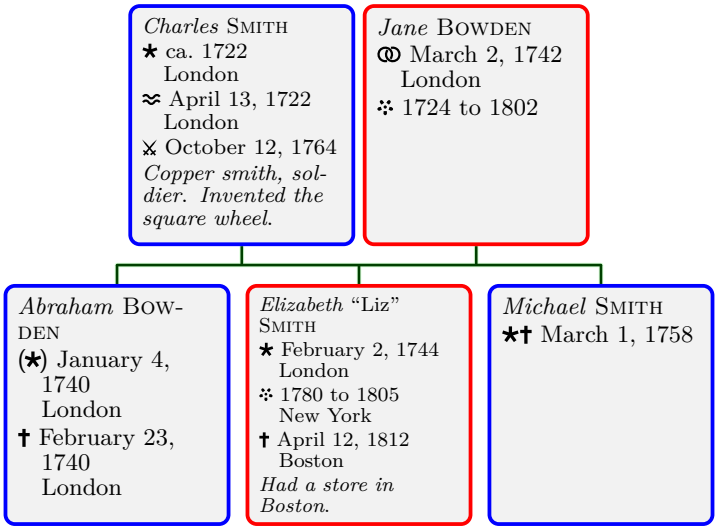
This example data is used in the following (also documented in Section 14.2 on page 288).

File «example.database.graph» for the following examples

```
child[id=SmitBowd1742]{
  g[id=SmitChar1722]{
    male,
    name       = {\pref{Charles} \surn{Smith}},
    birth      = {(caAD)1722}{London},
    baptism    = {1722-04-13}{London},
    death+     = {1764-10-12}{}{killed},
    profession = {Copper smith, soldier},
    comment    = {Invented the square wheel},
  }
  p[id=BowdJane1724]{
    female,
    name     = {\pref{Jane} \surn{Bowden}},
    floruit- = {1724/1802},
    marriage = {1742-03-02}{London},
  }
  c[id=BowdAbra1740]{
    male,
    name   = {\pref{Abraham} \surn{Bowden}},
    birth+ = {1740-01-04}{London}{out of wedlock},
    death  = {1740-02-23}{London}
  }
  c[id=SmitEliz1744]{
    female,
    name    = {\pref{Elizabeth} \nick{Liz} \surn{Smith}},
    birth   = {1744-02-02}{London},
    floruit = {1780/1805}{New York},
    death   = {1812-04-12}{Boston},
    comment = {Had a store in Boston},
  }
  c[id=SmitMich1758]{
    male,
    name   = {\pref{Michael} \surn{Smith}},
    birth+ = {1758-03-01}{}{died},
  }
}
```

Note especially the /gtr/id → P. 90 values. They are essential as handle to access a singular node from an importing document without changing the database.

```
\begin{genealogypicture}[
  processing=database,database format=full,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style={solid},
    halign=left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



7.3 Data Keys

/gtr/database/name=⟨*full name*⟩ (no default, initially empty)

This key holds the ⟨*full name*⟩ of a person presumably with markup. For customization, the markup should be done with \pref→P. 172, \surn→P. 172, \nick→P. 172 instead of common LaTeX font settings.

```
%...
name = {\pref{Elizabeth} \nick{Liz} \surn{Smith}},
%...
```

- \pref→P. 172 marks a preferred given name.
- \nick→P. 172 marks a nickname.
- \surn→P. 172 marks a surname.

The saved data is accessible by **\gtrDBname**.

/gtr/database/shortname=⟨*short name*⟩ (no default, initially empty)

This key holds an optional ⟨*short name*⟩ of a person presumably with markup. For customization, the markup should be done with \pref→P. 172, \surn→P. 172, \nick→P. 172 instead of common LaTeX font settings.

```
%...
shortname = {\nick{Liz} \surn{Smith}},
%...
```

The saved data is accessible by **\gtrDBshortname**.

/gtr/database/sex=⟨*sex*⟩ (no default, initially `neuter`)

This key holds the ⟨*sex*⟩ of a person. Feasible values are **male** and **female**. **neuter** is an additional feasible default value, if the sex is unknown, e.g. for a stillborn child without further data. The saved data is accessible by **\gtrDBsex**.

```
%...
sex = female,
%...
```

/gtr/database/female (style, no value)

Shortcut for /gtr/database/sex=**female**.

/gtr/database/male (style, no value)

Shortcut for /gtr/database/sex=**male**.

/gtr/database/neuter (style, no value)

Shortcut for /gtr/database/sex=**neuter**.

/gtr/database/comment=⟨*text*⟩ (no default, initially empty)

This key holds some comment ⟨*text*⟩ about a person, e.g. occupation or a very concise life description. The saved data is accessible by **\gtrDBcomment**.

```
%...
comment = {Had a store in Boston},
%...
```

/gtr/database/profession=⟨*text*⟩ (no default, initially empty) N 2017-01-27

This key holds some ⟨*text*⟩ about the profession a person. The saved data is accessible by \gtrDBprofession.

```
%...
profession = {Copper smith, soldier},
%...
```

/gtr/database/image=⟨*file name*⟩ (no default, initially empty)

This key holds an image ⟨*file name*⟩ of a person's portrait. The saved data is accessible by \gtrDBimage.

```
%...
image = Marry_Smith_1720.jpg,
%...
```

/gtr/database/uuid=⟨*text*⟩ (no default, initially empty)

This key holds an *universally unique identifier* (UUID) ⟨*text*⟩ of a person. In contrast to /gtr/id → P. 90, the UUID should be globally constant. It may be used for interlinking beyond the scope of a genealogy tree diagram. The saved data is accessible by \gtrDBuuid.

```
%...
uuid = 1021aa0c-2508-488c-9760-f9f84b4df1dd,
%...
```

/gtr/database/kekule=⟨*number*⟩ (no default, initially empty)

This key holds the Kekulé number of a person. The saved data is accessible by \gtrDBkekule.

```
%...
kekule = 1024,
%...
```

/gtr/database/relationship=⟨*text*⟩ (no default, initially empty)

This key holds a relationship ⟨*text*⟩ describing the person. The saved data is accessible by \gtrDBrelationship.

```
%...
relationship = Grandfather,
%...
```

The following data keys hold *events*. Every *event* consists of

- a *date*, see Section 7.4 on page 160,
- optionally a *place*
- and sometimes a *modifier*.

The three main events are

- Birth,
- Marriage,
- Death.

The other events may or may not be considered for data formatting.

> The saved data for the events is accessible by `\gtrPrintEvent` → P. 179, `\gtrPrintDate` → P. 174, and `\gtrPrintPlace` → P. 178. The existence of data can be checked by `\gtrifdatedefined` → P. 174 and `\gtrifplacedefined` → P. 178.

`/gtr/database/birth={⟨date⟩}{⟨place⟩}` (no default)
This key holds a birth event with given ⟨*date*⟩ and ⟨*place*⟩.

```
%...
birth = {1744-02-02}{London},
%...
```

`/gtr/database/birth+={⟨date⟩}{⟨place⟩}{⟨modifier⟩}` (no default)
This key holds a birth event with given ⟨*date*⟩, ⟨*place*⟩, and a ⟨*modifier*⟩ to describe the event further. Feasible values for the ⟨*modifier*⟩ are

- *empty* (normal),
- `out of wedlock`,
- `stillborn`,
- `died`.

```
%...
birth+ = {1740-01-04}{London}{out of wedlock},
%...
```

`/gtr/database/birth-=⟨date⟩` (no default)
This key holds a birth event with given ⟨*date*⟩.

```
%...
birth- = {1744-02-02},
%...
```

`/gtr/database/baptism={⟨date⟩}{⟨place⟩}` (no default)
This key holds a baptism event with given ⟨*date*⟩ and ⟨*place*⟩.

`/gtr/database/baptism+={⟨date⟩}{⟨place⟩}{⟨modifier⟩}` (no default)
Identical to `/gtr/database/baptism` since there is no valid ⟨*modifier*⟩.

`/gtr/database/baptism-=⟨date⟩` (no default)
This key holds a baptism event with given ⟨*date*⟩.

`/gtr/database/engagement={⟨date⟩}{⟨place⟩}` (no default)
This key holds an engagement event with given ⟨*date*⟩ and ⟨*place*⟩.

`/gtr/database/engagement+={⟨date⟩}{⟨place⟩}{⟨modifier⟩}` (no default)
Identical to `/gtr/database/engagement` since there is no valid ⟨*modifier*⟩.

`/gtr/database/engagement-=⟨date⟩` (no default)
This key holds an engagement event with given ⟨*date*⟩.

`/gtr/database/marriage={⟨date⟩}{⟨place⟩}` (no default)
This key holds a marriage event with given ⟨*date*⟩ and ⟨*place*⟩.

`/gtr/database/marriage+={⟨date⟩}{⟨place⟩}{⟨modifier⟩}` (no default)
This key holds a marriage event with given ⟨*date*⟩, ⟨*place*⟩, and a ⟨*modifier*⟩ to describe the event further. Feasible values for the ⟨*modifier*⟩ are

- *empty* (normal),
- `other`.

`/gtr/database/marriage-=⟨date⟩` (no default)
This key holds a marriage event with given ⟨*date*⟩.

`/gtr/database/divorce={⟨date⟩}{⟨place⟩}` (no default)
This key holds a divorce event with given ⟨*date*⟩ and ⟨*place*⟩.

`/gtr/database/divorce+={⟨date⟩}{⟨place⟩}{⟨modifier⟩}` (no default)
Identical to `/gtr/database/divorce` since there is no valid ⟨*modifier*⟩.

`/gtr/database/divorce-=⟨date⟩` (no default)
This key holds a divorce event with given ⟨*date*⟩.

`/gtr/database/floruit={⟨date⟩}{⟨place⟩}` (no default) N 2017-07-18
This key holds a floruit event with given ⟨*date*⟩ and ⟨*place*⟩.

`/gtr/database/floruit+={⟨date⟩}{⟨place⟩}{⟨modifier⟩}` (no default) N 2017-07-18
Identical to `/gtr/database/floruit` since there is no valid ⟨*modifier*⟩.

`/gtr/database/floruit-={⟨date⟩}{⟨place⟩}` (no default) N 2017-07-18
This key holds a floruit event with given ⟨*date*⟩.

`/gtr/database/death={⟨date⟩}{⟨place⟩}` (no default)
This key holds a death event with given ⟨*date*⟩ and ⟨*place*⟩.

`/gtr/database/death+={⟨date⟩}{⟨place⟩}{⟨modifier⟩}` (no default)
This key holds a death event with given ⟨*date*⟩, ⟨*place*⟩, and a ⟨*modifier*⟩ to describe the event further. Feasible values for the ⟨*modifier*⟩ are

- *empty* (normal),
- `killed`.

`/gtr/database/death-=⟨date⟩` (no default)
This key holds a death event with given ⟨*date*⟩.

`/gtr/database/burial={`⟨*date*⟩`}{`⟨*place*⟩`}` (no default)

This key holds a burial event with given ⟨*date*⟩ and ⟨*place*⟩.

`/gtr/database/burial+={`⟨*date*⟩`}{`⟨*place*⟩`}{`⟨*modifier*⟩`}` (no default)

This key holds a burial event with given ⟨*date*⟩, ⟨*place*⟩, and a ⟨*modifier*⟩ to describe the event further. Feasible values for the ⟨*modifier*⟩ are

- *empty* (normal),
- `cremated`.

`/gtr/database/burial-=`⟨*date*⟩ (no default)

This key holds a burial event with given ⟨*date*⟩.

`/gtr/database unknown key=`⟨*option*⟩ (no default, initially `warn`)

The node data may contain more key-value pairs than needed for the current processing. This option controls how the package should react when detecting unknown keys. Feasible ⟨*option*⟩ values are

- `ignore`: ignore unknown keys,
- `warn`: warn about unknown keys,
- `error`: stop processing at unknown keys,
- `save`: store the value of an unknown key. If a key `dummy` is detected, its value is stored under `/gtr/database/save/dummy`.

7.4 Input Format for Dates

A *date* can be given as a *single date* or as a *date range*. A *single date* is specified in the format

`(c)yyyy-mm-dd`

with calendar `c`, year `yyyy`, month `mm`, and day `dd`. The calendar `c` flag is optional and can be

- `AD`: Anno Domini; this is the default setting, if the calender flag is omitted. Use this (or nothing) for every *'normal'* date.
- `BC`: Before Christ; obviously used for dates before Christ.
- `GR`: Gregorian calendar; use this in situations, where the difference betweeen Gregorian and Julian calendar should be emphasized.
- `JU`: Julian calendar; use this in situations, where the difference betweeen Gregorian and Julian calendar should be emphasized.
- `caAD`: circa, but AD; use this for insecure date settings.
- `caBC`: circa, but BC; use this for insecure date settings.
- `ca`: circa; do not use this directly. The language settings for this will be used automatically, if `caAD` is given and `/gtr/calendar print` → P. 176`=all but AD` is set.
- *other*: other flags may be used without error. The flag is just noted.

The date format can be shortened to `(c)yyyy-mm` and `(c)yyyy`. Since the calendar flag is optional, `yyyy-mm-dd`, `yyyy-mm`, and `yyyy` are also possible.

A *date range* is specified in the format

`(c)yyyy-mm-dd/(c)yyyy-mm-dd`

Every partial date may be shortened as described above.
Also, `/(c)yyyy-mm-dd` and `(c)yyyy-mm-dd/` are valid to denote open ranges.

Date Examples

	Formatted Dates		
Specification	`d.M.yyyy`	`month d yyyy`	`dd/mon/yyyy`
`1875-12-07`	7.XII.1875	December 7, 1875	07/Dec/1875
`(JU)1642-12-25`	25.XII.1642$^{jul.}$	December 25, 1642$^{jul.}$	25/Dec/1642$^{jul.}$
`(GR)1599-08`	VIII.1599$^{greg.}$	August, 1599$^{greg.}$	Aug/1599$^{greg.}$
`1475`	1475	1475	1475
`(BC)27-01-16`	16.I.27 BC	January 16, 27 BC	16/Jan/27 BC
`/1690-03`	before III.1690	before March, 1690	before Mar/1690
`1775-07-15/`	after 15.VII.1775	after July 15, 1775	after 15/Jul/1775
`1888-05/1889-06-07`	V.1888 to 7.VI.1889	May, 1888 to June 7, 1889	May/1888 to 07/Jun/1889
`(caAD)1955-02`	ca. II.1955	ca. February, 1955	ca. Feb/1955

\gtrParseDate{⟨*name*⟩}{⟨*date*⟩}

Dates are parsed as part of events automatically, see Section 7.3 on page 155. But with \gtrParseDate, a ⟨*date*⟩ can be parsed directly. The parsed data is stored using the given ⟨*name*⟩ as \gtrDB⟨*name*⟩cal, \gtrDB⟨*name*⟩day, \gtrDB⟨*name*⟩month, \gtrDB⟨*name*⟩year, \gtrDB⟨*name*⟩endcal, \gtrDB⟨*name*⟩endday, \gtrDB⟨*name*⟩endmonth, \gtrDB⟨*name*⟩endyear.

```
\gtrParseDate{xy}{1875-12-07}
The parsed date is
|\gtrDBxycal| (\gtrDBxycal),
|\gtrDBxyday| (\gtrDBxyday),
|\gtrDBxymonth| (\gtrDBxymonth),
|\gtrDBxyyear| (\gtrDBxyyear).

Formatted date: \gtrset{date format=d/M/yyyy}\gtrPrintDate{xy}
```

The parsed date is \gtrDBxycal (AD), \gtrDBxyday (07), \gtrDBxymonth (12), \gtrDBxyyear (1875).
Formatted date: 7/XII/1875

7.5 Formatting the Node Data

While the macros and options of the next sections describe how to format a single piece of data, the `/gtr/database format` integrates a collection of these pieces to format the total content of a node.

`/gtr/database format`=⟨*format*⟩ (style, no default, initially `medium`) U 2017-01-27

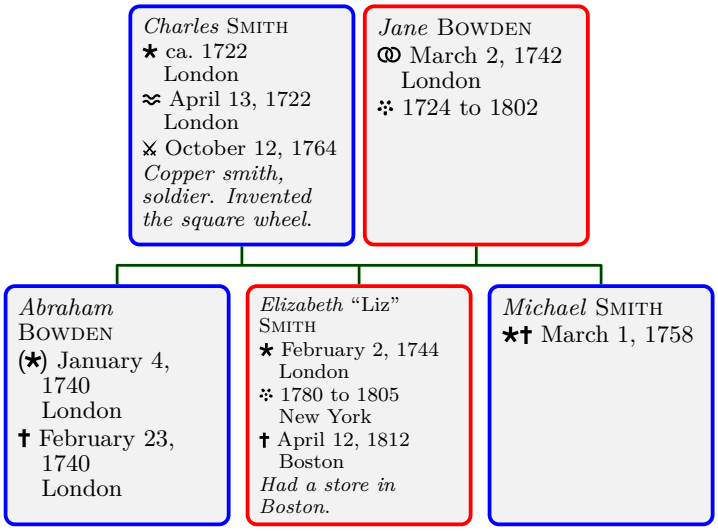
Selects a predefined ⟨*format*⟩ for selecting and arranging data values. The standard ⟨*format*⟩ designs use `gtrprintlist` → P. 181 to list events. New ⟨*format*⟩ designs can be added by `\gtrDeclareDatabaseFormat` → P. 170. The following sections describe how to customize certain parts of the standard ⟨*format*⟩ designs, e.g. `/gtr/date format` → P. 174 for changing the date style.

Feasible (standard) ⟨*format*⟩ values are

- `full`: name, birth, baptism, engagement, marriage, divorce, floruit, death, burial, and informations as profession and comment.
- `full marriage above`: identical to `full`, but engagement, marriage, divorce is put above and separated by a `\tcbline`.
- `full marriage below`: identical to `full`, but engagement, marriage, divorce is put below and separated by a `\tcbline`.
- `full no marriage`: identical to `full`, but without engagement, marriage, and divorce.
- `medium`: name, birth (or baptism), marriage (or engagement or divorce), death (or burial), and informations as profession and comment. Floruit is displayed, if there is no birth, baptism, death, and burial.
- `medium marriage above`: identical to `medium`, but marriage (or engagement or divorce) is put above and separated by a `\tcbline`.
- `medium marriage below`: identical to `medium`, but marriage (or engagement or divorce) is put below and separated by a `\tcbline`.
- `medium no marriage`: identical to `medium`, but without engagement, marriage, and divorce.
- `short`: name, birth (or baptism), marriage (or engagement or divorce), and death (or burial). Floruit is displayed, if there is no birth, baptism, death, and burial.
- `short marriage above`: identical to `short`, but marriage (or engagement or divorce) is put above and separated by a `\tcbline`.
- `short marriage below`: identical to `short`, but marriage (or engagement or divorce) is put below and separated by a `\tcbline`.
- `short no marriage`: identical to `short`, but without engagement, marriage, and divorce.
- `name`: name only.
- `symbol`: symbol only.
- `empty`: nothing.
- `marriage`: only marriage (or engagement or divorce). This format is intended to be used not for nodes, but for edge labels, see `/gtr/label database options` → P. 205.

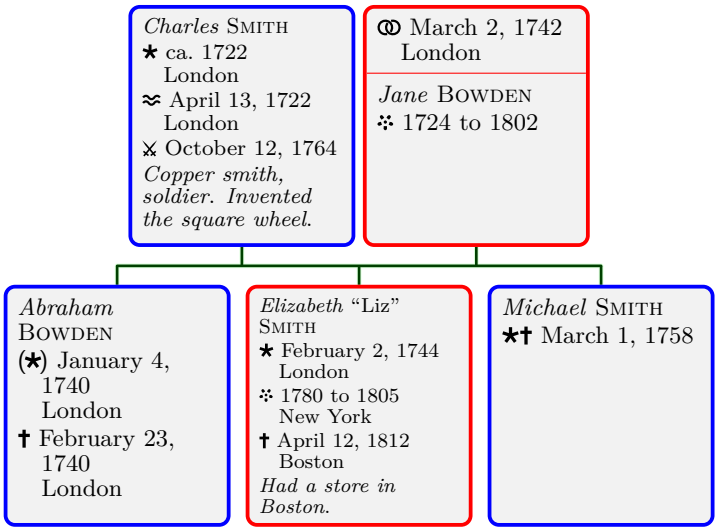
database format=full

```
\begin{genealogypicture}[
  processing=database,database format=full,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



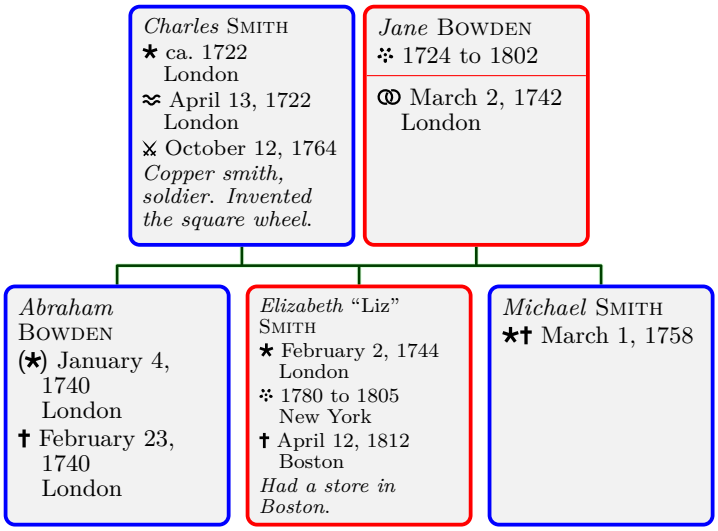
database format=full marriage above

```
\begin{genealogypicture}[
  processing=database,database format=full marriage above,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



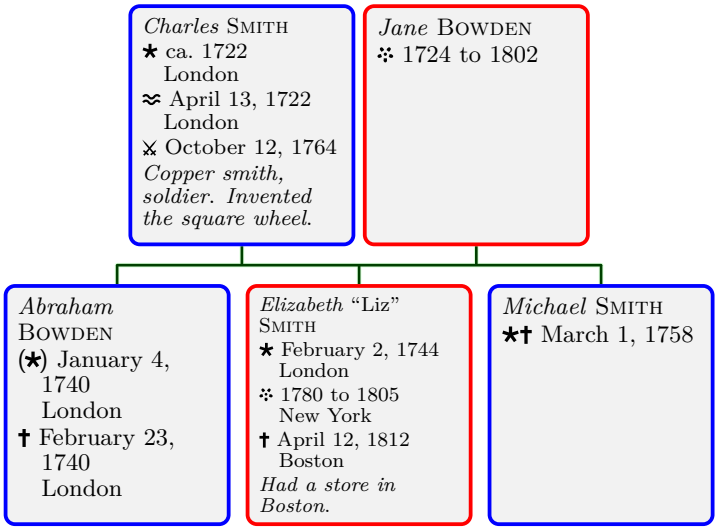
database format=full marriage below

```
\begin{genealogypicture}[
  processing=database,database format=full marriage below,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



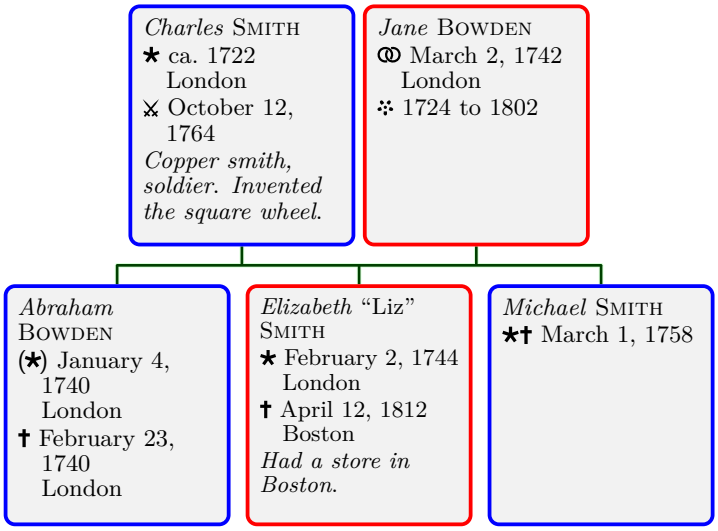
database format=full no marriage

```
\begin{genealogypicture}[
  processing=database,database format=full no marriage,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



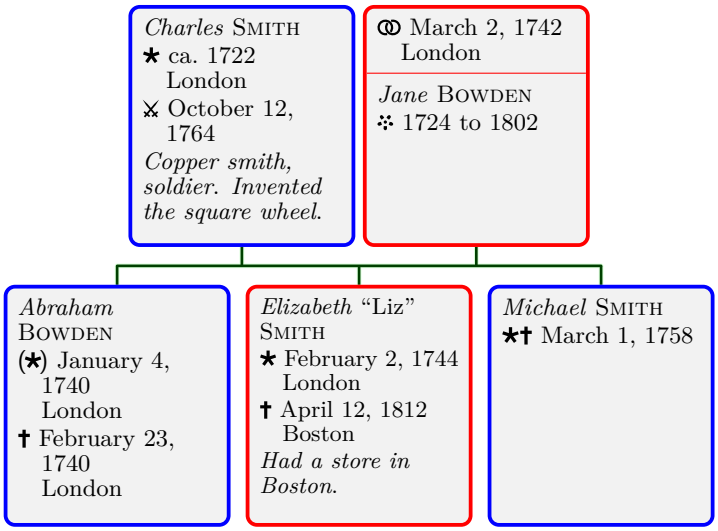
database format=medium

```
\begin{genealogypicture}[
  processing=database,database format=medium,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



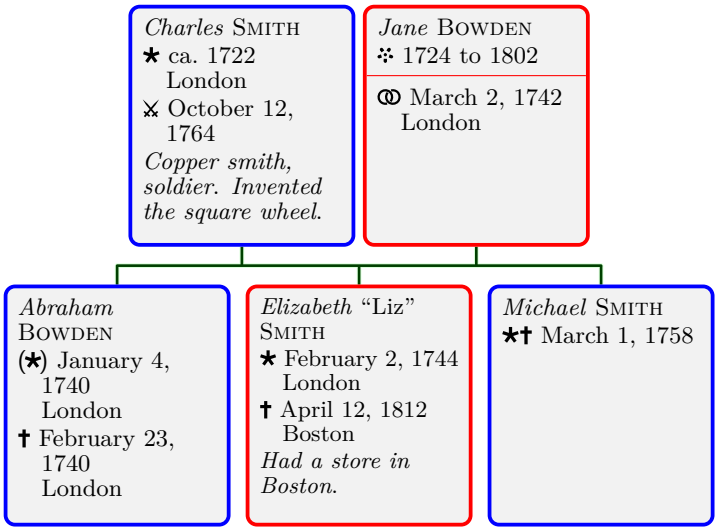
database format=medium marriage above

```
\begin{genealogypicture}[
  processing=database,database format=medium marriage above,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



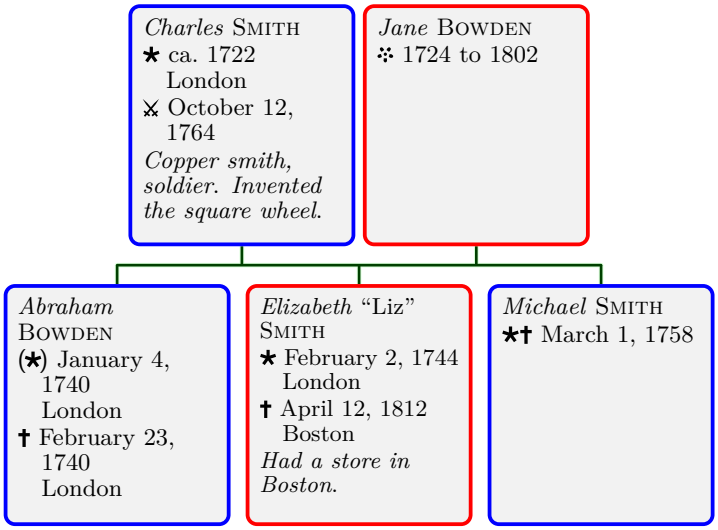
database format=medium marriage below

```
\begin{genealogypicture}[
  processing=database,database format=medium marriage below,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



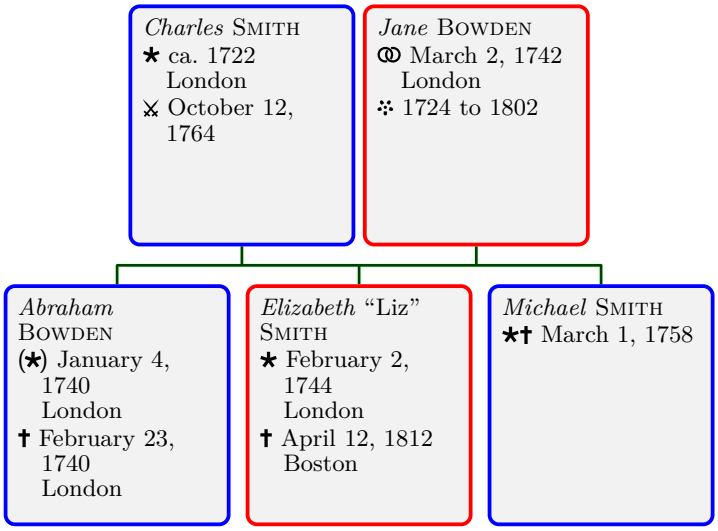
database format=medium no marriage

```
\begin{genealogypicture}[
  processing=database,database format=medium no marriage,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



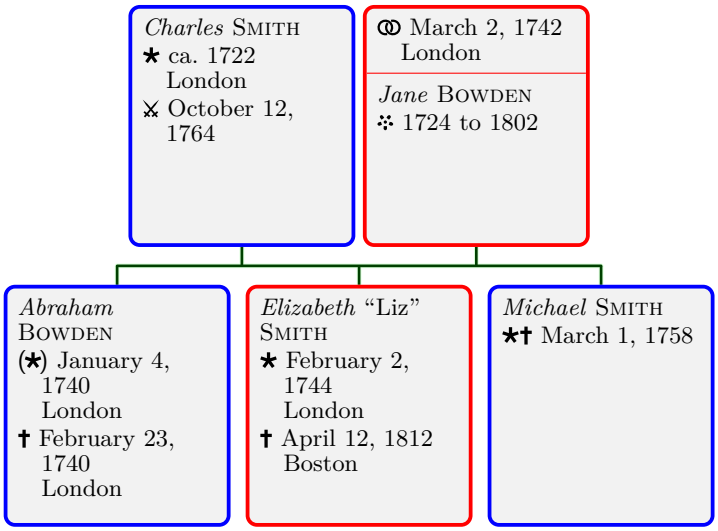
database format=short

```
\begin{genealogypicture}[
  processing=database,database format=short,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



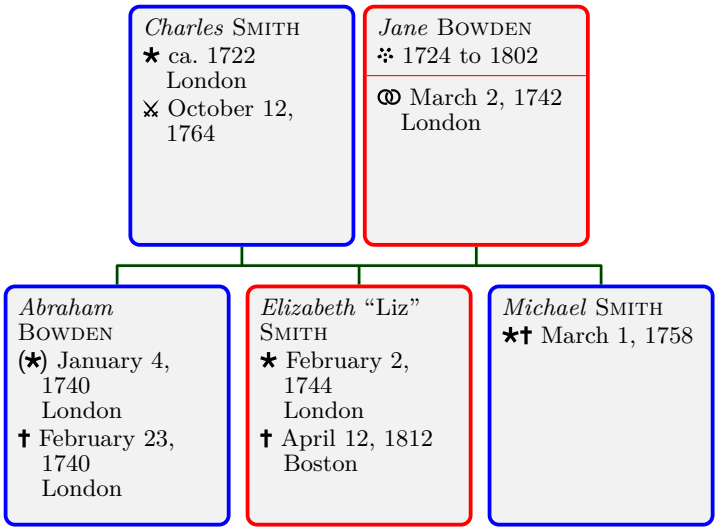
database format=short marriage above

```
\begin{genealogypicture}[
  processing=database,database format=short marriage above,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



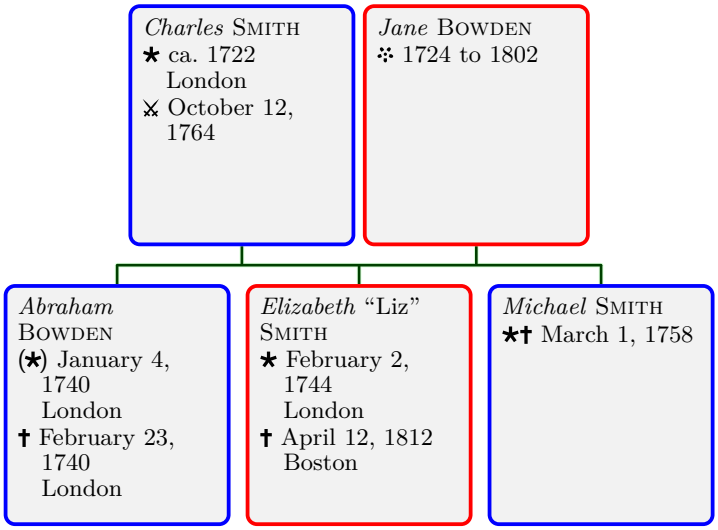
database format=short marriage below

```
\begin{genealogypicture}[
  processing=database,database format=short marriage below,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



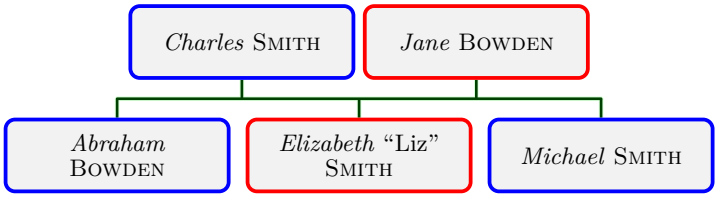
database format=short no marriage

```
\begin{genealogypicture}[
  processing=database,database format=short no marriage,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



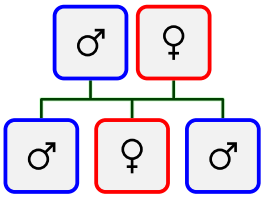
database format=name

```
\begin{genealogypicture}[
  processing=database,database format=name,
  node size=3cm,level size=1cm,
  box={fit basedim=9pt,boxsep=2pt,
    halign=flush center,valign=center,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



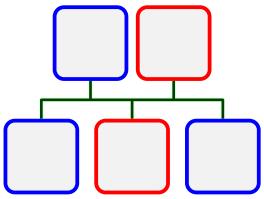
database format=symbol

```
\begin{genealogypicture}[
  processing=database,database format=symbol,
  node size=1cm,level size=1cm,
  box={fit basedim=16pt,boxsep=2pt,
    halign=flush center,valign=center,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



database format=empty

```
\begin{genealogypicture}[
  processing=database,database format=empty,
  node size=1cm,level size=1cm,
  box={fit basedim=16pt,boxsep=2pt,
    halign=flush center,valign=center,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```

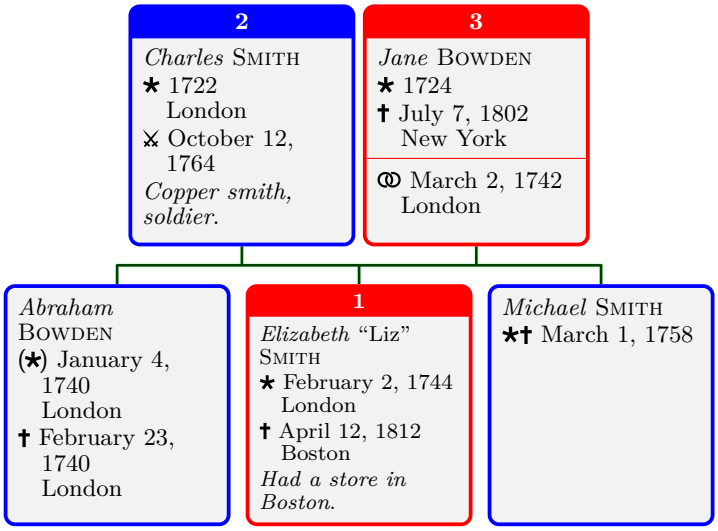


`\gtrDeclareDatabaseFormat{⟨format⟩}{⟨option code⟩}{⟨content code⟩}`

Declares a new ⟨*format*⟩ to be used as value for `/gtr/database format` → P. 162. The ⟨*option code*⟩ is used after the data is read and before the box is set. The ⟨*content code*⟩ is used to fill the box content. It is recommended to start a new ⟨*format*⟩ name with the letter 'x' to avoid collisions with future standard values.

```
\gtrDeclareDatabaseFormat{xkekule}{%
  \ifdefvoid{\gtrDBkekule}{}{\gtrset{box={title=\gtrDBkekule}}}%
}{%
  \gtrPrintName%
  \begin{gtreventlist}%
    \gtrifdatedefined{birth}{\gtrlistseparator\gtrPrintEvent{birth}}{
      \gtrifdatedefined{baptism}{\gtrlistseparator\gtrPrintEvent{baptism}}{}%
    }%
    \gtrifdatedefined{death}{\gtrlistseparator\gtrPrintEvent{death}}{
      \gtrifdatedefined{burial}{\gtrlistseparator\gtrPrintEvent{burial}}{}%
    }%
  \end{gtreventlist}%
  \begin{gtrinfolist}%
    \gtrifprofessiondefined{\gtrlistseparator\gtrPrintProfession}{}%
    \gtrifcommentdefined{\gtrlistseparator\gtrPrintComment}{}%
  \end{gtrinfolist}%
  \gtrifdatedefined{marriage}{\tcbline\begin{gtreventlist}%
    \gtrlistseparator\gtrPrintEvent{marriage}\end{gtreventlist}}{}%
}%
%
\begin{genealogypicture}[
  processing=database,database format=xkekule,
  node size=3cm,level size=3.2cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    center title,fonttitle=\bfseries\normalsize,
    halign=flush left,before upper=\parskip1pt,\gtrDBsex } ]
child{
  g[id=SmitChar1722]{
    male,
    kekule     = 2,
    name       = {\pref{Charles} \surn{Smith}},
    birth      = {1722}{London},
    baptism    = {1722-04-13}{London},
    death+     = {1764-10-12}{}{killed},
    profession = {Copper smith, soldier},
  }
  p[id=BowdJane1724]{
    female,
    kekule   = 3,
    name     = {\pref{Jane} \surn{Bowden}},
    birth-   = {1724},
    marriage = {1742-03-02}{London},
    death    = {1802-07-07}{New York},
  }
  c[id=BowdAbra1740]{
    male,
    name   = {\pref{Abraham} \surn{Bowden}},
    birth+ = {1740-01-04}{London}{out of wedlock},
    death  = {1740-02-23}{London}
  }
  c[id=SmitEliz1744]{
    female,
    kekule  = 1,
    name    = {\pref{Elizabeth} \nick{Liz} \surn{Smith}},
    birth   = {1744-02-02}{London},
    death   = {1812-04-12}{Boston},
```

```
    comment = {Had a store in Boston},
  }
  c[id=SmitMich1758]{
    male,
    name   = {\pref{Michael} \surn{Smith}},
    birth+ = {1758-03-01}{}{died},
  }
}
\end{genealogypicture}
```



7.6 Formatting Names

\gtrPrintName

Used to insert the formatted name. The output format of the name is controlled by /gtr/name and other following options.

```
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}}
%...
\gtrPrintName
```

Elizabeth "Liz" Smith

\pref{⟨*given name*⟩**}**

Marks a preferred ⟨*given name*⟩. May be redefined directly or using /gtr/pref code.

\surn{⟨*surname*⟩**}**

Marks a ⟨*surname*⟩. May be redefined directly or using /gtr/surn code.

\nick{⟨*nickname*⟩**}**

Marks a ⟨*nickname*⟩. May be redefined directly or using /gtr/nick code.

/gtr/pref code={⟨*code*⟩} (no default)

Redefines \pref using ⟨*code*⟩.

```
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}}
\gtrset{pref code={\textcolor{blue}{\bfseries #1}}}
%...
\gtrPrintName
```

Elizabeth "Liz" Smith

/gtr/surn code={⟨*code*⟩} (no default)

Redefines \surn using ⟨*code*⟩.

```
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}}
\gtrset{surn code={\textcolor{blue}{\bfseries #1}}}
%...
\gtrPrintName
```

Elizabeth "Liz" **Smith**

/gtr/nick code={⟨*code*⟩} (no default)

Redefines \nick using ⟨*code*⟩.

```
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}}
\gtrset{nick code={\textcolor{blue}{\bfseries #1}}}
%...
\gtrPrintName
```

Elizabeth **Liz** Smith

/gtr/name=full|short (no default, initially `full`)

Controls, if \gtrPrintName should preferably use the `full` version (/gtr/database/name → P. 155) or the `short` version (/gtr/database/shortname → P. 155) of a name. If the preferred version is not available, the other version is used.

/gtr/name font={⟨*code*⟩} (no default)

Sets the font (and/or color) for \gtrPrintName → P. 172.

```
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}}
\gtrset{name font=\fontfamily{ptm}\selectfont\color{green!50!black}}
%...
\gtrPrintName
```

Elizabeth “Liz” Smith

/gtr/empty name text={⟨*text*⟩} (no default, initially ??)

Sets the text to be print by \gtrPrintName → P. 172, if /gtr/database/name → P. 155 and /gtr/database/shortname → P. 155 were not set.

```
\gtrPrintName

\gtrset{empty name text={N.N.}}
\gtrPrintName
```

??
N.N.

/gtr/name code={⟨*code*⟩} (no default)

Defines ⟨*code*⟩ to be executed by \gtrPrintName → P. 172. Use this, if /gtr/name → P. 172 and /gtr/name font are not flexible enough.

```
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}},female}
\gtrset{name code={\gtrPrintSex~\gtrDBname}}
%...
\gtrPrintName
```

♀ *Elizabeth* “Liz” Smith

7.7 Formatting Dates

`\gtrPrintDate{`⟨*name*⟩`}`

Used to insert a formatted date referred by ⟨*name*⟩. This ⟨*name*⟩ is an event name like *birth*, see Section 7.3 on page 155, or any other name used by `\gtrParseDate` → P. 161. The output format of the date is controlled by `/gtr/date format` and other following options.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
%...
The birth was \gtrPrintDate{birth}.
The death was \gtrPrintDate{death}.
```

The birth was February 9, 1354. The death was ??.

`\gtrifdatedefined{`⟨*name*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if a date with the given ⟨*name*⟩ is defined, and to ⟨*false*⟩ otherwise.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
%...
\gtrifdatedefined{birth}{The birth was \gtrPrintDate{birth}.}{}
\gtrifdatedefined{death}{The death was \gtrPrintDate{death}.}{}
```

The birth was February 9, 1354.

`/gtr/date format={`⟨*format*⟩`}` (no default, initially `typical`) U 2017-01-26

This option controls how day, month, and year of a date are formatted when using `\gtrPrintDate`. This setting is *not* `/gtr/language` → P. 225 dependent, but month names are. One exception to this rule is

- `typical` *A typical format for the language, here:* February 9, 1354

Further feasible ⟨*format*⟩ values are

- `dd.mm.yyyy` 09.02.1354
- `d.m.yyyy` 9.2.1354
- `d.M.yyyy` 9.II.1354
- `d.month yyyy` 9. February 1354
- `dd.mon.yyyy` 09. Feb. 1354
- `d.mon.yyyy` 9. Feb. 1354
- `dd mon.yyyy` 09 Feb. 1354
- `d mon.yyyy` 9 Feb. 1354
- `dd/mm/yyyy` 09/02/1354
- `d/m/yyyy` 9/2/1354
- `d/M/yyyy` 9/II/1354
- `dd/month/yyyy` 09/February/1354
- `d/month/yyyy` 9/February/1354
- `dd/mon/yyyy` 09/Feb/1354
- `d/mon/yyyy` 9/Feb/1354
- `dd/mm yyyy` 09/02 1354
- `dd mm yyyy` 09 02 1354
- `d M yyyy` 9 II 1354
- `d month yyyy` 9 February 1354
- `dd mon yyyy` 09 Feb 1354
- `d mon yyyy` 9 Feb 1354
- `dd-mm-yyyy` 09-02-1354
- `d-m-yyyy` 9-2-1354

- d-M-yyyy 9-II-1354
- dd-month-yyyy 09-February-1354
- d-month-yyyy 9-February-1354
- dd-mon-yyyy 09-Feb-1354
- d-mon-yyyy 9-Feb-1354
- ddmonyyyy 09Feb1354
- yyyy.mm.dd 1354.02.09
- yyyy.m.d 1354.2.9
- yyyy.m.d. 1354. 2. 9.
- yyyy.M.d. 1354. II. 9.
- yyyy.month d. 1354. February 9.
- yyyy.mon.d. 1354. Feb. 9.
- yyyy/mm/dd 1354/02/09
- yyyy/m/d 1354/2/9
- yyyy mm dd 1354 02 09
- yyyy month d 1354 February 9
- yyyy mon dd 1354 Feb 09
- yyyy-mm-dd 1354-02-09
- yyyy-mon-dd 1354-Feb-09
- yyyy-mon-d 1354-Feb-9
- yyyymondd 1354Feb09
- yyyymmdd 13540209
- mm.dd.yyyy 02.09.1354
- m.d.yyyy 2.9.1354
- mm/dd/yyyy 02/09/1354
- m/d/yyyy 2/9/1354
- mm-dd-yyyy 02-09-1354
- m-d-yyyy 2-9-1354
- month d yyyy February 9, 1354
- mon.d yyyy Feb. 9, 1354
- mon d yyyy Feb 9, 1354
- yyyy 1354

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
\gtrset{date format=month d yyyy}
%...
The birth was \gtrPrintDate{birth}.
```

The birth was February 9, 1354.

/gtr/date code={⟨*code*⟩} (no default)

Defines ⟨*code*⟩ to be executed by \gtrPrintDate→ P. 174. Use this, if /gtr/date format→ P. 174 is not flexible enough.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
\gtrset{date code={%
  \ifcsdef{#1month}{%
    \ifcsdef{#1day}{\csuse{#1day}}{}%
    (\csuse{#1month})%
  }{}%
  \csuse{#1year}%
}}
%...
The birth was \gtrPrintDate{birth}.
```

The birth was 09(02)1354.

`/gtr/calendar text for=`⟨*calendar*⟩ `is {`⟨*prefix*⟩`}{`⟨*postfix*⟩`}` (no default)

Defines a ⟨*prefix*⟩ and a ⟨*postfix*⟩ text for a ⟨*calendar*⟩. This setting is `/gtr/language` → P. 225 dependent for known calendars. This option also allows to set up new ⟨*calendar*⟩ entries.

```
\gtrset{database/.cd,birth={(AUC)2107-02-09}{Rome}}
\gtrset{calendar text for=AUC is {}{ a.u.c.}}
%...
The birth was \gtrPrintDate{birth}.
```

The birth was February 9, 2107 a.u.c..

`/gtr/calendar print={`⟨*option*⟩`}` (no default, initially `all but AD`)

Defines, if the calendar setting is used for formatting. Feasible ⟨*option*⟩ values are

- `all`: all calendar settings, including AD.
- `none`: no calendar settings.
- `all but AD`: all calendar settings, but excluding AD.

```
\gtrset{database/.cd,birth={(BC)63-09-23}{Rome},death={(AD)14-08-19}{Nola}}
%...
Augustus was born \gtrPrintDate{birth} and died \gtrPrintDate{death}.
\par\gtrset{calendar print=none}
Augustus was born \gtrPrintDate{birth} and died \gtrPrintDate{death}.
\par\gtrset{calendar print=all}
Augustus was born \gtrPrintDate{birth} and died \gtrPrintDate{death}.
```

Augustus was born September 23, 63 BC and died August 19, 14.
Augustus was born September 23, 63 and died August 19, 14.
Augustus was born September 23, 63 BC and died AD August 19, 14.

`/gtr/date range full={`⟨*pre*⟩`}{`⟨*mid*⟩`}{`⟨*app*⟩`}` (no default, initially `{}{ to }{}`) U 2015-06-16

If the date is a *date range* with a start date and an end date, the ⟨*pre*⟩, ⟨*mid*⟩, and ⟨*app*⟩ texts are placed appropriately. This setting is `/gtr/language` → P. 225 dependent.

```
\gtrset{database/.cd,birth={1354-02-09/1355-07-20}{Rome}}
\gtrset{date range full={between }{ and }{}}
%...
The birth was \gtrPrintDate{birth}.
```

The birth was between February 9, 1354 and July 20, 1355.

`/gtr/date range before={`⟨*pre*⟩`}{`⟨*app*⟩`}` (no default, initially `{before }{}`)

If the date is a *date range* an end date, but without start date, the ⟨*pre*⟩ and ⟨*app*⟩ texts are placed around the end date. This setting is `/gtr/language` → P. 225 dependent.

```
\gtrset{database/.cd,birth={/1355-07-20}{Rome}}
\gtrset{date range before={\textless\,}{}{}}
%...
The birth was \gtrPrintDate{birth}.
```

The birth was < July 20, 1355.

`/gtr/date range after={⟨pre⟩}{⟨app⟩}` (no default, initially `{after }{}`)

If the date is a *date range* a start date, but without end date, the ⟨*pre*⟩ and ⟨*app*⟩ texts are placed around the start date. This setting is `/gtr/language` → P. 225 dependent.

```
\gtrset{database/.cd,birth={1354-02-09/}{Rome}}
\gtrset{date range after={\textgreater\,}{}}
%...
The birth was \gtrPrintDate{birth}.
```

The birth was > February 9, 1354.

U 2015-06-16 `/gtr/date range separator={⟨text⟩}` (style, default `--`, initially unset)

Sets the same separator *text* for `/gtr/date range full` → P. 176, `/gtr/date range before` → P. 176, `/gtr/date range after`. Use this for shortened range printing.

```
\gtrset{database/.cd,birth={1354-02-09/}{Rome}}
\gtrset{date range separator={--}}
%...
The birth was \gtrPrintDate{birth}.
```

The birth was February 9, 1354–.

7.8 Formatting Places

`\gtrPrintPlace{`⟨*name*⟩`}`

Used to insert a formatted place referred by ⟨*name*⟩. This ⟨*name*⟩ is an event name like *birth*, see Section 7.3 on page 155. The output format of the place is controlled by `/gtr/place text`.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
%...
The birth was \gtrPrintDate{birth} \gtrPrintPlace{birth}.
```

The birth was February 9, 1354 in Rome.

`\gtrifplacedefined{`⟨*name*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if a place with the given ⟨*name*⟩ is defined, and to ⟨*false*⟩ otherwise.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
%...
The birth was \gtrPrintDate{birth}%
  \gtrifplacedefined{birth}{ \gtrPrintPlace{birth}}{}.
```

The birth was February 9, 1354 in Rome.

`/gtr/place text={`⟨*pre*⟩`}{`⟨*app*⟩`}` (no default, initially `{in }{}`)

The ⟨*pre*⟩ and ⟨*app*⟩ texts are placed around the place text. This setting is `/gtr/language` → P. 225 dependent.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
\gtrset{place text={(}{)}}
%...
The birth was \gtrPrintDate{birth}%
  \gtrifplacedefined{birth}{ \gtrPrintPlace{birth}}{}.
```

The birth was February 9, 1354 (Rome).

7.9 Formatting Events

`\gtrPrintEvent{`⟨*name*⟩`}`

Used to insert a formatted event referred by ⟨*name*⟩. This ⟨*name*⟩ is an event name like *birth*, see Section 7.3 on page 155. The output format of the event is controlled by `/gtr/event text` →P. 180, `\gtrPrintEventPrefix`, `\gtrPrintDate` →P. 174, and `\gtrPrintPlace` →P. 178.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
%...
\gtrPrintEvent{birth}
```

★ February 9, 1354 in Rome

`\gtrifeventdefined{`⟨*name*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if an event with the given ⟨*name*⟩ is defined, and to ⟨*false*⟩ otherwise. This is an alias for `\gtrifdatedefined` →P. 174.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
%...
\gtrifeventdefined{birth}{\gtrPrintEvent{birth}}{}
```

★ February 9, 1354 in Rome

`\gtrPrintEventPrefix{`⟨*name*⟩`}`

Used to insert an event prefix like a symbol. The prefix depends upon the ⟨*name*⟩ of the event and upon an optional modifier. The output format of the prefix is controlled by the following options with the `/gtr/event prefix` path.

```
Birth: \gtrPrintEventPrefix{birth}
\par\gtrset{/gtr/event prefix/birth=(b)}
Birth: \gtrPrintEventPrefix{birth}
```

Birth: ★
Birth: (b)

`/gtr/event prefix/birth=`⟨*text*⟩ (no default, initially `\gtrsymBorn` →P. 219)
Prefix ⟨*text*⟩ for a normal birth.

`/gtr/event prefix/birth/out of wedlock=`⟨*text*⟩ (no default, initially `\gtrsymBornoutofwedlock` →P. 219)
Prefix ⟨*text*⟩ for a birth out of wedlock.

`/gtr/event prefix/birth/stillborn=`⟨*text*⟩ (no default, initially `\gtrsymStillborn` →P. 219)
Prefix ⟨*text*⟩ for a birth of a stillborn child.

`/gtr/event prefix/birth/died=`⟨*text*⟩ (no default, initially `\gtrsymDiedonbirthday` →P. 219)
Prefix ⟨*text*⟩ for a birth if a child who died on birthday.

`/gtr/event prefix/baptism=`⟨*text*⟩ (no default, initially `\gtrsymBaptized` →P. 219)
Prefix ⟨*text*⟩ for a baptism.

`/gtr/event prefix/engagement=`⟨*text*⟩ (no default, initially `\gtrsymEngaged` →P. 220)
Prefix ⟨*text*⟩ for a engagement.

`/gtr/event prefix/marriage=`⟨*text*⟩ (no default, initially `\gtrsymMarried` →P. 220)
Prefix ⟨*text*⟩ for a normal marriage.

`/gtr/event prefix/marriage/other`=⟨*text*⟩ (no default, initially `\gtrsymPartnership` → P. 220)
Prefix ⟨*text*⟩ for another partnership.

`/gtr/event prefix/divorce`=⟨*text*⟩ (no default, initially `\gtrsymDivorced` → P. 220)
Prefix ⟨*text*⟩ for a divorce.

`/gtr/event prefix/floruit`=⟨*text*⟩ (no default, initially `\gtrsymFloruit` → P. 221) N 2017-07-18
Prefix ⟨*text*⟩ for a floruit event.

`/gtr/event prefix/death`=⟨*text*⟩ (no default, initially `\gtrsymDied` → P. 220)
Prefix ⟨*text*⟩ for a normal death.

`/gtr/event prefix/death/killed`=⟨*text*⟩ (no default, initially `\gtrsymKilled` → P. 220)
Prefix ⟨*text*⟩ for a death in war.

`/gtr/event prefix/burial`=⟨*text*⟩ (no default, initially `\gtrsymBuried` → P. 221)
Prefix ⟨*text*⟩ for a normal burial.

`/gtr/event prefix/burial/cremated`=⟨*text*⟩ (no default, initially `\gtrsymFuneralurn` → P. 221)
Prefix ⟨*text*⟩ for a cremation.

`/gtr/event format`={⟨*format*⟩} (no default, initially `d.M.yyyy`)
This option controls how events are formatted when using `\gtrPrintEvent` → P. 179.
Feasible ⟨*format*⟩ values are

- `prefix date place` ★ February 9, 1354 in Rome
- `prefix date` ★ February 9, 1354
- `date` February 9, 1354

`/gtr/event text`={⟨*pre*⟩}{⟨*sep date*⟩}{⟨*sep place*⟩}{⟨*app*⟩} (no default, initially `{}{~}{ }{}`)
The four text pieces are placed inside `\gtrPrintEvent` → P. 179 as follows:
⟨*pre*⟩prefix⟨*sep date*⟩date⟨*sep place*⟩place⟨*app*⟩
This setting is *not* `/gtr/language` → P. 225 dependent.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
\gtrset{event text={[}{: }{ }{]}}
%...
\gtrPrintEvent{birth}
```

[★: February 9, 1354 in Rome]

`/gtr/event code`={⟨*code*⟩} (no default)
Defines ⟨*code*⟩ to be executed by `\gtrPrintEvent` → P. 179. Use this, if `/gtr/event format` and `/gtr/event text` are not flexible enough.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
\gtrset{event code={%
  \gtrPrintEventPrefix{#1}
  \grtrifplacedefined{#1}{(\gtrPrintPlace{#1}) }{}%
  \gtrPrintDate{#1}%
}}
%...
\gtrPrintEvent{birth}
```

★ (in Rome) February 9, 1354

7.10 Formatting Lists of Events

\begin{gtrprintlist}{⟨*first*⟩}{⟨*middle*⟩}{⟨*last*⟩}{⟨*empty*⟩}
⟨*environment content*⟩
\end{gtrprintlist}

This environment is intended for automatically generated content. Inside this environment, a macro \gtrlistseparator is defined.

- \gtrlistseparator expands to ⟨*first*⟩, when it is called the first time.
- \gtrlistseparator expands to ⟨*middle*⟩, when it is called later.
- ⟨*last*⟩ is used at the end of the environment, if \gtrlistseparator was called at least once.
- ⟨*empty*⟩ is used at the end of the environment, if \gtrlistseparator was never called.

```
\begin{gtrprintlist}{\unskip}%
    {\unskip,\ }{\unskip.}{\unskip}
  \gtrlistseparator One
  \gtrlistseparator Two
  \gtrlistseparator Three
  \gtrlistseparator Four
\end{gtrprintlist}
```

One, Two, Three, Four.

```
\begin{gtrprintlist}%
    {\begin{itemize}\item}{\item}%
    {\end{itemize}}{\unskip}
  \gtrlistseparator One
  \gtrlistseparator Two
  \gtrlistseparator Three
  \gtrlistseparator Four
\end{gtrprintlist}
```

- One
- Two
- Three
- Four

\begin{gtreventlist}
⟨*environment content*⟩
\end{gtreventlist}

This is a gtrprintlist environment with parameters specified by /gtr/list separators → P. 182. This environment is used internally by most /gtr/database format → P. 162 settings to print event lists.

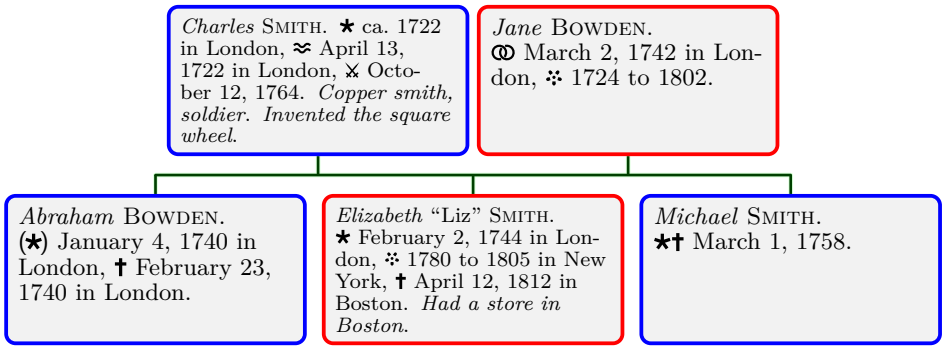
```
\gtrset{list separators=
    {\unskip}{\unskip,\ }%
    {\unskip.}{\unskip}}

\begin{gtreventlist}%
  \gtrlistseparator One
  \gtrlistseparator Two
  \gtrlistseparator Three
  \gtrlistseparator Four
\end{gtreventlist}
```

One, Two, Three, Four.

`/gtr/list separators={⟨first⟩}{⟨middle⟩}{⟨last⟩}{⟨empty⟩}` (no default, initially `{\par}{\par}{\par}{\par}`)

Defines `gtreventlist` → P. 181 as a `gtrprintlist` → P. 181 with the given parameters. This is used to list events.

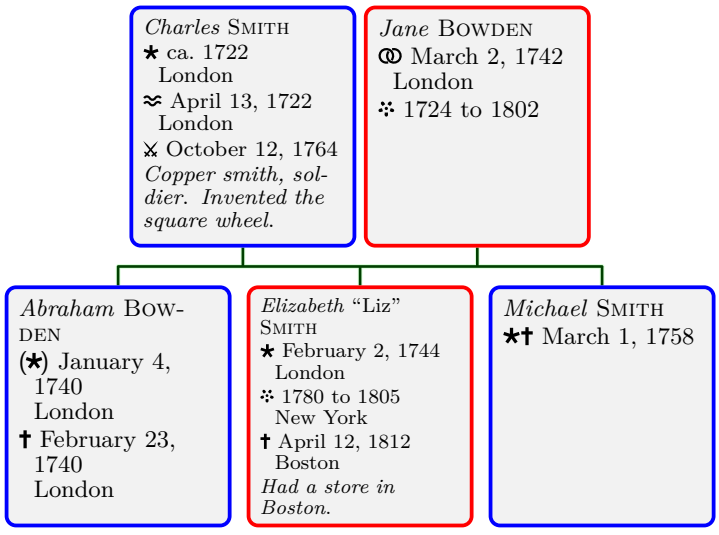
```
\begin{genealogypicture}[
  processing=database,database format=full,
  node size=4cm,level size=2cm,
  name code={\gtrDBname.},
  list separators={ }{, }{.}{},
  info separators={ }{. }{.}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



`/gtr/list separators hang=⟨length⟩` (style, default `\tcbfitdim`)

Defines `gtreventlist` → P. 181 as a `gtrprintlist` → P. 181 where the items hang with ⟨*length*⟩ after the first line.

```
\begin{genealogypicture}[
  processing=database,database format=full,
  node size=3cm,level size=3.2cm,
  list separators hang=2mm,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



7.11 Formatting Comments

\gtrPrintComment

Used to insert the formatted comment. May be redefined directly or using /gtr/comment code.

```
\gtrset{database/.cd,comment={Had a store in Boston}}
%...
\gtrPrintComment
```

Had a store in Boston

\gtrifcommentdefined{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩, if a comment is defined, and to ⟨*false*⟩ otherwise.

```
\gtrset{database/.cd,comment={Had a store in Boston}}
%...
\gtrifcommentdefined{\gtrPrintComment}{}
```

Had a store in Boston

/gtr/comment code={⟨*code*⟩} (no default, initially {\itshape\gtrDBcomment})

Redefines \gtrPrintComment using ⟨*code*⟩.

```
\gtrset{database/.cd,comment={Had a store in Boston}}
\gtrset{comment code={(\gtrDBcomment)}}
%...
\gtrPrintComment
```

(Had a store in Boston)

7.12 Formatting Professions

`\gtrPrintProfession` N 2017-01-27

Used to insert the formatted profession. May be redefined directly or using `/gtr/profession code`.

```
\gtrset{database/.cd,profession={Copper smith, soldier}}
%...
\gtrPrintProfession
```

Copper smith, soldier

`\gtrifprofessiondefined{`⟨*true*⟩`}{`⟨*false*⟩`}` N 2017-01-27

Expands to ⟨*true*⟩, if a profession is defined, and to ⟨*false*⟩ otherwise.

```
\gtrset{database/.cd,profession={Copper smith, soldier}}
%...
\gtrifprofessiondefined{\gtrPrintProfession}{}
```

Copper smith, soldier

`/gtr/profession code=`{⟨*code*⟩} (no default, initially `{\itshape\gtrDBprofession}`) N 2017-01-27

Redefines `\gtrPrintProfession` using ⟨*code*⟩.

```
\gtrset{database/.cd,profession={Copper smith, soldier}}
\gtrset{profession code={(\gtrDBprofession)}}
%...
\gtrPrintProfession
```

(Copper smith, soldier)

7.13 Formatting Lists of Information

N 2017-01-27

```
\begin{gtrinfolist}
  ⟨environment content⟩
\end{gtrinfolist}
```

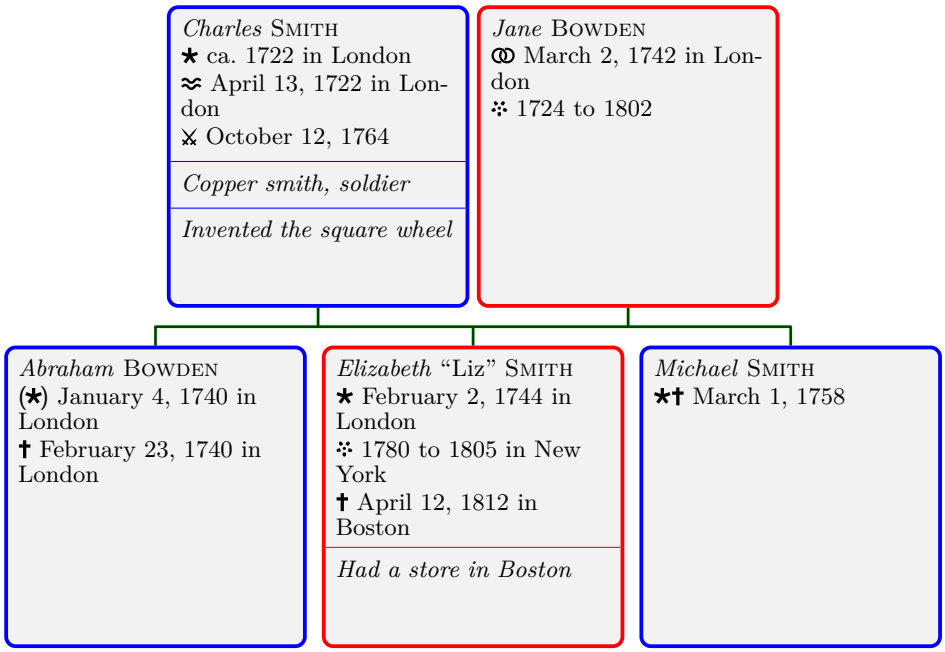
This is a gtrprintlist → P. 181 environment with parameters specified by /gtr/info separators. This environment is used internally by most /gtr/database format → P. 162 settings to print information lists consisting of /gtr/database/profession → P. 156 and /gtr/database/comment → P. 155 entries.

N 2017-01-27

/gtr/info separators={⟨*first*⟩}{⟨*middle*⟩}{⟨*last*⟩}{⟨*empty*⟩} (no default, initially {\par}{. }{.}{})

Defines gtrinfolist as a gtrprintlist → P. 181 with the given parameters. This is used to list informations.

```
\begin{genealogypicture}[
  processing=database,database format=full,
  node size=4cm,level size=4cm,
  info separators={\tcbline}{\tcbline}{}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=left,before upper=\parskip1pt,\gtrDBsex } ]
  input{example.database.graph}
\end{genealogypicture}
```



7.14 Formatting Sex

`\gtrPrintSex`

Used to insert a symbolic sign for the sex.

```
\gtrset{database/.cd,sex=female}
%...
\gtrPrintSex
```

♀

`\gtriffemale{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if `\gtrDBsex` holds `female`, and to ⟨*false*⟩ otherwise.

`\gtrifmale{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if `\gtrDBsex` holds `male`, and to ⟨*false*⟩ otherwise.

Note that the content of the data key `/gtr/database/sex` → P. 155 is accessible by `\gtrDBsex`. Since `/tcb/female` → P. 99, `/tcb/male` → P. 99, `/tcb/neuter` → P. 99, and `/gtr/female` → P. 99, `/gtr/male` → P. 99, `/gtr/neuter` → P. 99 are defined, `\gtrDBsex` can be used directly as a formatting option, see Section 7.2 on page 153 and the examples in Section 7.5 on page 162.

7.15 Formatting Images

`\gtrifimagedefined{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩, if an image is defined, and to ⟨*false*⟩ otherwise.

```
\gtrset{database/.cd,image=Carl_Friedrich_Gauss.jpg}
%...
\gtrifimagedefined{\includegraphics[width=3cm]{\gtrDBimage}}{no image}
```

`/tcb/if image defined=`{⟨*true*⟩}{⟨*false*⟩} (style, no value)

Sets ⟨*true*⟩ `tcolorbox` options, if an image is defined, and sets ⟨*false*⟩ `tcolorbox` options otherwise. This key is intended to be used inside `/gtr/box` → P. 96 constructs.

```
\gtrset{
  options for node={mynode}{
    box={if image defined={watermark graphics=\gtrDBimage}{}}
  }
}
%...
```

`/tcb/image prefix=`{⟨*text*⟩} (no default, initially empty)

Add a prefix ⟨*text*⟩ to every image file name.

```
\gtrset{image prefix=picturedir/}
\gtrset{database/.cd,image=mytest.jpg}
%...
Picture file: \texttt{\gtrDBimage}
```

Picture file: `picturedir/mytest.jpg`

8

Edges

Edges are drawn between all nodes of a *family*. For the auto-layout algorithm, the edges are opaque. Space is reserved for the edges according to the various distance settings for nodes, but the edge dimensions themselves are not considered during layout. The following settings and options influence the visual appearance of the edges.

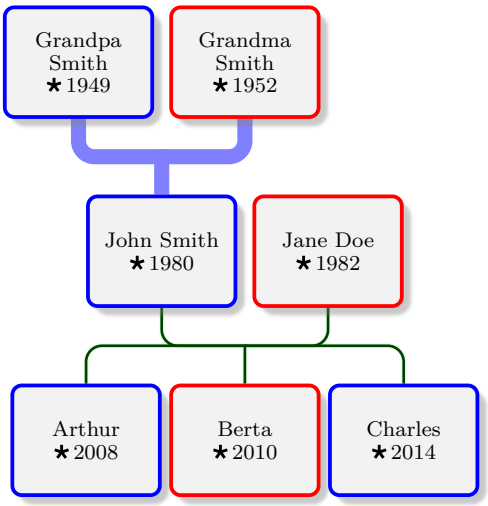
Edges are drawn in two steps: a `/gtr/edge/background` → P. 200 followed by a `/gtr/edge/foreground` → P. 199. After all edges are drawn, the nodes are drawn (possibly over the edges).

8.1 Edge Settings

`/gtr/edges={`⟨*edge options*⟩`}` (style, no default, initially `perpendicular`)

Defines the ⟨*edge options*⟩ for drawing the edges between the nodes of a family. Normally, an edge is drawn with a `/gtr/edge/background` → P. 200 graph and a `/gtr/edge/foreground` → P. 199 graph to allow visual separation of superposed edges. This setting may be given globally, as option of `\genealogytree` → P. 55 or locally wrapped by `/gtr/family` → P. 103. Also see Section 5.1.2 on page 77.

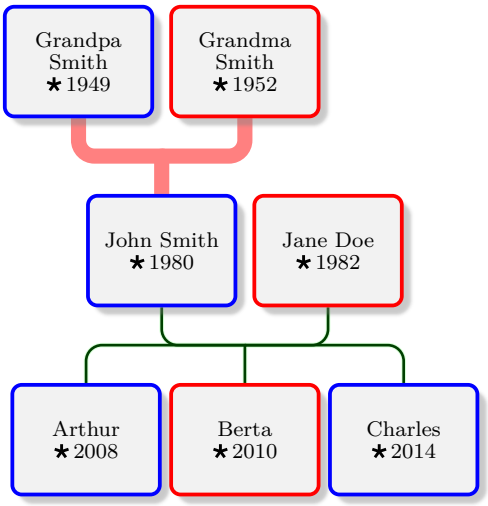
```
\begin{tikzpicture}
\genealogytree[template=signpost,edges=rounded]
{
  parent[id=SmithDoe]{
    g[id=Arth2008,male]{Arthur\\\gtrsymBorn\,2008}
    c[id=Bert2010,female]{Berta\\\gtrsymBorn\,2010}
    c[id=Char2014,male]{Charles\\\gtrsymBorn\,2014}
    parent[id=Smith,family={edges={foreground={blue!50,line width=2mm}}}]{
      g[id=John1980,male]{John Smith\\\gtrsymBorn\,1980}
      p[id=GpSm1949,male]{Grandpa Smith\\\gtrsymBorn\,1949}
      p[id=GmSm1952,female]{Grandma Smith\\\gtrsymBorn\,1952}
    }
    p[id=Jane1982,female]{Jane Doe\\\gtrsymBorn\,1982}
  }
}
\end{tikzpicture}
```



`/gtr/family edges={⟨edge options⟩}` (style, no default)

This is a shortcut for embedding `/gtr/edges` → P. 190 into `/gtr/family` → P. 103.

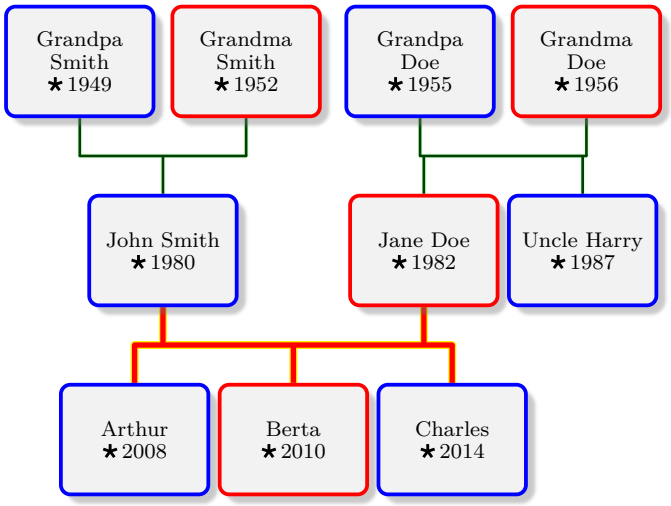
```
\begin{tikzpicture}
\genealogytree[template=signpost,edges={rounded}]
{
  parent[id=SmithDoe]{
    g[id=Arth2008,male]{Arthur\\\gtrsymBorn\,2008}
    c[id=Bert2010,female]{Berta\\\gtrsymBorn\,2010}
    c[id=Char2014,male]{Charles\\\gtrsymBorn\,2014}
    parent[id=Smith,family edges={foreground={red!50,line width=2mm}}]{
      g[id=John1980,male]{John Smith\\\gtrsymBorn\,1980}
      p[id=GpSm1949,male]{Grandpa Smith\\\gtrsymBorn\,1949}
      p[id=GmSm1952,female]{Grandma Smith\\\gtrsymBorn\,1952}
    }
    p[id=Jane1982,female]{Jane Doe\\\gtrsymBorn\,1982}
  }
}
\end{tikzpicture}
```



`/gtr/edges for family={⟨family⟩}{⟨edge options⟩}` (style, no default)

This is a shortcut for embedding `/gtr/edges` → P. 190 into `/gtr/options for family` → P. 102.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  edges for family={SmithDoe}{
    foreground={red,line width=2pt},background={yellow,line width=3pt}},
]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/subtree edges={⟨edge options⟩}` (style, no default)

This is a shortcut for embedding `/gtr/edges` → P. 190 into `/gtr/subtree` → P. 106.

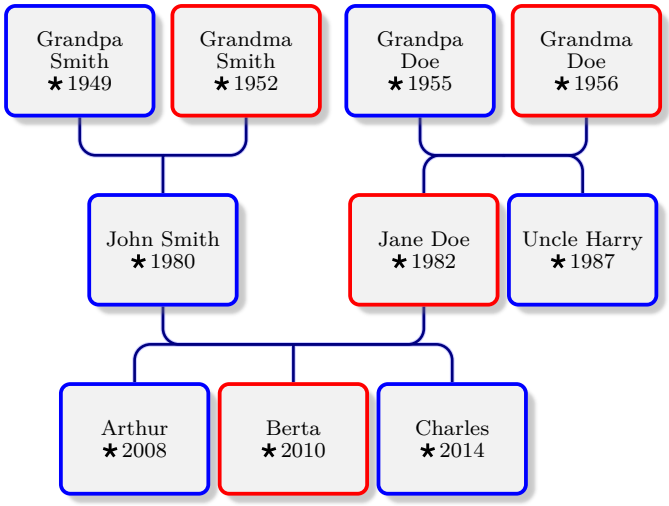
`/gtr/edges for subtree={⟨family⟩}{⟨edge options⟩}` (style, no default)

This is a shortcut for embedding `/gtr/edges` → P. 190 into `/gtr/options for subtree` → P. 105.

`\gtredgeset{⟨options⟩}`

Sets ⟨*options*⟩ for the `/gtr/edge` key subtree. Mainly, this macro is intended to easily set up styles for edges.

```
\begin{tikzpicture}
\gtredgeset{myedges/.style={rounded=6pt,
  foreground={blue!50!black},background={blue!20!white}}}
%...
\genealogytree[template=signpost,edges=myedges]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/tikz/genealogytree edges scope` (style, initially empty)

This style is used to scope the drawing of the edges. It may be redefined e.g. to draw edges on a certain layer.

```
% draw edges on the background layer
\usetikzlibrary{backgrounds}
\tikzset{genealogytree edges scope/.style={on background layer}}
```

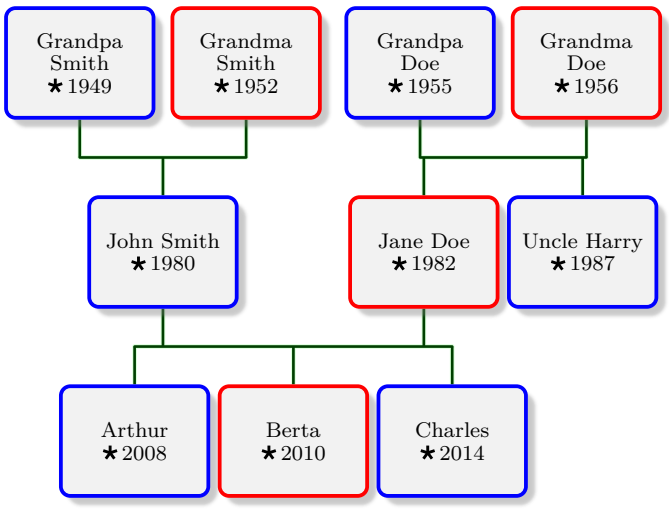
Note that edges are drawn before nodes. Typically, the setting to draw on the background layer is not necessary. If two genealogy trees are merged, this additional setting may be useful.

8.2 Edge Types

`/gtr/edge/perpendicular` (no value, initially set)

The edges are drawn in a perpendicular style.

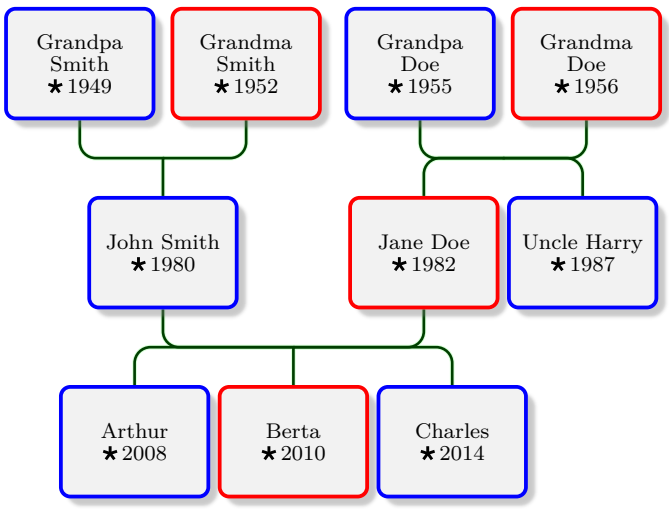
```
\begin{tikzpicture}
\genealogytree[template=signpost,edges={perpendicular}]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/edge/rounded=`⟨*length*⟩ (default `6pt`)

The edges are drawn in a perpendicular but rounded style. The ⟨*length*⟩ describes the size of the rounding.

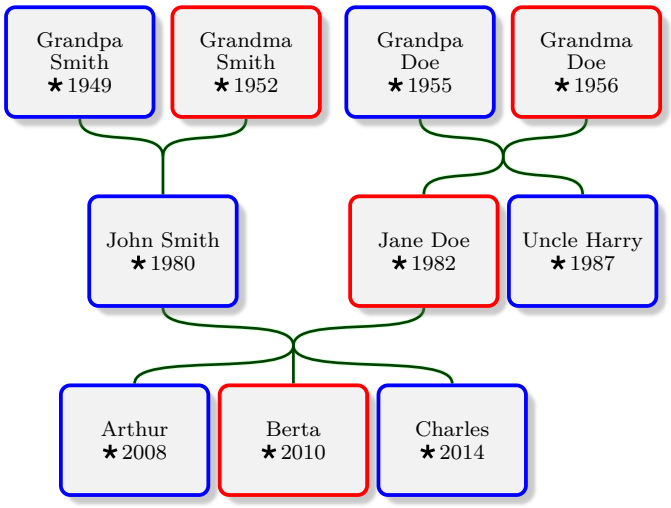
```
\begin{tikzpicture}
\genealogytree[template=signpost,edges={rounded=6pt}]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/edge/swing=`⟨*length*⟩ (default `12pt`)

The edges are drawn in a swinging style. The ⟨*length*⟩ describes the control parameter of the underlying curved path.

```
\begin{tikzpicture}
\genealogytree[template=signpost,edges={swing=12pt}]
  {input{example.option.graph}}
\end{tikzpicture}
```

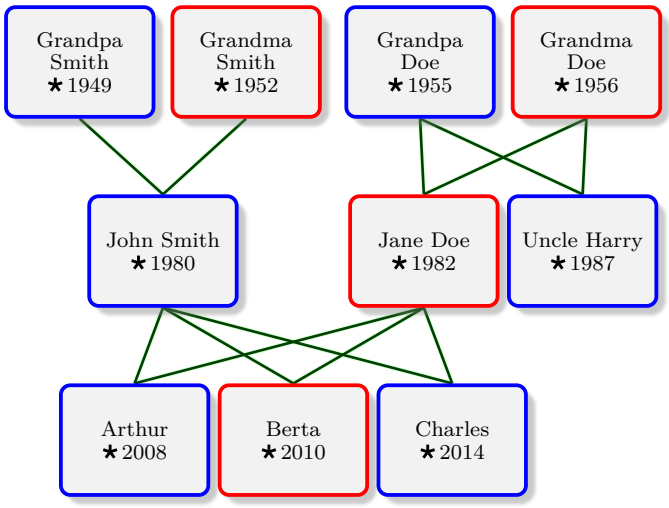


`/gtr/edge/mesh={⟨options⟩}` (default empty)

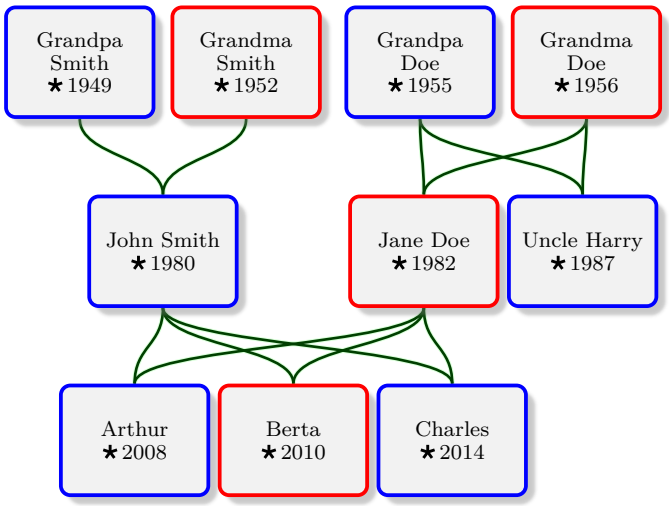
The edges are drawn meshed between parents and children. The ⟨*options*⟩ are TikZ `to path` options.

- For a family without children or without parents, no edge will be drawn.
- For a family with at least two parents and at least two children, a mesh is drawn. The intended use case is for families with just one parent or just one child, i.e. for ordinary trees.

```
\begin{tikzpicture}
\genealogytree[template=signpost,edges=mesh]
  {input{example.option.graph}}
\end{tikzpicture}
```




```
\begin{tikzpicture}
\genealogytree[template=signpost,
  edges={mesh={%
    to path={.. controls +(270:0.5) and +(90:0.5) .. (\tikztotarget)}
}}]{input{example.option.graph}}
\end{tikzpicture}
```



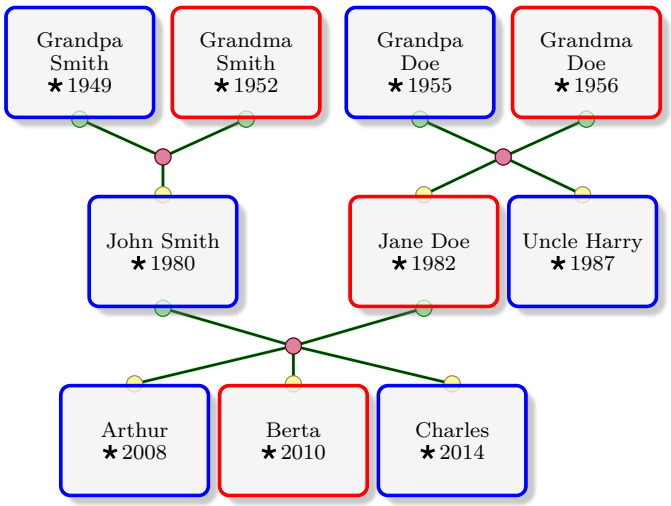
`/gtr/edge/custom={⟨down⟩}{⟨up⟩}{⟨left⟩}{⟨right⟩}` (no default)

The edges are drawn in a custom style. This option takes four mandatory parameters ⟨*down*⟩, ⟨*up*⟩, ⟨*left*⟩, and ⟨*right*⟩, each of which is a macro. The ⟨*down*⟩ macro is used to draw edges for `/gtr/timeflow` → P. 78`=down`, etc.

Every macro has to take four mandatory parameters:

1. An `etoolbox` ⟨*listmacro*⟩ which contains the list of anchor positions for the parents.
2. An `etoolbox` ⟨*listmacro*⟩ which contains the list of anchor positions for the children.
3. A Ti*k*Z node name which denotes the family core (center).
4. A Ti*k*Z style which should be applied to draw the edges.

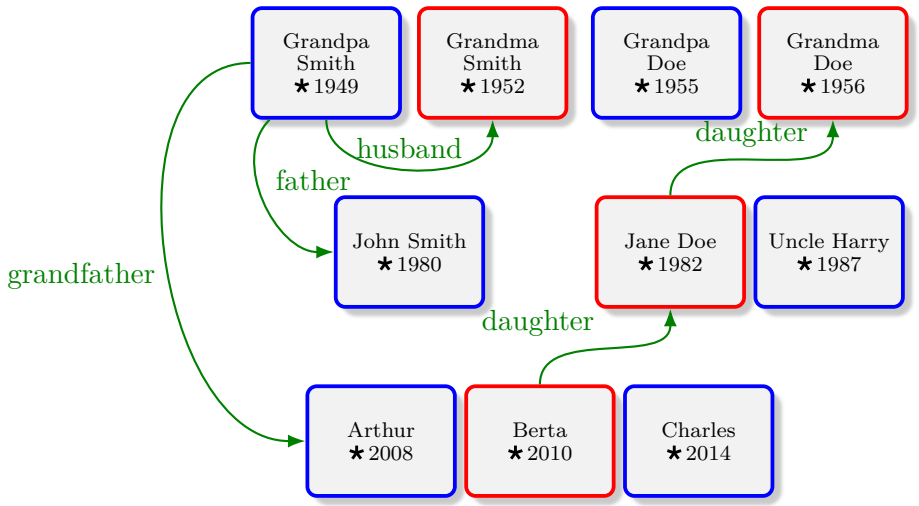
```
\newcommand{\myedgedraw}[4]{%
  % parents (#1):
  \renewcommand*{\do}[1]{
    \draw[#4] (##1)--(#3);
    \path[draw=green!50!black,fill=green!30] (##1) circle (3pt);}%
  \dolistloop{#1}%
  % children (#2):
  \renewcommand*{\do}[1]{
    \draw[#4] (##1)--(#3);
    \path[draw=yellow!50!black,fill=yellow!50] (##1) circle (3pt);}%
  \dolistloop{#2}%
  % family core (#3):
  \path[draw=purple!50!black,fill=purple!50] (#3) circle (3pt);
}

\begin{tikzpicture}
\genealogytree[template=signpost,
  box={enhanced jigsaw,opacityback=0.75},
  edges={
    custom={\myedgedraw}{\myedgedraw}{\myedgedraw}{\myedgedraw},
  }]
{input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/edge/none` (no value)

This is a special `/gtr/edge/custom` → P. 197 style which simply draws nothing. May be used for just this purpose or to replace automatic edge drawing by manual edge drawing.

```
\usetikzlibrary{quotes}
\begin{tikzpicture}
\genealogytree[template=signpost,edges=none]
  {input{example.option.graph}}

\draw[-Latex,green!50!black,thick]
  (GpSm1949) edge[out=270,in=270,"husband"] (GmSm1952)
             edge[out=225,in=180,"father"] (John1980)
             edge[out=180,in=180,"grandfather"'] (Arth2008)
  (Bert2010) edge[out=90,in=270,"daughter"] (Jane1982)
  (Jane1982) edge[out=90,in=270,"daughter"] (GmDo1956);
\end{tikzpicture}
```

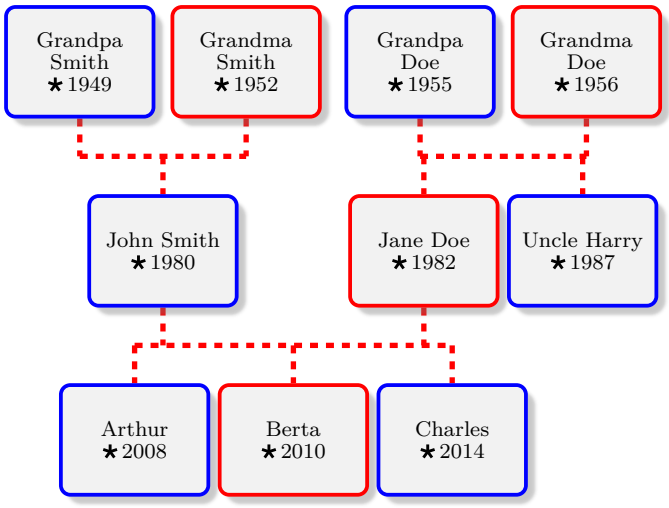


8.3 Edge Parameters

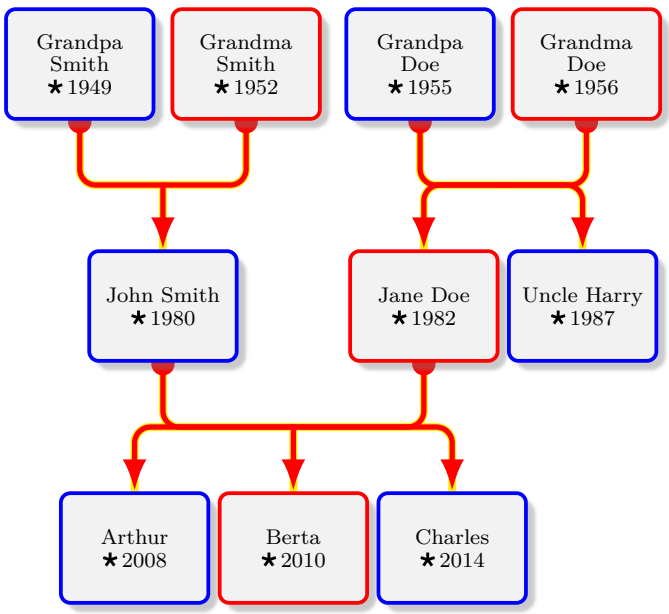
`/gtr/edge/foreground`={⟨*tikz options*⟩} (style, no default)

Defines the foreground ⟨*tikz options*⟩ for drawing the edges between the nodes.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  edges={foreground={line width=2pt,red,dashed,line cap=butt},no background}]
  {input{example.option.graph}}
\end{tikzpicture}
```




```
\begin{tikzpicture}
\genealogytree[template=signpost,level distance=1.7cm,
  edges={rounded,foreground={line width=2pt,red,Circle-LaTeX,shorten <=-4pt},
    background={line width=3pt,yellow}}]
  {input{example.option.graph}}
\end{tikzpicture}
```



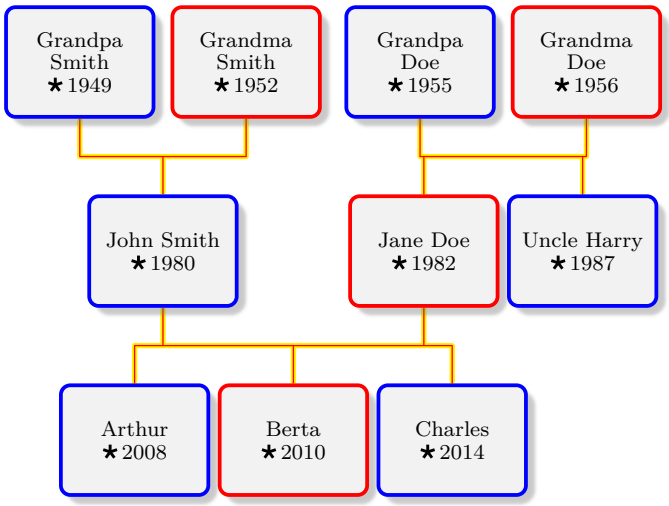
`/gtr/edge/no foreground` (style, no value)

Removes the `/gtr/edge/foreground` edges.

`/gtr/edge/background={⟨tikz options⟩}` (style, no default)

Defines the background ⟨*tikz options*⟩ for drawing the edges between the nodes.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  edges={foreground={line width=0.5pt,red},
    background={line width=2pt,yellow}}]
  {input{example.option.graph}}
\end{tikzpicture}
```



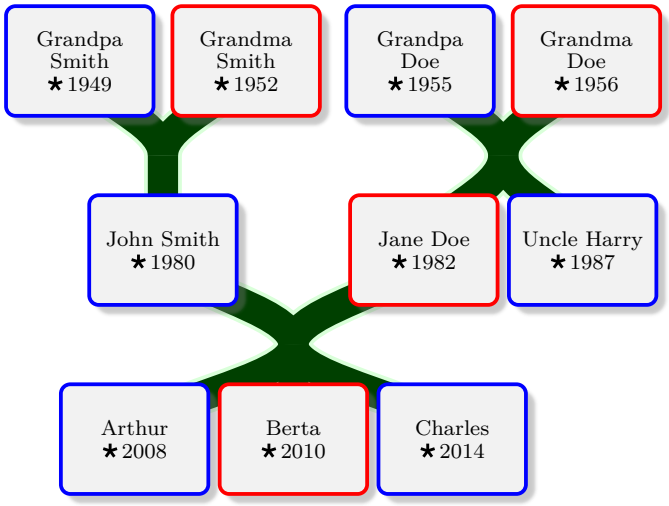
`/gtr/edge/no background` (style, no value)

Removes the `/gtr/edge/background` edges.

`/gtr/edge/anchoring=periphery|center` (no default, initially `periphery`)

Defines anchoring points for the edges. Feasible value are `periphery` and `center`.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  edges={swing=5mm,anchoring=center,
    foreground={line width=4mm},background={line width=5mm}}]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/edge/hide single leg=true|false` (default `true`, initially `true`)

If set to `true`, the orphan leg of a family with just one member is hidden.

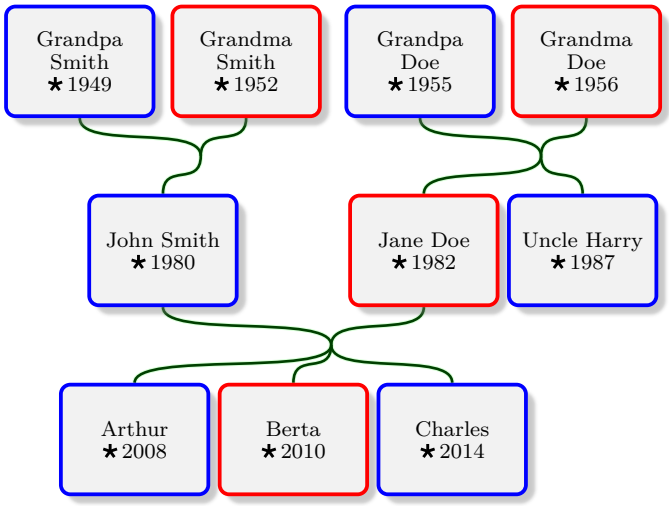
```
\begin{tikzpicture}
\genealogytree[template=signpost,
  edges={hide single leg}]
{
  parent{ g{Orphan} }
}
\end{tikzpicture}
```


```
\begin{tikzpicture}
\genealogytree[template=signpost,
  edges={hide single leg=false}]
{
  parent{ g{Orphan} }
}
\end{tikzpicture}
```

`/gtr/edge/xshift=`⟨*length*⟩ (no default, in initially `0pt`)

Shifts the edge core position horizontally by ⟨*length*⟩.

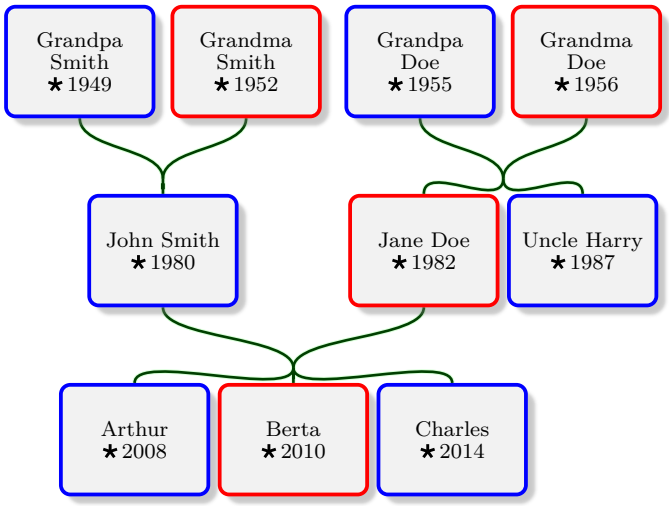
```
\begin{tikzpicture}
\genealogytree[template=signpost,edges={swing,xshift=5mm}]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/edge/yshift=`⟨*length*⟩ (no default, in initially `0pt`)

Shifts the edge core position vertically by ⟨*length*⟩.

```
\begin{tikzpicture}
\genealogytree[template=signpost,edges={swing,yshift=-3mm}]
  {input{example.option.graph}}
\end{tikzpicture}
```

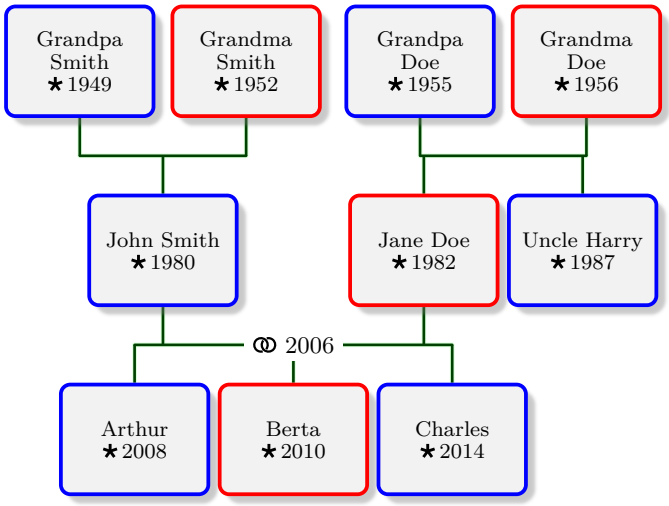


8.4 Edge Labels

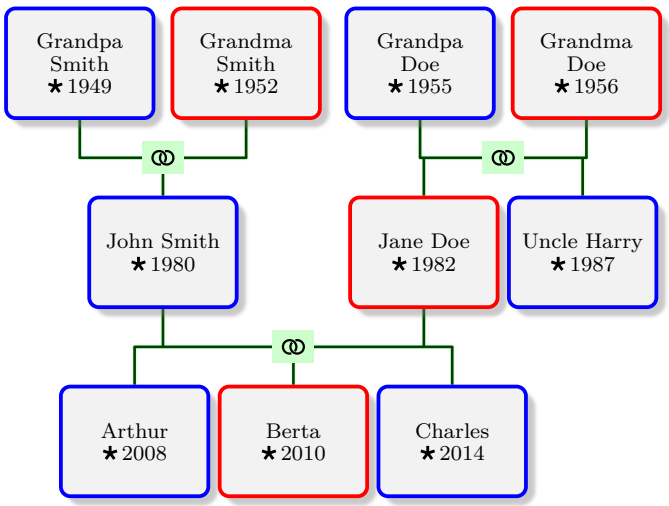
`/gtr/label={`⟨*text*⟩`}` (style, no default)

Adds a label ⟨*text*⟩ to the current family. This is realized by a Ti*k*Z node with `/gtr/label options`. The current family is determined by a surrounding `/gtr/family` → P. 103 or `/gtr/options for family` → P. 102.

```
\begin{genealogypicture}[template=signpost,
    label options={fill=white,node font=\footnotesize},
    options for family={SmithDoe}{label={\gtrsymMarried~2006}} ]
  input{example.option.graph}
\end{genealogypicture}
```




```
\begin{genealogypicture}[template=signpost,
    label options={fill=green!20,node font=\footnotesize},
    label={\gtrsymMarried} ]
  input{example.option.graph}
\end{genealogypicture}
```



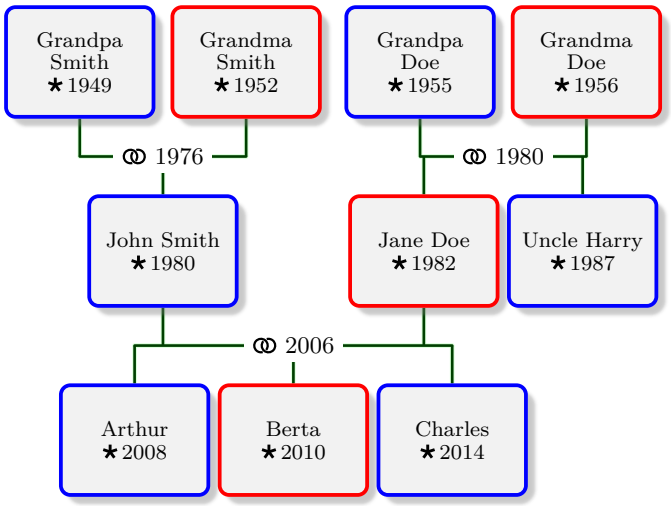
`/gtr/label options={`⟨*options*⟩`}` (style, no default)

Sets Ti*k*Z `node` ⟨*options*⟩ to be used for `/gtr/label`. See `/gtr/label` for an example.

`/gtr/family label={⟨text⟩}` (style, no default)

Shortcut for using `/gtr/label` → P. 203 inside `/gtr/family` → P. 103.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  label options={fill=white,node font=\footnotesize},
]{
  parent[id=SmithDoe,family label={\gtrsymMarried~2006}]{
    g[id=Arth2008,male]{Arthur\\\gtrsymBorn\,2008}
    c[id=Bert2010,female]{Berta\\\gtrsymBorn\,2010}
    c[id=Char2014,male]{Charles\\\gtrsymBorn\,2014}
    parent[id=Smith,family label={\gtrsymMarried~1976}]{
      g[id=John1980,male]{John Smith\\\gtrsymBorn\,1980}
      p[id=GpSm1949,male]{Grandpa Smith\\\gtrsymBorn\,1949}
      p[id=GmSm1952,female]{Grandma Smith\\\gtrsymBorn\,1952}
    }
    parent[id=Doe,family label={\gtrsymMarried~1980}]{
      g[id=Jane1982,female]{Jane Doe\\\gtrsymBorn\,1982}
      c[id=Harr1987,male]{Uncle Harry\\\gtrsymBorn\,1987}
      p[id=GpDo1955,male]{Grandpa Doe\\\gtrsymBorn\,1955}
      p[id=GmDo1956,female]{Grandma Doe\\\gtrsymBorn\,1956}
    }
  }
}
\end{tikzpicture}
```



`/gtr/subtree label={⟨text⟩}` (style, no default)

Shortcut for using `/gtr/label` → P. 203 inside `/gtr/subtree` → P. 106.

8.5 Edge Labels Database

Analog to database processing for nodes, see Chapter 7 on page 151, the edge labels can be formatted by database style entries.

The database content for edge labels has to be given inside the option list for a `parent` or `child` using `/gtr/family database`.

`/gtr/family database`={⟨*data keys*⟩} (no default, initially empty)

Sets ⟨*data keys*⟩ for the edge labeling of the current family. For ⟨*data keys*⟩, any setting from Section 7.3 on page 155 can be used, but only marriage information or similar may be reasonable.

```
%...
child[id=SmitBowd1742,family database={marriage={1742-03-02}{London}}]{
%...
```

`/gtr/label database options`={⟨*options*⟩} (no default)

The ⟨*options*⟩ settings define how the `/gtr/family database` values are used to create label content. The default operations are `/gtr/use family database` and `/gtr/database format`→ P. 162=`marriage`. Note that setting `/gtr/database format`→ P. 162 inside `/gtr/label database options` does only change the format for edge labels, but not for nodes.

```
%...
  label database options={
    database format=marriage, %  that is the default value
    place text={(}{)}         %  changed only for labels
  },
%...
```

`/gtr/ignore family database` (no value)

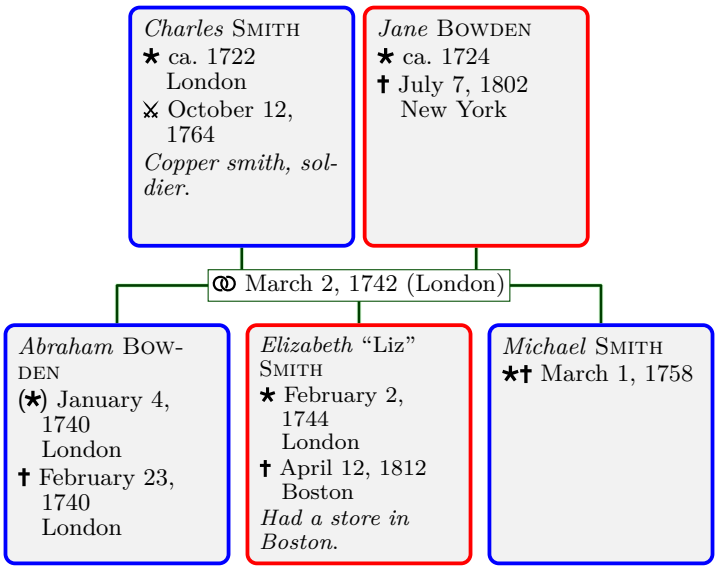
If set, then all `/gtr/family database` values are simply ignored. This has to be used inside `/gtr/label database options` to have an effect.

```
%...
  label database options={ignore family database},
%...
```

`/gtr/use family database` (no value)

If set, then all `/gtr/family database` values are processed to generate label content. This has to be used inside `/gtr/label database options` to have an effect.

```
\begin{genealogypicture}[
  processing=database,database format=medium no marriage,
  node size=3cm,level size=3.2cm,level distance=1cm,
  list separators hang,place text={\newline}{},
  box={fit basedim=9pt,boxsep=2pt,segmentation style=solid,
    halign=left,before upper=\parskip1pt,\gtrDBsex },
  label database options={place text={(}{)}},
  label options={fill=white,node font=\footnotesize,inner sep=0.5mm,draw=green!30!black},
]
child[id=SmitBowd1742,family database={marriage={1742-03-02}{London}}]{
  g[id=SmitChar1722]{
    male,
    name    = {\pref{Charles} \surn{Smith}},
    birth   = {(caAD)1722}{London},
    baptism = {1722-04-13}{London},
    death+  = {1764-10-12}{}{killed},
    comment = {Copper smith, soldier},
  }
  p[id=BowdJane1724]{
    female,
    name     = {\pref{Jane} \surn{Bowden}},
    birth-   = {(caAD)1724},
    death    = {1802-07-07}{New York},
  }
  c[id=BowdAbra1740]{
    male,
    name   = {\pref{Abraham} \surn{Bowden}},
    birth+ = {1740-01-04}{London}{out of wedlock},
    death  = {1740-02-23}{London}
  }
  c[id=SmitEliz1744]{
    female,
    name    = {\pref{Elizabeth} \nick{Liz} \surn{Smith}},
    birth   = {1744-02-02}{London},
    death   = {1812-04-12}{Boston},
    comment = {Had a store in Boston},
  }
  c[id=SmitMich1758]{
    male,
    name   = {\pref{Michael} \surn{Smith}},
    birth+ = {1758-03-01}{}{died},
  }
}
\end{genealogypicture}
```

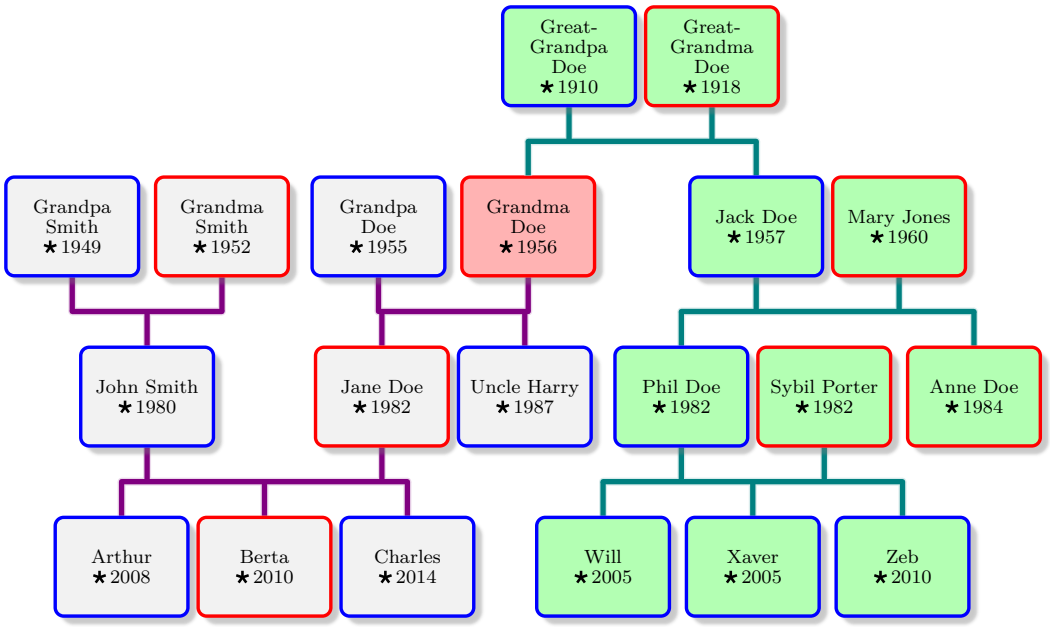


8.6 Adding and Removing Nodes from Edge Drawing

`/gtr/add child=`⟨*child*⟩ `to` ⟨*family*⟩ (style, no default)

Connect a node of an existing graph as ⟨*child*⟩ to a ⟨*family*⟩ of the current graph. The auto-layout algorithm is not aware of this addition.

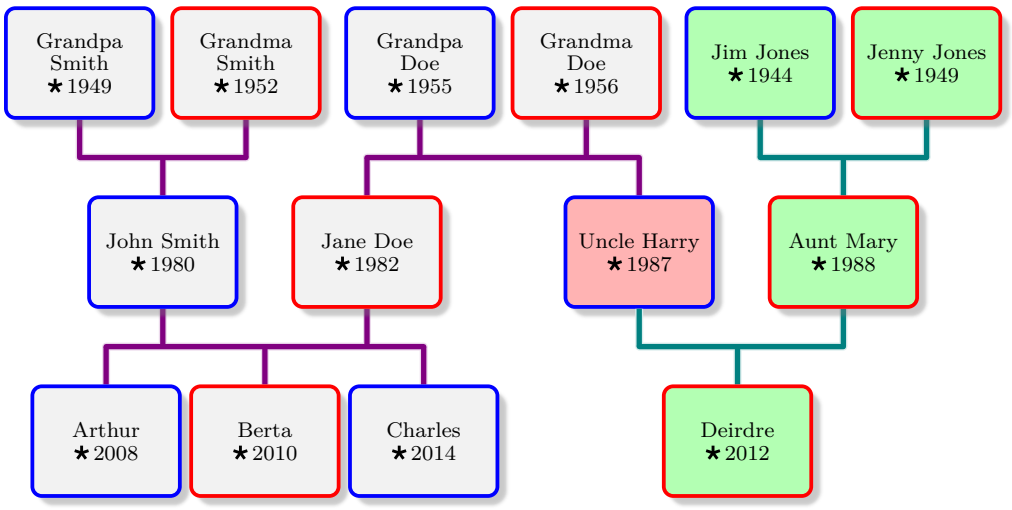
```
\begin{tikzpicture}[scale=0.9,transform shape]
\gtrset{template=signpost}
\genealogytree[
    edges={foreground={red!50!blue,line width=2pt},
           background={red!50!blue!15!white,line width=3pt}},
    options for node={GmDo1956}{box={colback=red!30!white}}  ]
  {input{example.option.graph}}

\genealogytree[
    edges={foreground={green!50!blue,line width=2pt},
           background={green!50!blue!15!white,line width=3pt}},
    box={colback=green!30!white},
    adjust node=PhDo1982 right of Harr1987 distance 3mm,
    add child=GmDo1956 to GreatDoe    ]
{
  child[id=GreatDoe,pivot shift=1.7cm]{
    g[id=GgpDo1910,male]{Great-Grandpa Doe\\\gtrsymBorn\,1910}
    p[id=GgmDo1918,female]{Great-Grandma Doe\\\gtrsymBorn\,1918}
    child{
      g[id=JaDo1957,male]{Jack Doe\\\gtrsymBorn\,1957}
      p[id=MaJo1960,female]{Mary Jones\\\gtrsymBorn\,1960}
      child{
        g[id=PhDo1982,male]{Phil Doe\\\gtrsymBorn\,1982}
        p[id=SyPo1982,female]{Sybil Porter\\\gtrsymBorn\,1982}
        c[id=Will2005,male]{Will\\\gtrsymBorn\,2005}
        c[id=Xave2005,male]{Xaver\\\gtrsymBorn\,2005}
        c[id=Zeb2010,male]{Zeb\\\gtrsymBorn\,2010}
      }
      c[id=AnDo1984,female]{Anne Doe\\\gtrsymBorn\,1984}
    }
  }
}
\end{tikzpicture}
```



`/gtr/add parent=`⟨*parent*⟩ `to` ⟨*family*⟩ (style, no default)

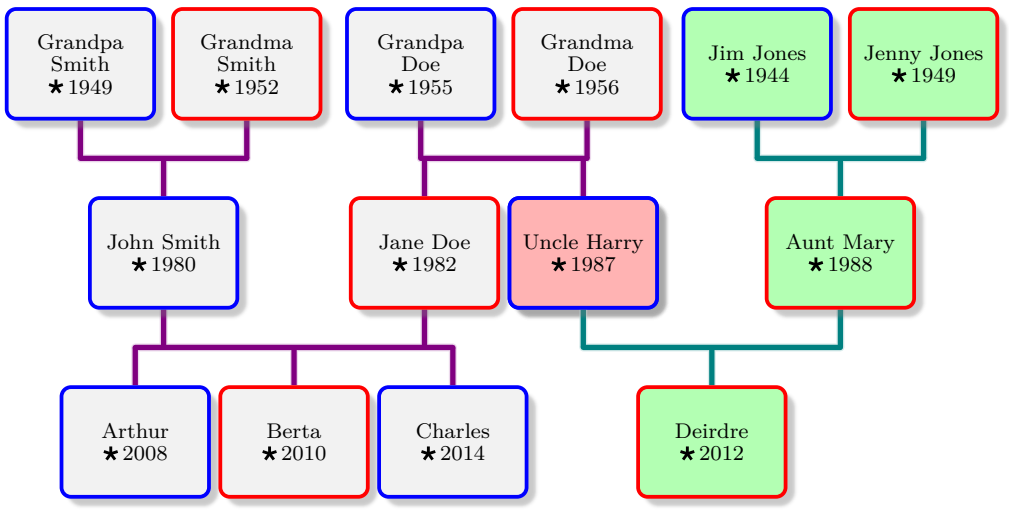
Connect a node of an existing graph as ⟨*parent*⟩ to a ⟨*family*⟩ of the current graph. The auto-layout algorithm is not aware of this addition.

```
\begin{tikzpicture}
\gtrset{template=signpost}
\genealogytree[
    edges={foreground={red!50!blue,line width=2pt},
           background={red!50!blue!15!white,line width=3pt}},
    options for node={Harr1987}{distance=1.6cm,box={colback=red!30!white}}  ]
  {input{example.option.graph}}

\genealogytree[
    edges={foreground={green!50!blue,line width=2pt},
           background={green!50!blue!15!white,line width=3pt}},
    box={colback=green!30!white},
    adjust node=JimJ1944 right of GmDo1956 distance 3mm,
    add parent=Harr1987 to DoeJones  ]
{
  parent[id=DoeJones,pivot shift=-1.4cm]{
    g[id=Deir2012,female]{Deirdre\\\gtrsymBorn\,2012}
    parent[id=Jones]{
      g[id=Mary1988,female]{Aunt Mary\\\gtrsymBorn\,1988}
      p[id=JimJ1944,male]{Jim Jones\\\gtrsymBorn\,1944}
      p[id=Jenn1949,female]{Jenny Jones\\\gtrsymBorn\,1949}
    }
  }
}
\end{tikzpicture}
```



An alternative approach to /gtr/add child[→ P. 207] and /gtr/add parent[→ P. 208] is to draw the interconnecting node twice (the first one could be drawn as /gtr/phantom*[→ P. 124]). The second instance is drawn over the first instance using /gtr/set position[→ P. 109]. Both instances need to have different /gtr/id[→ P. 90] values. Note that both parts of the graph can still be overlapping and may have to be adjusted manually, since the auto-layout algorithms handles each \genealogytree[→ P. 55] separately. The second tree gets an /gtr/id prefix[→ P. 92] of 2: to address the second Uncle Harry by 2:Harr1987.

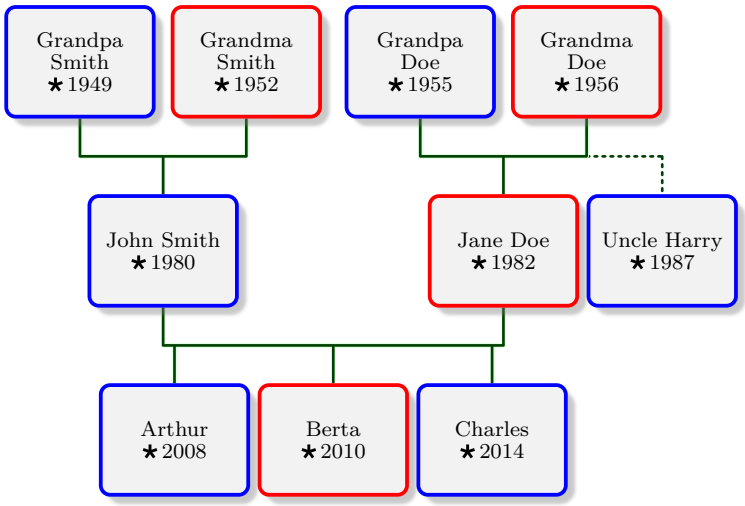
```
\begin{tikzpicture}
\gtrset{template=signpost}
\genealogytree[
    edges={foreground={red!50!blue,line width=2pt},
           background={red!50!blue!15!white,line width=3pt}},
    ]
  {input{example.option.graph}}

\genealogytree[id prefix=2:,
    edges={foreground={green!50!blue,line width=2pt},
           background={green!50!blue!15!white,line width=3pt}},
    box={colback=green!30!white},
    set position=2:Harr1987 at Harr1987,
    ]
{
  parent[id=DoeJones]{
    g[id=Deir2012,female]{Deirdre\\\gtrsymBorn\,2012}
    p[id=Harr1987,male,box={colback=red!30!white}]{Uncle Harry\\\gtrsymBorn\,1987}
    parent[id=Jones]{
      g[id=Mary1988,female,distance=1.4cm]{Aunt Mary\\\gtrsymBorn\,1988}
      p[id=JimJ1944,male]{Jim Jones\\\gtrsymBorn\,1944}
      p[id=Jenn1949,female]{Jenny Jones\\\gtrsymBorn\,1949}
    }
  }
}
\end{tikzpicture}
```



`/gtr/remove child=`⟨*child*⟩ `from` ⟨*family*⟩ (style, no default)

Removes a node as ⟨*child*⟩ from a ⟨*family*⟩ of the current graph. The auto-layout algorithm is not aware of this removal.

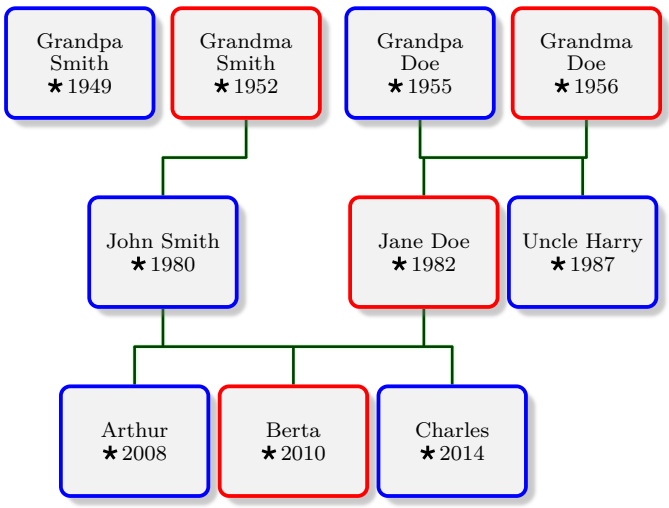
```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for node={Jane1982}{pivot=child},% make Jane the pivot child
  remove child=Harr1987 from Doe,          % remove Harry
  extra edges prepend for family=          % add Harry again with dots
    {Doe}{GmDo1956}{Harr1987}{foreground={dotted,line cap=round},
      no background}
]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/remove parent=`⟨*parent*⟩ `from` ⟨*family*⟩ (style, no default)

Removes a node as ⟨*parent*⟩ from a ⟨*family*⟩ of the current graph. The auto-layout algorithm is not aware of this removal.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
    remove parent=GpSm1949 from Smith ]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/disconnect=`⟨*value*⟩ (default `both`, initially unset)

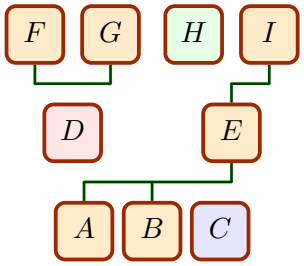
Using this option inside node options disconnects the current node from the edges of the current graph. Using this option elsewhere may cause unwanted side effects. The auto-layout algorithm is not aware of this removal. Depending of the given ⟨*value*⟩, the node is disconnected as parent or child or both.

Feasible values are

- `child`: disconnect the node as child of a family.
- `parent`: disconnect the node as parent of a family. Note that a `g` node is only removed from its primary family, but not from connected `union` families.
- `both`: disconnect the node as child and as parent.

`/gtr/remove child` → P. 210 and `/gtr/remove parent` → P. 210 allow more precise control, but `/gtr/disconnect` needs no `/gtr/id` → P. 90 values.

```
\begin{tikzpicture}
\genealogytree[template=formal graph]
{
  parent{
    g{A}
    c{B}
    c[disconnect,box={colback=blue!10}]{C}
    parent{
      g[disconnect,box={colback=red!10}]{D}
      p{F}
      p{G}
    }
    parent{
      g{E}
      p[disconnect,box={colback=green!10}]{H}
      p{I}
    }
  }
}
\end{tikzpicture}
```

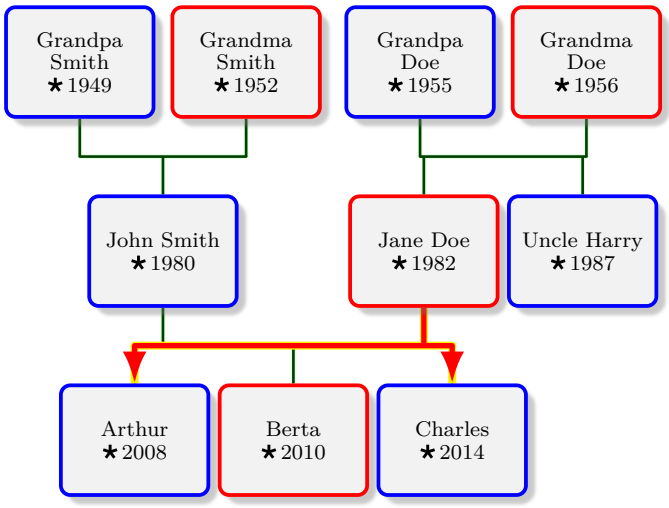


8.7 Extra Edges

`/gtr/extra edges={⟨parents⟩}{⟨children⟩}{⟨edge options⟩}` (style, no default)

Appends an extra set of edges to the current family. The edges are drawn between the given ⟨*parents*⟩ list and the given ⟨*children*⟩ list using the ⟨*edge options*⟩. Note that parents and children are defined by their `/gtr/id` → P. 90 values. They do not necessarily have to be *real* members of the current family. The current family is given by a surrounding `/gtr/family` → P. 103 or `/gtr/options for family` → P. 102.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for family={SmithDoe}{extra edges={Jane1982}{Arth2008,Char2014}{
    foreground={red,line width=2pt,-Latex},background={yellow,line width=3pt}}},
]
  {input{example.option.graph}}
\end{tikzpicture}
```



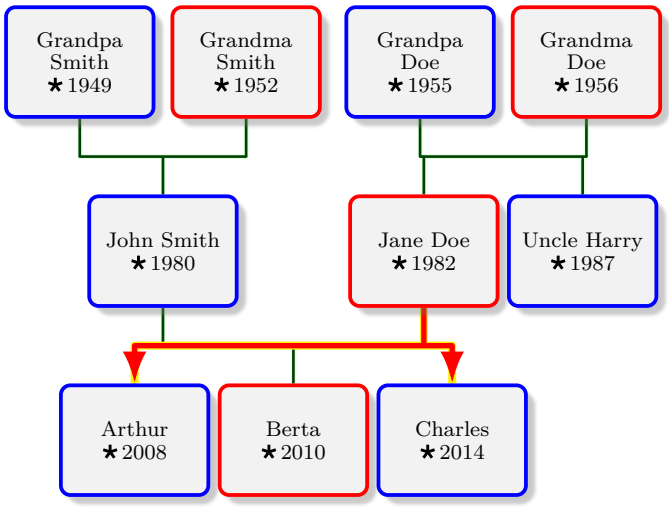
`/gtr/family extra edges={⟨parents⟩}{⟨childs⟩}{⟨edge options⟩}` (style, no default)

This is a shortcut for embedding `/gtr/extra edges` into `/gtr/family` → P. 103.

/gtr/extra edges for family={⟨*family*⟩}{⟨*parents*⟩}{⟨*childs*⟩}{⟨*edge options*⟩} (style, no default)

This is a shortcut for embedding /gtr/extra edges → P. 212 into /gtr/options for family → P. 102.

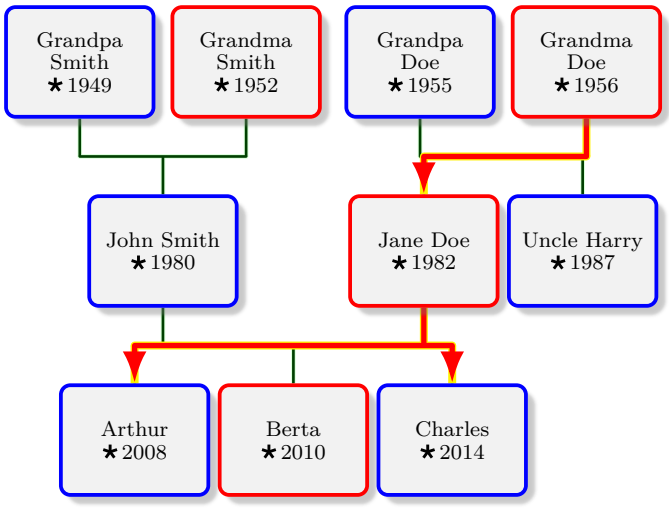
```
\begin{tikzpicture}
\genealogytree[template=signpost,
  extra edges for family={SmithDoe}{Jane1982}{Arth2008,Char2014}{
    foreground={red,line width=2pt,-Latex},background={yellow,line width=3pt}},
]
  {input{example.option.graph}}
\end{tikzpicture}
```



/gtr/extra edges for families={⟨*family list*⟩}{⟨*edge options*⟩} (style, no default)

This allows to set /gtr/extra edges for family for multiple families. Therefore, the ⟨*family list*⟩ is a comma separated list of entries of type x={⟨*family*⟩}{⟨*parents*⟩}{⟨*children*⟩}

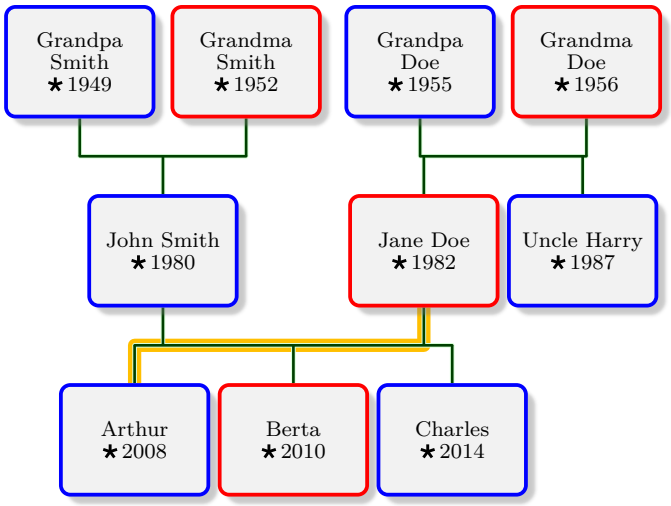
```
\begin{tikzpicture}
\genealogytree[template=signpost,
  extra edges for families={x={Doe}{GmDo1956}{Jane1982},
      x={SmithDoe}{Jane1982}{Arth2008,Char2014}}{
    foreground={red,line width=2pt,-Latex},background={yellow,line width=3pt}},
]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/extra edges prepend={⟨parents⟩}{⟨children⟩}{⟨edge options⟩}` (style, no default)

Appends an extra set of edges to the current family. The edges are drawn between the given ⟨*parents*⟩ list and the given ⟨*children*⟩ list using the ⟨*edge options*⟩. This is identical to `/gtr/extra edges` → P. 212, but the drawing lies under the normal edges.

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  options for family={SmithDoe}{extra edges prepend={Jane1982}{Arth2008}{
    foreground={red!25!yellow,line width=5pt},no background}},
]
  {input{example.option.graph}}
\end{tikzpicture}
```



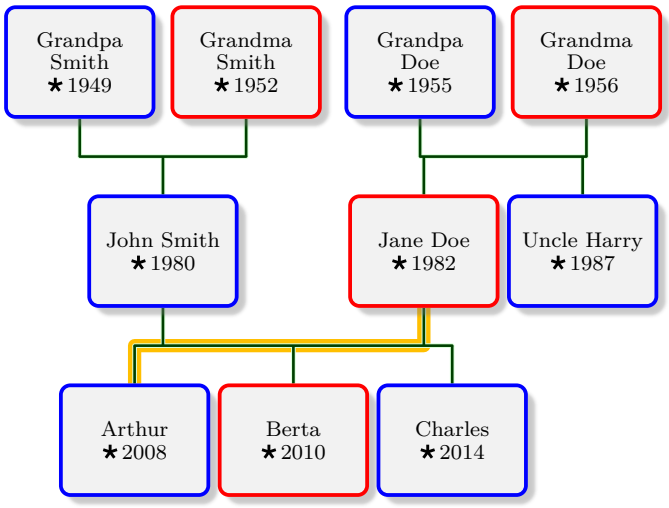
`/gtr/family extra edges prepend={⟨parents⟩}{⟨children⟩}{⟨edge options⟩}` (style, no default)

This is a shortcut for embedding `/gtr/extra edges prepend` into `/gtr/family` → P. 103.

`/gtr/extra edges prepend for family`={⟨*family*⟩}{⟨*parents*⟩}{⟨*children*⟩}{⟨*edge options*⟩} (style, no default)

This is a shortcut for embedding `/gtr/extra edges prepend` → P. 214 into `/gtr/options for family` → P. 102.

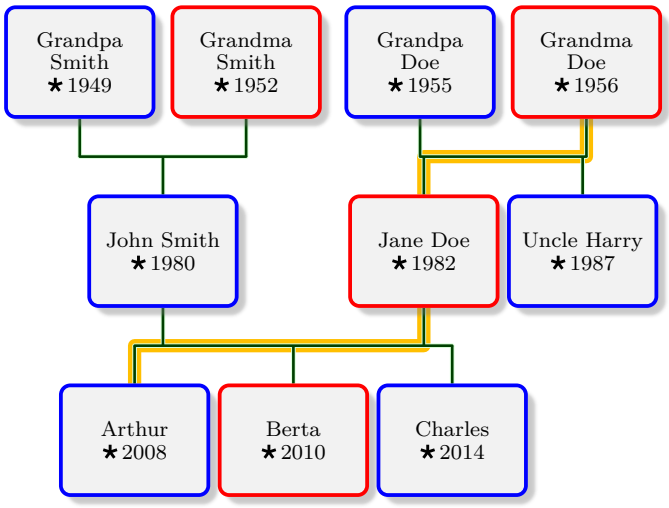
```
\begin{tikzpicture}
\genealogytree[template=signpost,
  extra edges prepend for family={SmithDoe}{Jane1982}{Arth2008}{
    foreground={red!25!yellow,line width=5pt},no background},
]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/extra edges prepend for families`={⟨*family list*⟩}{⟨*edge options*⟩} (style, no default)

This allows to set `/gtr/extra edges prepend for family` for multiple families. Therefore, the ⟨*family list*⟩ is a comma separated list of entries of type `x`={⟨*family*⟩}{⟨*parents*⟩}{⟨*children*⟩}

```
\begin{tikzpicture}
\genealogytree[template=signpost,
  extra edges prepend for families={x={Doe}{GmDo1956}{Jane1982},
      x={SmithDoe}{Jane1982}{Arth2008}}{
    foreground={red!25!yellow,line width=5pt},no background}
]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/tikz/genealogytree extra edges scope` (style, initially empty) N 2015-10-23

This style is used to scope the drawing of extra edges. It may be redefined e.g. to draw edges on a certain layer. This scope is embedded into the general scope of `/tikz/genealogytree edges scope` → P. 193.

```
% draw extra edges on the background layer
\usetikzlibrary{backgrounds}
\tikzset{genealogytree extra edges scope/.style={on background layer}}
```

Note that changing the drawing layer for extra edges applies to all extra edges, but does not change the drawing layer for *normal* edges. Therefore, *all* extra edges would be drawn behind *normal* edges, if the example above is used.

9

Genealogy Symbols

9.1 Symbol Color Settings

If the genealogy symbols are only needed in black color, there is nothing special to consider. Currently, the symbols are drawn as pictures and saved in boxes for efficiency. If different colors are needed, the symbols have to be redrawn. The named color `gtrsymbol` holds the (current) symbol color.

9.1.1 Global Color Settings

In the preamble, the color of all genealogy symbols can be set by redefining the color `gtrsymbol`. For example, if all symbols should be created in blue, one can use:

```
% ...
\colorlet{gtrsymbol}{blue}
% ...
\begin{document}
% ...
\end{document}
```

Note that this setting has to be given inside the preamble *after* the package is loaded and *before* `\begin{document}`.

9.1.2 Local Color Settings

If symbols with a color different from the global symbol color should be used inside the document, one of the following commands can be used.

`\gtrSymbolsSetCreate{`⟨*color*⟩`}`

Recreates all symbols for the current TEX group with the given ⟨*color*⟩. The named color `gtrsymbol` will also be set to ⟨*color*⟩. Use this macro, if it is expected that many symbols of this color will be used inside the current TEX group.

```
\gtrsymBorn\,14.XI.1475
{
  \gtrSymbolsSetCreate{red}
  \gtrsymMarried\,22.II.1502,
  \gtrsymDied\,8.X.1553,
}
\gtrsymBuried\,10.X.1553
```

★14.XI.1475 ⚭22.II.1502, †8.X.1553, ⚰10.X.1553

`\gtrSymbolsSetCreateSelected{`⟨*color*⟩`}{`⟨*list*⟩`}`

Recreates all symbols from the given comma separated ⟨*list*⟩ for the current TEX group with the given ⟨*color*⟩. The named color `gtrsymbol` will also be set to ⟨*color*⟩. The ⟨*list*⟩ contains the base names of the selected symbols, e.g. `Born` for `\gtrsymBorn` → P. 219. Symbols which are not present in this list, will keep their old color. Use this macro, if it is expected that many symbols of this color will be used inside the current TEX group.

```
\gtrSymbolsSetCreateSelected{blue}{Male}
\gtrSymbolsSetCreateSelected{red}{Female}
\gtrSymbolsSetCreateSelected{yellow!50!black}{Born,Died}

\gtrsymBorn, \gtrsymMale, \gtrsymFemale, \gtrsymNeuter, \gtrsymDied.
```

★, ♂, ♀, ⚲, †.

`\gtrSymbolsSetDraw{`⟨*color*⟩`}`

Inside the current TEX group, every symbol is drawn with the given ⟨*color*⟩ when it is used. It is drawn again, if it is used again. The named color `gtrsymbol` will also be set to ⟨*color*⟩. Use this macro, if it is expected that only few symbols of this color will be used inside the current TEX group or if colors constantly change.

```
\gtrsymBorn\,14.XI.1475
{
  \gtrSymbolsSetDraw{red}
  \gtrsymMarried\,22.II.1502,
  \gtrSymbolsSetDraw{blue}
  \gtrsymDied\,8.X.1553,
}
\gtrsymBuried\,10.X.1553
```

★14.XI.1475 ⚭22.II.1502, †8.X.1553, ⚰10.X.1553

9.2 List of Symbols

`\gtrsymBorn` ★

Birth / born (Unicode U+2A).

```
Johann Maier \gtrsymBorn\,14.XI.1475
```

Johann Maier ★ 14.XI.1475

`\gtrsymBornoutofwedlock` (★)

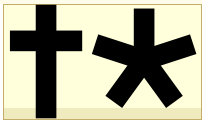
Born out of wedlock / illegitimate.

```
Johann Maier \gtrsymBornoutofwedlock\,14.XI.1475
```

Johann Maier (★) 14.XI.1475

`\gtrsymStillborn` †★

Stillborn.

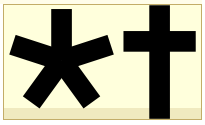


```
\textit{Anonymus} Maier \gtrsymStillborn\,14.XI.1475
```

Anonymus Maier †★ 14.XI.1475

`\gtrsymDiedonbirthday` ★†

Died on the birthday.

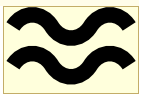


```
Johann Maier \gtrsymDiedonbirthday\,14.XI.1475
```

Johann Maier ★† 14.XI.1475

`\gtrsymBaptized` ≋

Baptism / baptized (Unicode U+2248).

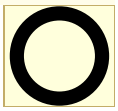


```
Johann Maier \gtrsymBaptized\,14.XI.1475
```

Johann Maier ≋ 14.XI.1475

`\gtrsymEngaged` ○

Engagement / engaged (Unicode U+26AC).



```
Johann Maier \gtrsymEngaged\,14.XI.1475
```

Johann Maier ○ 14.XI.1475

`\gtrsymMarried` ⚭

Marriage / married (Unicode U+26AD).

```
Johann Maier \gtrsymMarried\,14.XI.1475
```

Johann Maier ⚭ 14.XI.1475

`\gtrsymDivorced` ⚮

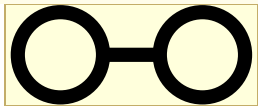
Divorce / divorced (Unicode U+26AE).

```
Johann Maier \gtrsymDivorced\,14.XI.1475
```

Johann Maier ⚮ 14.XI.1475

`\gtrsymPartnership` ⚯

Partnership / unmarried (Unicode U+26AF).



```
Johann Maier \gtrsymPartnership\,14.XI.1475
```

Johann Maier ⚯ 14.XI.1475

`\gtrsymDied` †

Death / died (Unicode U+2020, U+271D).

```
Johann Maier \gtrsymDied\,14.XI.1475
```

Johann Maier † 14.XI.1475

`\gtrsymKilled` ⚔

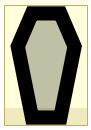
Killed in action / fallen (Unicode U+2694).

```
Johann Maier \gtrsymKilled\,14.XI.1475
```

Johann Maier ⚔ 14.XI.1475

\gtrsymBuried ⚰

Burial / buried (Unicode U+26B0).



```
Johann Maier \gtrsymBuried\,14.XI.1475
```

Johann Maier ⚰ 14.XI.1475

\gtrsymFuneralurn ⚱

Funeral urn / cremated (Unicode U+26B1).

```
Johann Maier \gtrsymFuneralurn\,14.XI.1475
```

Johann Maier ⚱ 14.XI.1475

N 2017-07-18 \gtrsymFloruit ⁘

Floruit / flourished.

```
Johann Maier \gtrsymFloruit\,1475--1503
```

Johann Maier ⁘ 1475–1503

\gtrsymFemale ♀

Female (Unicode U+2640).

```
Maria Maier \gtrsymFemale
```

Maria Maier ♀

\gtrsymMale ♂

Male (Unicode U+2642).

```
Johann Maier \gtrsymMale
```

Johann Maier ♂

\gtrsymNeuter ⚲

Neuter / Unknown sex (Unicode U+26B2).

```
\textit{Anonymus} Maier \gtrsymNeuter
```

Anonymus Maier ⚲

9.3 Legend to Symbols

The further macros and options allow to create a legend to symbols. This legend contains either all symbols or only the currently used symbols. Also, the description texts can be adapted to different languages or individual settings.

9.3.1 Printing a Legend

`\gtrSymbolsRecordReset`

The occurrence of a symbol inside the document text is recorded. `\gtrSymbolsLegend` → P. 223 prints all recorded symbols. To clear the current recording (locally), `\gtrSymbolsRecordReset` can be used. Note that records are taken globally, but resets are local to the current TEX group.

```
\gtrSymbolsRecordReset
Use symbol: \gtrsymBorn\par
{
  \gtrSymbolsRecordReset
  Use symbol inside group: \gtrsymMarried\par
  {
    Use symbol further inside: \gtrsymDied\par
  }
  Local legend inside group: \gtrSymbolsLegend\par
}
Global legend: \gtrSymbolsLegend
```

Use symbol: ★
Use symbol inside group: ⚭
Use symbol further inside: †
Local legend inside group: ⚭=married, †=died.
Global legend: ★=born, ⚭=married, †=died.

`/gtr/symbols record reset` (no value)

Identical to `\gtrSymbolsRecordReset`. This option is useful for application inside `\genealogytree` → P. 55 or `genealogypicture` → P. 57. See `\gtrSymbolsLegend` → P. 223 for an example.

`\gtrSymbolsFullLegend`[⟨*language*⟩] U 2017-07-18

Prints a full unabridged legend to symbols according to `/gtr/language` → P. 225 or optionally according to ⟨*language*⟩.

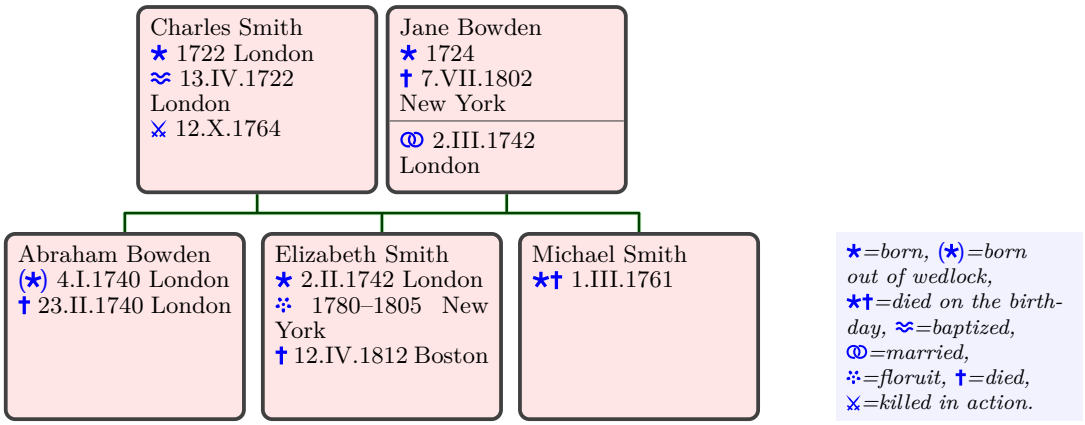
```
\textsl{\gtrSymbolsFullLegend[english]}
```

★=*born*, (★)=*born out of wedlock*, †★=*stillborn*, ★†=*died on the birthday*, ≈=*baptized*, ○=*engaged*, ⚭=*married*, ○|○=*divorced*, ○–○=*unmarried partnership*, ⁘=*floruit*, †=*died*, ⚔=*killed in action*, ⚱=*buried*, ⚱=*cremated*, ♀=*female*, ♂=*male*, ⚲=*neuter*.

`\gtrSymbolsLegend[⟨language⟩]`

Prints a legend to symbols according to `/gtr/language` → P. 225 or optionally according to ⟨*language*⟩. The legend only contains these symbols which were actually used inside the document so far or since the last call to `\gtrSymbolsRecordReset` → P. 222 or `/gtr/symbols record reset` → P. 222.

```
\begin{genealogypicture}[node size=3.2cm,level size=2.5cm,
  box={fit basedim=9pt,boxsep=2pt,colback=red!10,segmentation style={solid}},
  symbols record reset,code={\gtrSymbolsSetCreate{blue}},
  after tree={\node[font=\scriptsize\itshape,text width=3cm,above right,
      fill=blue!5] at ([xshift=1cm]Michael.south east) {\gtrSymbolsLegend};},
]
  child{
    g[id=Charles]{Charles Smith\par\gtrsymBorn~1722 London\par
      \gtrsymBaptized~13.IV.1722\\ London\par
      \gtrsymKilled~12.X.1764}
    p{Jane Bowden\par\gtrsymBorn~1724\par
      \gtrsymDied~7.VII.1802\\ New York
      \tcbline
      \gtrsymMarried~2.III.1742\\ London
      }
    c{Abraham Bowden\par
      \gtrsymBornoutofwedlock~4.I.1740 London\par
      \gtrsymDied~23.II.1740 London}
    c{Elizabeth Smith\par
      \gtrsymBorn~2.II.1742 London\par
      \gtrsymFloruit~1780--1805 New York\par
      \gtrsymDied~12.IV.1812 Boston}
    c[id=Michael]{Michael Smith\par\gtrsymDiedonbirthday~1.III.1761}
  }
\end{genealogypicture}
```



9.3.2 Description Texts and Language Settings

The following options of the key family `/gtr/symlang/` are language dependent and can be set globally by `/gtr/language` → P. 225.

english

```
\gtrset{language=english}
% ...
\gtrSymbolsFullLegend
```

★=*born*, (★)=*born out of wedlock*, †★=*stillborn*, ★†=*died on the birthday*, ≈=*baptized*, ○=*engaged*, ⚭=*married*, ○|○=*divorced*, ○–○=*unmarried partnership*, ⁘=*floruit*, †=*died*, ⚔=*killed in action*, ⚱=*buried*, ⚱=*cremated*, ♀=*female*, ♂=*male*, ⚲=*neuter*.

german

```
\gtrset{language=german}
% ...
\gtrSymbolsFullLegend
```

★=*geboren*, (★)=*außerehelich geboren*, †★=*tot geboren*, ★†=*am Tag der Geburt gestorben*, ≈=*getauft*, ○=*verlobt*, ⚭=*verheiratet*, ○|○=*geschieden*, ○–○=*außerehelich verbunden*, ⁘=*blühte*, †=*gestorben*, ⚔=*gefallen*, ⚰=*begraben*, ⚱=*eingeäschert*, ♀=*weiblich*, ♂=*männlich*, ⚲=*Geschlecht unbekannt*.

/gtr/symlang/Born=⟨*text*⟩ (no default, initially `born`)
Legend ⟨*text*⟩ used for ★.

/gtr/symlang/Bornoutofwedlock=⟨*text*⟩ (no default, initially `born out of wedlock`)
Legend ⟨*text*⟩ used for (★).

/gtr/symlang/Stillborn=⟨*text*⟩ (no default, initially `stillborn`)
Legend ⟨*text*⟩ used for †★.

/gtr/symlang/Diedonbirthday=⟨*text*⟩ (no default, initially `died on the birthday`)
Legend ⟨*text*⟩ used for ★†.

/gtr/symlang/Baptized=⟨*text*⟩ (no default, initially `baptized`)
Legend ⟨*text*⟩ used for ≈.

/gtr/symlang/Engaged=⟨*text*⟩ (no default, initially `engaged`)
Legend ⟨*text*⟩ used for ○.

/gtr/symlang/Married=⟨*text*⟩ (no default, initially `married`)
Legend ⟨*text*⟩ used for ⚭.

/gtr/symlang/Divorced=⟨*text*⟩ (no default, initially `divorced`)
Legend ⟨*text*⟩ used for ○|○.

/gtr/symlang/Partnership=⟨*text*⟩ (no default, initially `unmarried partnership`)
Legend ⟨*text*⟩ used for ○–○.

/gtr/symlang/Floruit=⟨*text*⟩ (no default, initially `floruit`)
Legend ⟨*text*⟩ used for ⁘.

/gtr/symlang/Died=⟨*text*⟩ (no default, initially `died`)
Legend ⟨*text*⟩ used for †.

/gtr/symlang/Killed=⟨*text*⟩ (no default, initially `killed in action`)
Legend ⟨*text*⟩ used for ⚔.

/gtr/symlang/Buried=⟨*text*⟩ (no default, initially `buried`)
Legend ⟨*text*⟩ used for ⚰.

/gtr/symlang/Funeralurn=⟨*text*⟩ (no default, initially `cremated`)
Legend ⟨*text*⟩ used for ⚱.

/gtr/symlang/Female=⟨*text*⟩ (no default, initially `female`)
Legend ⟨*text*⟩ used for ♀.

/gtr/symlang/Male=⟨*text*⟩ (no default, initially `male`)
Legend ⟨*text*⟩ used for ♂.

/gtr/symlang/Neuter=⟨*text*⟩ (no default, initially `neuter`)
Legend ⟨*text*⟩ used for ⚲.

10

Language and Text Settings

The document `genealogytree-languages.pdf` displays the effects of language specific settings.

10.1 Preamble Settings

U 2017-07-19 **/gtr/language=⟨*language*⟩** (no default, initially `english`)

Sets the ⟨*language*⟩ for the description texts of the package. Typically, this option should be used inside the preamble, but it may also be used inside the document to switch between languages.

- If this option is used inside the preamble, the corresponding language library is loaded automatically.
- If this option is used inside the document, the corresponding language library has to be loaded separately inside the preamble by `\gtrloadlanguage` → P. 226.
- If this option is not used at all, the `english` language is set.

Feasible values for ⟨*language*⟩ are:

- `danish` (Translation provided by Mikkel Eide Eriksen)
- `dutch` (Translation provided by Dirk Bosmans)
- `english`
- `french` (Translation provided by Denis Bitouzé)
- `german` with variants:
  - `german-german`
  - `german-austrian`
- `italian` (Translation provided by Andrea Vaccari)

```
\documentclass{...}
%...
\gtrset{language=german-austrian}
%...
\begin{document}
%...
\end{document}
```

The current language name is stored inside `\gtrlanguagename`.

```
The current language is '\gtrlanguagename'.
```

The current language is 'english'.

The /gtr/language option sets various keys for description texts. These texts can be customized selectively, if needed.

```
\gtrset{language=german}
\gtrSymbolsRecordReset
\gtrsymBorn\ 1775, \gtrsymDied\ 1832.
\hfill(\gtrSymbolsLegend)

\gtrset{symlang/Born=geb.}
\gtrSymbolsRecordReset
\gtrsymBorn\ 1775, \gtrsymDied\ 1832.
\hfill(\gtrSymbolsLegend)

\gtrset{language=english}
\gtrSymbolsRecordReset
\gtrsymBorn\ 1775, \gtrsymDied\ 1832.
\hfill(\gtrSymbolsLegend)
```

★ 1775, † 1832. (★=geboren, †=gestorben.)
★ 1775, † 1832. (★=geb., †=gestorben.)
★ 1775, † 1832. (★=born, †=died.)

`\gtrloadlanguage{`⟨*list of languages*⟩`}`

Loads a comma separated ⟨*list of languages*⟩. This has to be given inside the preamble, if more than one language should be used in the document. Every loaded language can be used by /gtr/language → P. 225 inside the document. For a list of feasible language names, see /gtr/language → P. 225.

```
\documentclass{...}
%...
\gtrloadlanguage{english,german}
%...
\begin{document}
%...
\end{document}
```

10.2 Document Settings

Switching between languages inside the document is done by setting /gtr/language → P. 225. Note that every language to be used has to be loaded inside the preamble by \gtrloadlanguage.

11

Debugging: Library debug

The library is loaded by a package option or inside the preamble by:

```
\gtruselibrary{debug}
```

This also loads the packages `array` and `tabularx` and the `breakable` library of `tcolorbox`

11.1 Parser Debugging

The debugger for the parser can be used to check a manually or automatically generated tree source code to be well-formed. In this context, well-formedness means correct (LaTeX) grouping and correct nesting with subgraph elements following the given graph grammar, see Chapter 4. It is not checked, if all mandatory graph elements are present or if too many elements are given.

Also, the debugger gives a formal structured view of the given data which is useful to search for input errors if the graphical representation fails.

`\gtrparserdebug[⟨options⟩]{⟨graph content⟩}`

Parses the given ⟨*graph content*⟩. If the content is well-formed, a structured list of the given data is produced. The families are automatically colored in the list. Any ⟨*options*⟩ are checked by setting them and they are logged in the produced list.

```
\gtrparserdebug{
  parent{%
    c[id=pB]{B\\(child)}%
    g[id=pA]{A\\(proband)}%
    c[id=pC]{C\\(child)}%
    c[id=pD]{D\\(child)}%
    p[id=pE]{E\\(parent)}%
    p[id=pF]{F\\(parent)}%
  }
}
```

Genealogytree Parser Debugger

Start: Parent Family 1, Level 1

c
Child: Individual 1, Family 1, Level 0
Options: `id=pB`
Content: `B\\(child)`

g
Child: Individual 2, Family 1, Level 0
Options: `id=pA`
Content: `A\\(proband)`

c
Child: Individual 3, Family 1, Level 0
Options: `id=pC`
Content: `C\\(child)`

c
Child: Individual 4, Family 1, Level 0
Options: `id=pD`
Content: `D\\(child)`

p
Parent: Individual 5, Family 1, Level 1
Options: `id=pE`
Content: `E\\(parent)`

p
Parent: Individual 6, Family 1, Level 1
Options: `id=pF`
Content: `F\\(parent)`

End: Parent Family 1, Level 1

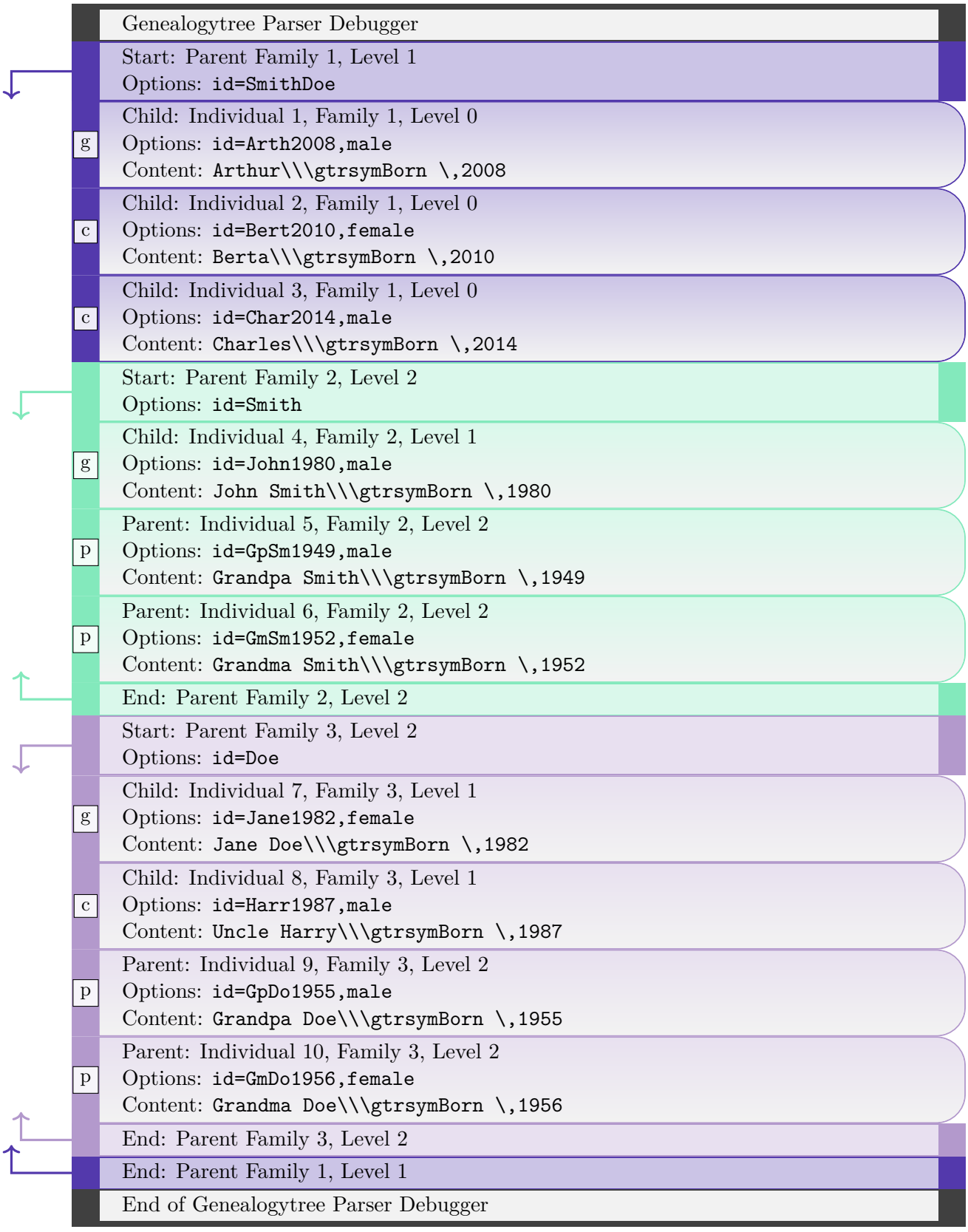
End of Genealogytree Parser Debugger

`\gtrparserdebuginput[⟨options⟩]{⟨file name⟩}`

Loads the file denoted by ⟨*file name*⟩ and parses its content. If the content is well-formed, a structured list of the given data is produced. The families are automatically colored in the list. Any ⟨*options*⟩ are checked by setting them and they are logged in the list.

The following example uses the graph from Section 14.1 on page 287.

```
\gtrparserdebuginput{example.option.graph}
```



11.2 Processor Debugging

`\gtrprocessordebug[`⟨*options*⟩`]{`⟨*graph content*⟩`}`

Processes the given ⟨*graph content*⟩. If the content can be processed without error, a structured list of the processed data is produced. The families are automatically colored in the list. Any ⟨*options*⟩ are set for processing.

```
\gtrprocessordebug{
  parent{%
    c[id=pB]{B\\(child)}%
    g[id=pA]{A\\(proband)}%
    c[id=pC]{C\\(child)}%
    c[id=pD]{D\\(child)}%
    p[id=pE]{E\\(parent)}%
    p[id=pF]{F\\(parent)}%
  }
}
```

Genealogytree Processor Debugger

Family 1

type:	par	*type of family*
id:	⟨none⟩	*identifier*
fam:	⟨none⟩	*enclosing family*
offset:	0.0pt	*x (or y) offset relative to enclosing family*
pos:	7.11319pt	*y (or x) absolute position*
cwest@anchor:	1	*west contour starting node*
ceast@anchor:	4	*east contour starting node*
g:	2	*g-node of the family*
par:	5, 6	*parent nodes*
chi:	1, 2, 3, 4	*child nodes*
patpar:	5, 6	*patchwork parent nodes*
patchi:	1, 2, 3, 4	*patchwork child nodes*
union:	⟨none⟩	*further partner families*
ps:	0pt	*pivot shift length (parents vs childs)*
x:	0.0pt	*x anchor*
y:	⟨none⟩	*y anchor*
frac:	0.5	*line positioning fraction*
opt@family:	⟨none⟩	*options for the family*
opt@subtree:	⟨none⟩	*options for the subtree*

Parents of Family 1

Person 5

id:	pE	*identifier (also node alias)*
fam:	1	*enclosing family*
chiof:	⟨none⟩	*child of family*
parof:	1	*parent of family*
x:	0.0pt	*x anchor*
y:	113.811pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	0.0pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	71.13188pt	*east contour value*
ceast@next:	⟨none⟩	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

E
(parent)

Person 6

id:	pF	*identifier (also node alias)*
fam:	1	*enclosing family*
chiof:	⟨none⟩	*child of family*
parof:	1	*parent of family*
x:	76.82242pt	*x anchor*
y:	113.811pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	76.82242pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	147.9543pt	*east contour value*
ceast@next:	⟨none⟩	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

F
(parent)

Childs of Family 1

Person 1

id:	pB	*identifier (also node alias)*
fam:	1	*enclosing family*
chiof:	1	*child of family*
parof:	⟨none⟩	*parent of family*
x:	-72.5545pt	*x anchor*
y:	0.0pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	-72.5545pt	*west contour value*
cwest@next:	5	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	-1.42262pt	*east contour value*
ceast@next:	6	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

B
(child)

Person 2

id:	pA	*identifier (also node alias)*
fam:	1	*enclosing family*
chiof:	1	*child of family*
parof:	⟨none⟩	*parent of family*
x:	1.42264pt	*x anchor*
y:	0.0pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	1.42264pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	72.55452pt	*east contour value*
ceast@next:	6	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

A
(proband)

Person 3

id:	pC	*identifier (also node alias)*
fam:	1	*enclosing family*
chiof:	1	*child of family*
parof:	⟨none⟩	*parent of family*
x:	75.39978pt	*x anchor*
y:	0.0pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	75.39978pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	146.53166pt	*east contour value*
ceast@next:	6	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

C
(child)

Person 4

id:	pD	*identifier (also node alias)*
fam:	1	*enclosing family*
chiof:	1	*child of family*
parof:	⟨none⟩	*parent of family*
x:	149.37692pt	*x anchor*
y:	0.0pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	149.37692pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	220.5088pt	*east contour value*
ceast@next:	6	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

D
(child)

End of Genealogytree Processor Debugger

`\gtrprocessordebuginput[⟨options⟩]{⟨file name⟩}`

Loads the file denoted by ⟨*file name*⟩ and processes its content. If the content can be processed without error, a structured list of the processed data is produced. The families are automatically colored in the list. Any ⟨*options*⟩ are set for processing.

The following example uses the graph from Section 14.1 on page 287.

```
\gtrprocessordebuginput{example.option.graph}
```

Genealogytree Processor Debugger

Family 1

type:	par	*type of family*
id:	SmithDoe	*identifier*
fam:	⟨none⟩	*enclosing family*
offset:	0.0pt	*x (or y) offset relative to enclosing family*
pos:	7.11319pt	*y (or x) absolute position*
cwest@anchor:	1	*west contour starting node*
ceast@anchor:	3	*east contour starting node*
g:	1	*g-node of the family*
par:	4, 7	*parent nodes*
chi:	1, 2, 3	*child nodes*
patpar:	4, 7	*patchwork parent nodes*
patchi:	1, 2, 3	*patchwork child nodes*
union:	⟨none⟩	*further partner families*
ps:	0pt	*pivot shift length (parents vs childs)*
x:	0.0pt	*x anchor*
y:	⟨none⟩	*y anchor*
frac:	0.5	*line positioning fraction*
opt@family:	⟨none⟩	*options for the family*
opt@subtree:	⟨none⟩	*options for the subtree*

Parents of Family 1

Person 4

id:	John1980	*identifier (also node alias)*
fam:	2	*enclosing family*
chiof:	2	*child of family*
parof:	1	*parent of family*
x:	38.41121pt	*x anchor*
y:	113.811pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	38.41121pt	*west contour value*
cwest@next:	5	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	109.54309pt	*east contour value*
ceast@next:	6	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

John Smith
★ 1980

Person 7

id:	Jane1982	*identifier (also node alias)*
fam:	3	*enclosing family*
chiof:	3	*child of family*
parof:	1	*parent of family*
x:	157.91275pt	*x anchor*
y:	113.811pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	1.42264pt	*west contour value*
cwest@next:	9	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	72.55452pt	*east contour value*
ceast@next:	10	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

Jane Doe
★ 1982

Childs of Family 1

Person 1

id:	Arth2008	*identifier (also node alias)*
fam:	1	*enclosing family*
chiof:	1	*child of family*
parof:	⟨none⟩	*parent of family*
x:	24.18484pt	*x anchor*
y:	0.0pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	24.18484pt	*west contour value*
cwest@next:	4	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	95.31673pt	*east contour value*
ceast@next:	8	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

Arthur
★ 2008

Person 2

id:	Bert2010	*identifier (also node alias)*
fam:	1	*enclosing family*
chiof:	1	*child of family*
parof:	⟨none⟩	*parent of family*
x:	98.16199pt	*x anchor*
y:	0.0pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	98.16199pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	169.29387pt	*east contour value*
ceast@next:	8	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

Berta
★ 2010

Person 3

id:	Char2014	*identifier (also node alias)*
fam:	1	*enclosing family*
chiof:	1	*child of family*
parof:	⟨none⟩	*parent of family*
x:	172.13913pt	*x anchor*
y:	0.0pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	172.13913pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	243.27101pt	*east contour value*
ceast@next:	8	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

Charles
★ 2014

Family 2

type:	par	*type of family*
id:	Smith	*identifier*
fam:	1	*enclosing family*
offset:	0pt	*x (or y) offset relative to enclosing family*
pos:	120.9242pt	*y (or x) absolute position*
cwest@anchor:	4	*west contour starting node*
ceast@anchor:	4	*east contour starting node*
g:	4	*g-node of the family*
par:	5, 6	*parent nodes*
chi:	4	*child nodes*
patpar:	5, 6	*patchwork parent nodes*
patchi:	4	*patchwork child nodes*
union:	⟨none⟩	*further partner families*
ps:	0pt	*pivot shift length (parents vs childs)*
x:	0.0pt	*x anchor*
y:	⟨none⟩	*y anchor*
frac:	0.5	*line positioning fraction*
opt@family:	⟨none⟩	*options for the family*
opt@subtree:	⟨none⟩	*options for the subtree*

Parents of Family 2

Person 5

id:	GpSm1949	*identifier (also node alias)*
fam:	2	*enclosing family*
chiof:	⟨none⟩	*child of family*
parof:	2	*parent of family*
x:	0.0pt	*x anchor*
y:	227.62201pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	0.0pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	71.13188pt	*east contour value*
ceast@next:	⟨none⟩	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

Grandpa Smith
★ 1949

Person 6

id:	GmSm1952	*identifier (also node alias)*
fam:	2	*enclosing family*
chiof:	⟨none⟩	*child of family*
parof:	2	*parent of family*
x:	76.82242pt	*x anchor*
y:	227.62201pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	76.82242pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	147.9543pt	*east contour value*
ceast@next:	⟨none⟩	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

Grandma Smith
★ 1952

Childs of Family 2

Person 4

id:	John1980	*identifier (also node alias)*
fam:	2	*enclosing family*
chiof:	2	*child of family*
parof:	1	*parent of family*
x:	38.41121pt	*x anchor*
y:	113.811pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	38.41121pt	*west contour value*
cwest@next:	5	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	109.54309pt	*east contour value*
ceast@next:	6	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

John Smith
★ 1980

Family 3

type:	par	*type of family*
id:	Doe	*identifier*
fam:	1	*enclosing family*
offset:	156.49011pt	*x (or y) offset relative to enclosing family*
pos:	120.9242pt	*y (or x) absolute position*
cwest@anchor:	7	*west contour starting node*
ceast@anchor:	8	*east contour starting node*
g:	7	*g-node of the family*
par:	9, 10	*parent nodes*
chi:	7, 8	*child nodes*
patpar:	9, 10	*patchwork parent nodes*
patchi:	7, 8	*patchwork child nodes*
union:	⟨none⟩	*further partner families*
ps:	0pt	*pivot shift length (parents vs childs)*
x:	156.49011pt	*x anchor*
y:	⟨none⟩	*y anchor*
frac:	0.5	*line positioning fraction*
opt@family:	⟨none⟩	*options for the family*
opt@subtree:	⟨none⟩	*options for the subtree*

Parents of Family 3

Person 9

id:	GpDo1955	*identifier (also node alias)*
fam:	3	*enclosing family*
chiof:	⟨none⟩	*child of family*
parof:	3	*parent of family*
x:	156.49011pt	*x anchor*
y:	227.62201pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	0.0pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	71.13188pt	*east contour value*
ceast@next:	⟨none⟩	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

Grandpa Doe
★ 1955

Person 10

id:	GmDo1956	*identifier (also node alias)*
fam:	3	*enclosing family*
chiof:	⟨none⟩	*child of family*
parof:	3	*parent of family*
x:	233.31253pt	*x anchor*
y:	227.62201pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	76.82242pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	147.9543pt	*east contour value*
ceast@next:	⟨none⟩	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

Grandma Doe
★ 1956

Childs of Family 3

Person 7

id:	Jane1982	*identifier (also node alias)*
fam:	3	*enclosing family*
chiof:	3	*child of family*
parof:	1	*parent of family*
x:	157.91275pt	*x anchor*
y:	113.811pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	1.42264pt	*west contour value*
cwest@next:	9	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	72.55452pt	*east contour value*
ceast@next:	10	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

Jane Doe
★ 1982

Person 8

id:	Harr1987	*identifier (also node alias)*
fam:	3	*enclosing family*
chiof:	3	*child of family*
parof:	⟨none⟩	*parent of family*
x:	231.8899pt	*x anchor*
y:	113.811pt	*y anchor*
dim:	71.13188pt	*width (or height)*
cwest@val:	75.39978pt	*west contour value*
cwest@next:	⟨none⟩	*west contour successor*
cwest@thread:	⟨none⟩	*west contour thread*
cwest@tgap:	⟨none⟩	*west contour thread gap*
ceast@val:	146.53166pt	*east contour value*
ceast@next:	10	*east contour successor*
ceast@thread:	⟨none⟩	*east contour thread*
ceast@tgap:	⟨none⟩	*east contour thread gap*

Uncle Harry
★ 1987

End of Genealogytree Processor Debugger

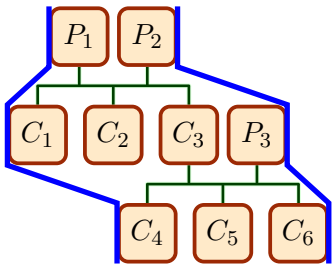
11.3 Graphical Debugging

`\gtrdebugdrawcontour{`⟨*options*⟩`}{`⟨*path options*⟩`}`

After a `\genealogytree` → P. 55 is drawn inside a `tikzpicture` environment, the auto-layout contour lines of a family can be displayed with this macro. For ⟨*options*⟩, the keys `/gtr/debug/family number` → P. 240, `/gtr/debug/family id` → P. 240, `/gtr/debug/contour` → P. 241 may be used to specify the family and the contour lines to draw. The ⟨*path options*⟩ are used to draw a TikZ path.

- Contour lines for the root family should always be displayed correctly.
- Contour lines for embedded families may be displayed prolonged, because these are used to build the contour lines of their embedding families. Note that `\gtrdebugdrawcontour` shows the remains of the building process, but not the dynamics of the process.
- Contour lines for `union` families are not displayed, since they are melted to their embedding `child` family.

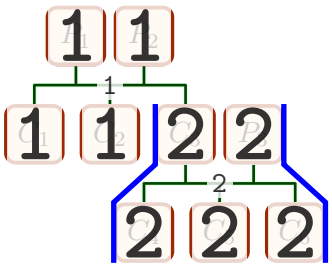
```
\begin{tikzpicture}
\genealogytree[template=formal graph]
{
  child{
    g{P_1}  p{P_2}  c{C_1}  c{C_2}
    child{
      g{C_3}  p{P_3}  c{C_4}  c{C_5}  c{C_6}
    }
  }
}
\gtrdebugdrawcontour{}{draw=blue,line width=2pt}
\end{tikzpicture}
```



`/gtr/debug/family number=`⟨*number*⟩ (no default, initially 1)

Selects a family by ⟨*number*⟩ inside the option list of `\gtrdebugdrawcontour` → P. 239.

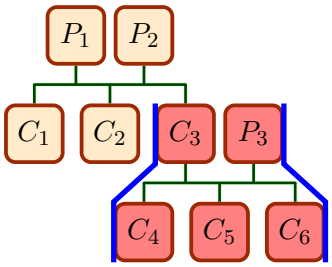
```
\begin{tikzpicture}
\genealogytree[template=formal graph,show family]
{
  child{
    g{P_1}  p{P_2}  c{C_1}  c{C_2}
    child{
      g{C_3}  p{P_3}  c{C_4}  c{C_5}  c{C_6}
    }
  }
}
\gtrdebugdrawcontour{family number=2}{draw=blue,line width=2pt}
\end{tikzpicture}
```



`/gtr/debug/family id=`⟨*id*⟩ (no default, initially unset)

Selects a family by ⟨*id*⟩ inside the option list of `\gtrdebugdrawcontour` → P. 239.

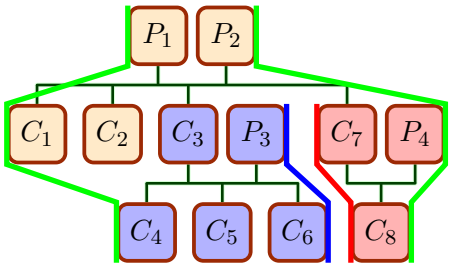
```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  options for family={fam_a}{box={colback=red!50}}]
{
  child{
    g{P_1}  p{P_2}  c{C_1}  c{C_2}
    child[id=fam_a]{
      g{C_3}  p{P_3}  c{C_4}  c{C_5}  c{C_6}
    }
  }
}
\gtrdebugdrawcontour{family id=fam_a}{draw=blue,line width=2pt}
\end{tikzpicture}
```



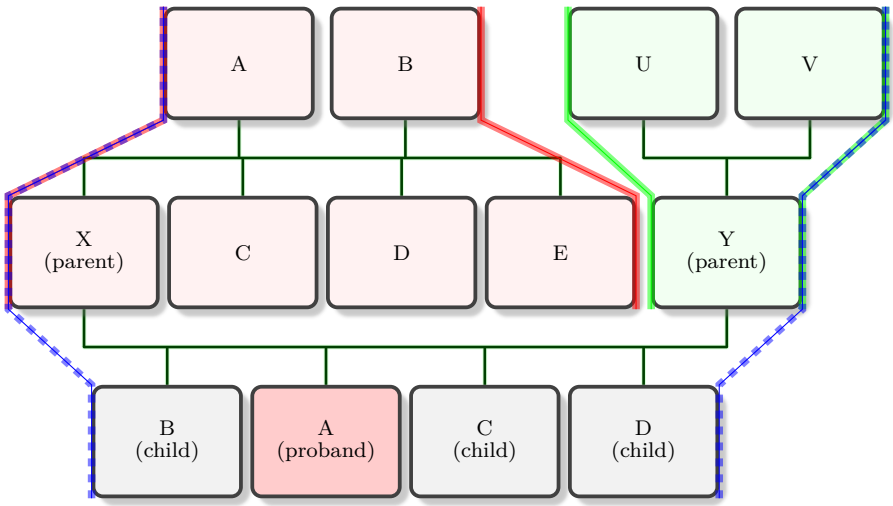
`/gtr/debug/contour=west|east|both|none` (no default, initially `both`)

The two contour lines are always referred to as `west` and `east` contour lines independent of the `/gtr/timeflow` → P. 78 setting. With this option, a partial contour drawing can be used.

```
\begin{tikzpicture}
\genealogytree[template=formal graph,
  options for family={fam_a}{box={colback=blue!30}},
  options for family={fam_b}{box={colback=red!30}},
  ]
{
  child{
    g{P_1}  p{P_2}  c{C_1}  c{C_2}
    child[id=fam_a]{
      g{C_3}  p{P_3}  c{C_4}  c{C_5}  c{C_6}
    }
    child[id=fam_b]{
      g{C_7}  p{P_4}  c{C_8}
    }
  }
}
\gtrdebugdrawcontour{}{draw=green,line width=2pt}
\gtrdebugdrawcontour{contour=east,family id=fam_a}{draw=blue,line width=2pt}
\gtrdebugdrawcontour{contour=west,family id=fam_b}{draw=red,line width=2pt}
\end{tikzpicture}
```



```
\begin{tikzpicture}
\genealogytree[template=signpost]{
  parent[id=myid]{
    c[id=pB]{B\\(child)}
    g[id=pA,box={colback=red!20!white}]{A\\(proband)}
    c[id=pC]{C\\(child)}
    c[id=pD]{D\\(child)}
    parent[id=partial,family={box={colback=red!5}}]{
      g[id=pX]{X\\(parent)}
      p{A}  p{B}  c{C}  c{D}  c{E}
    }
    parent[id=partial2,family={box={colback=green!5}}]{
      g[id=pY]{Y\\(parent)}
      p{U}  p{V}
    }
  }
}
\gtrdebugdrawcontour{family id=partial}
  {preaction={draw=red,line width=1mm,opacity=.5},draw=red,line width=0.4pt}
\gtrdebugdrawcontour{family id=partial2}
  {preaction={draw=green,line width=1mm,opacity=.5},draw=green,line width=0.4pt}
\gtrdebugdrawcontour{family id=myid}
  {preaction={draw=blue,line width=1mm,opacity=.5,dashed},draw=blue,line width=0.4pt}
\end{tikzpicture}
```



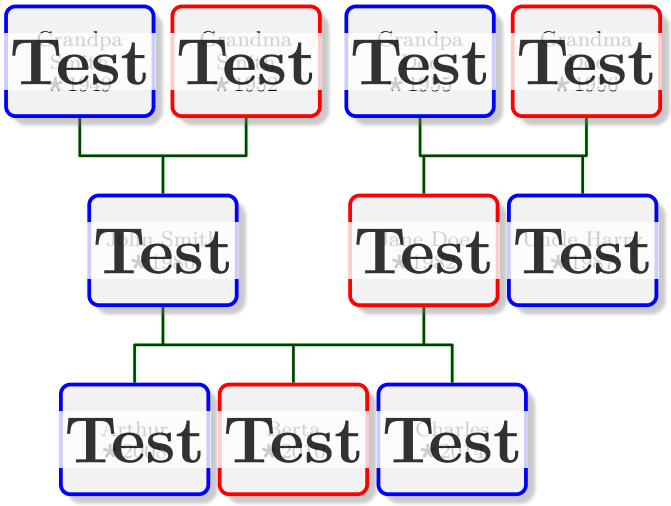
11.4 Show Information

Note that most options in this section only work, if a /gtr/processing → P. 128 based on a box from the tcolorbox package is chosen (this is the default setting).

/gtr/show=⟨*text*⟩ (style, no default)

Shows a ⟨*text*⟩ overlay for each node of the tree.

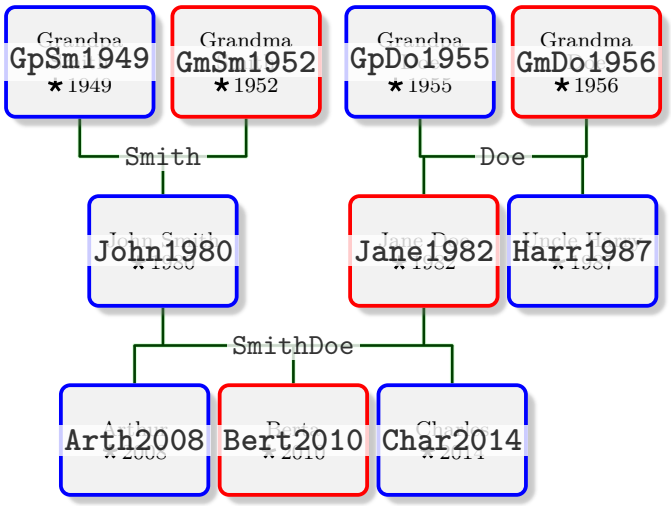
```
\begin{tikzpicture}
\genealogytree[template=signpost,show={Test}]
  {input{example.option.graph}}
\end{tikzpicture}
```



/gtr/show id (style, no value)

Shows the /gtr/id → P. 90 values of every node and every family. This can be very valuable not only for debugging, but also for visual identification of nodes to manipulate.

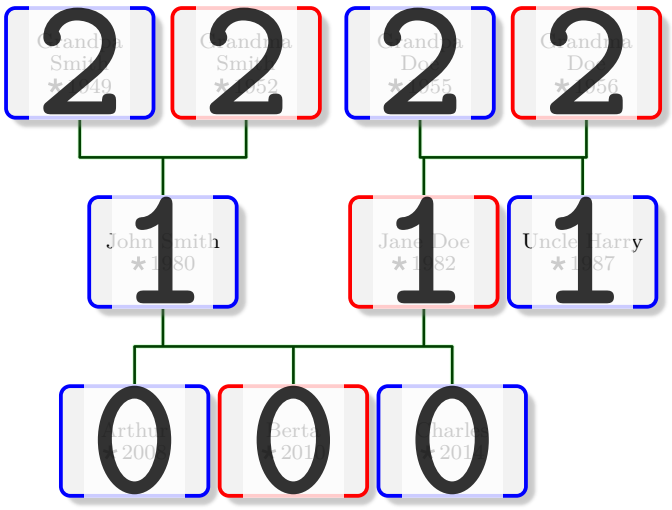
```
\begin{tikzpicture}
\genealogytree[template=signpost,show id]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/show level` (style, no value)

Shows the level numbers of every node. This information can be used for setting `/gtr/level` → P. 107 and `/gtr/level n` → P. 108.

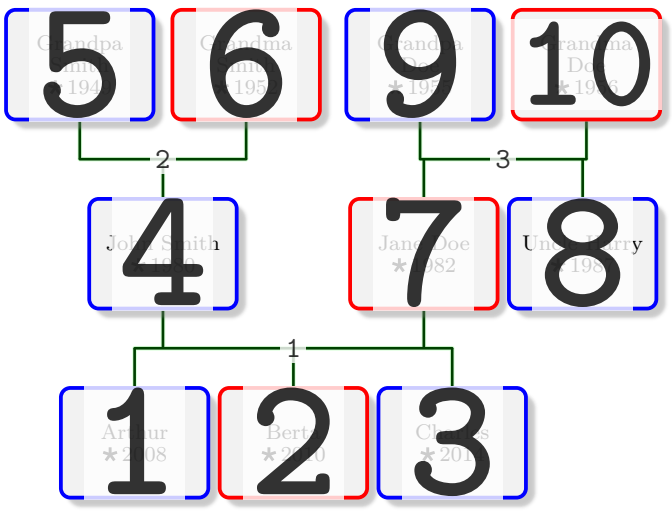
```
\begin{tikzpicture}
\genealogytree[template=signpost,show level]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/show number` (style, no value)

Shows the internal numbers of every node and every family. It is strongly recommended to reference a node by a chosen `/gtr/id` → P. 90 and not by its internal number, because numbers may easily change when editing the tree.

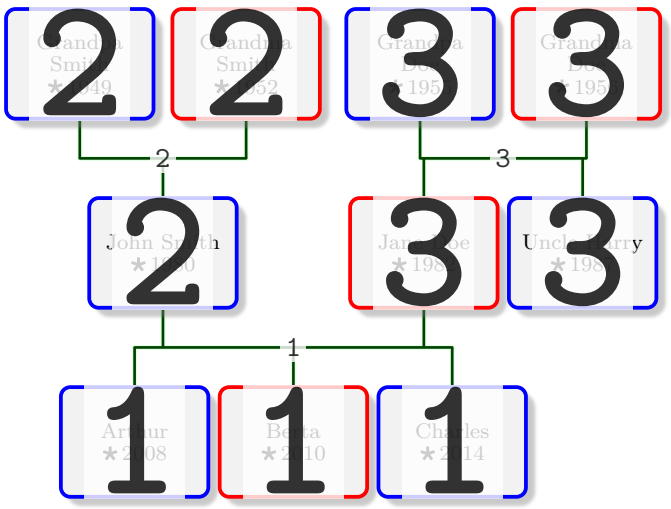
```
\begin{tikzpicture}
\genealogytree[template=signpost,show number]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/show family` (style, no value)

Shows the internal family numbers each node belongs to. A `g` node can be part of many families, but only one family is the *enclosing* family. For a `union` family, the family number is displayed, but the *enclosing* family is the family of the `g` node.

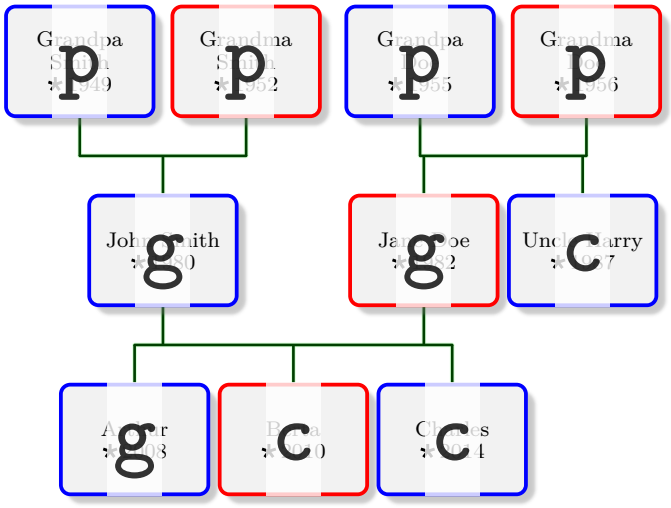
```
\begin{tikzpicture}
\genealogytree[template=signpost,show family]
  {input{example.option.graph}}
\end{tikzpicture}
```



`/gtr/show type` (style, no value)

Show the node type for every node.

```
\begin{tikzpicture}
\genealogytree[template=signpost,show type]
  {input{example.option.graph}}
\end{tikzpicture}
```



12

Templates: Library templates

The library is loaded by a package option or inside the preamble by:

```
\gtruselibrary{templates}
```

12.1 Using Templates

/gtr/template=⟨*name*⟩ (style, no default)

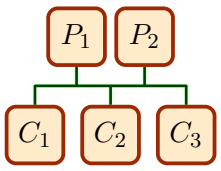
Sets a predefined style ⟨*name*⟩ for a genealogytree graph. A template does not provide new functionality, but combines various options for specific trees, e.g., used inside this documentation. It serves as a shortcut. If a template is used, it is recommended to apply it as very first option.

12.2 Template 'formal graph'

```
template=formal graph
```

This style is based on /gtr/processing→ P. 128=tcbox*. The box content is set as formula in mathematical mode. For further examples, see Section 5.3 on page 81.

```
\begin{tikzpicture}
\genealogytree[template=formal graph]{
  child{
    g{P_1}
    p{P_2}
    c{C_1}
    c{C_2}
    c{C_3}
  }
}
\end{tikzpicture}
```

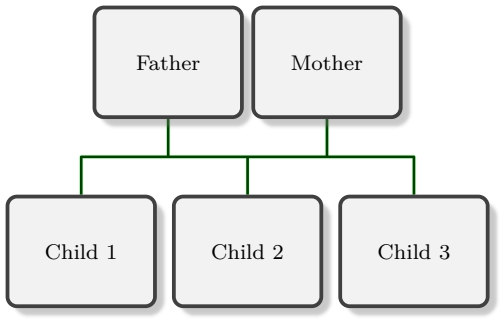


12.3 Template 'signpost'

```
template=signpost
```

This style is based on /gtr/processing → P. 128=fit. For further examples, see Section 5.2 on page 78 and many more.

```
\begin{tikzpicture}
\genealogytree[template=signpost]{
  child{
    g{Father}
    p{Mother}
    c{Child 1}
    c{Child 2}
    c{Child 3}
  }
}
\end{tikzpicture}
```

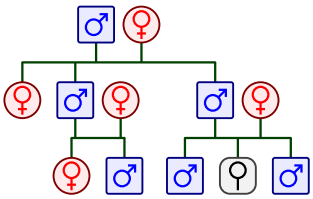


12.4 Template 'symbol nodes'

```
template=symbol nodes
```

This style is based on /gtr/processing → P. 128=tcbox*. For the content, a single token m selects a male node (also male), a single token f selects a female node (also female), and every other token selects a neuter node. The symbol coloring with \gtrSymbolsSetCreateSelected → P. 218 has to be done *before* entering a tikzpicture environment.

```
\gtrSymbolsSetCreateSelected{blue}{Male}
\gtrSymbolsSetCreateSelected{red}{Female}
\gtrSymbolsSetCreateSelected{black}{Neuter}
\begin{tikzpicture}
\genealogytree[template=symbol nodes]{
  child{
    gm pf cf
    child{gm pf cf cm}
    child{gm pf cm c- cm}
  }
}
\end{tikzpicture}
```

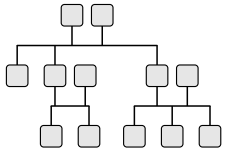


12.5 Template 'tiny boxes'

```
template=tiny boxes
```

This style is based on /gtr/processing→ P. 128=tcbox*. The content of all boxes is removed. Therefore, a single token like '-' is enough to declare the content. For further examples, see Chapter 13 on page 277.

```
\begin{tikzpicture}
\genealogytree[template=tiny boxes]{
  child{
    g-p-c-
    child{g-p-c-c-}
    child{g-p-c-c-c-}
  }
}
\end{tikzpicture}
```

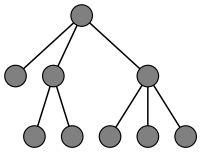


12.6 Template 'tiny circles'

```
template=tiny circles
```

This style is based on /gtr/processing→ P. 128=tcbox*. The content of all boxes is removed. Therefore, a single token like '-' is enough to declare the content. All distances are set equally and edges are drawn meshed. For further examples, see Chapter 13 on page 277.

```
\begin{tikzpicture}
\genealogytree[template=tiny circles]{
  child{
    g-c-
    child{g-c-c-}
    child{g-c-c-c-}
  }
}
\end{tikzpicture}
```

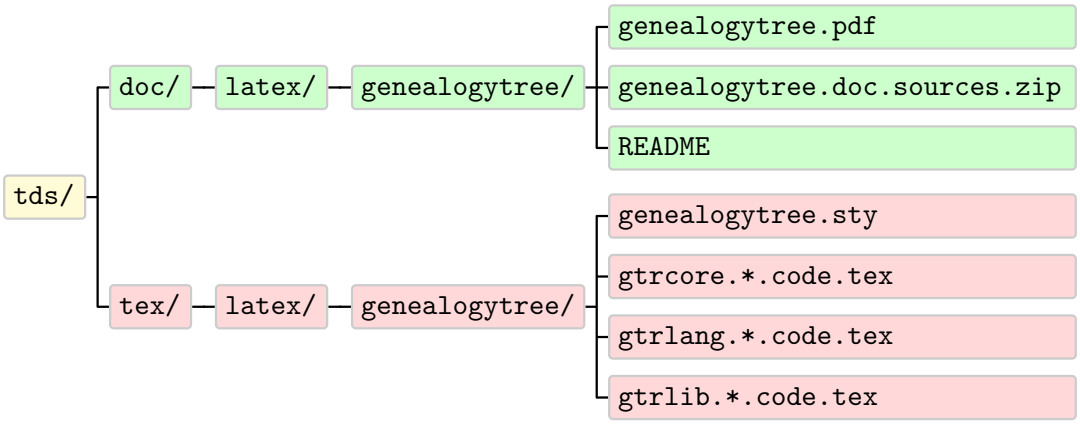


12.7 Template 'directory'

```
template=directory
```

This style is based on /gtr/processing→ P. 128=tcbox* and sets /gtr/timeflow→ P. 78=right. Note that optimal level sizes have to be set manually.

```
\begin{genealogypicture}[template=directory,
  level  0/.style={level size=11mm},
  level -1/.style={level size=11mm},
  level -2/.style={level size=15mm},
  level -3/.style={level size=31mm},
  level -4/.style={level size=62mm},
  ]
  child{  g{tds}
    child[subtree box={colback=green!20}]{  g{doc}
      child{  g{latex}
        child{  g{genealogytree}
          c{genealogytree.pdf}
          c{genealogytree.doc.sources.zip}
          c{README}
        }
      }
    }
    child[subtree box={colback=red!15}]{  g{tex}
      child{  g{latex}
        child{  g{genealogytree}
          c{genealogytree.sty}
          c{gtrcore.*.code.tex}
          c{gtrlang.*.code.tex}
          c{gtrlib.*.code.tex}
        }
      }
    }
  }
\end{genealogypicture}
```



12.8 Template 'database pole'

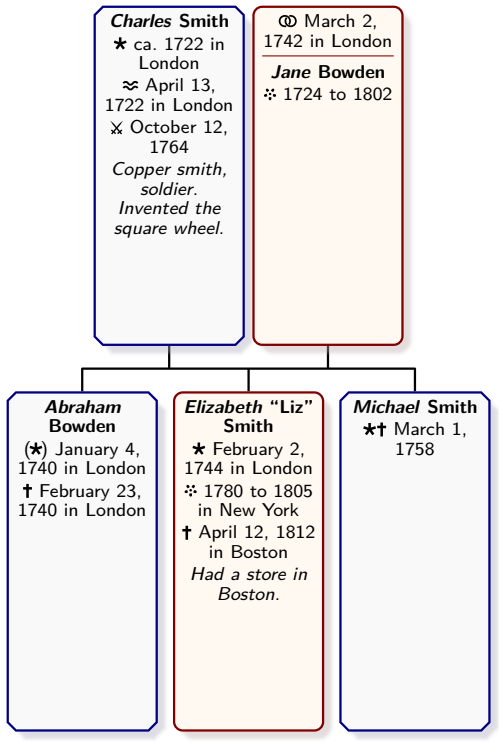
```
template=database pole
```

This style is based on /gtr/processing→ P. 128=database and sets /gtr/database format→ P. 162=full marriage above.

The boxes are quite small for placing many nodes horizontally. Also, many settings are adapted for this style.

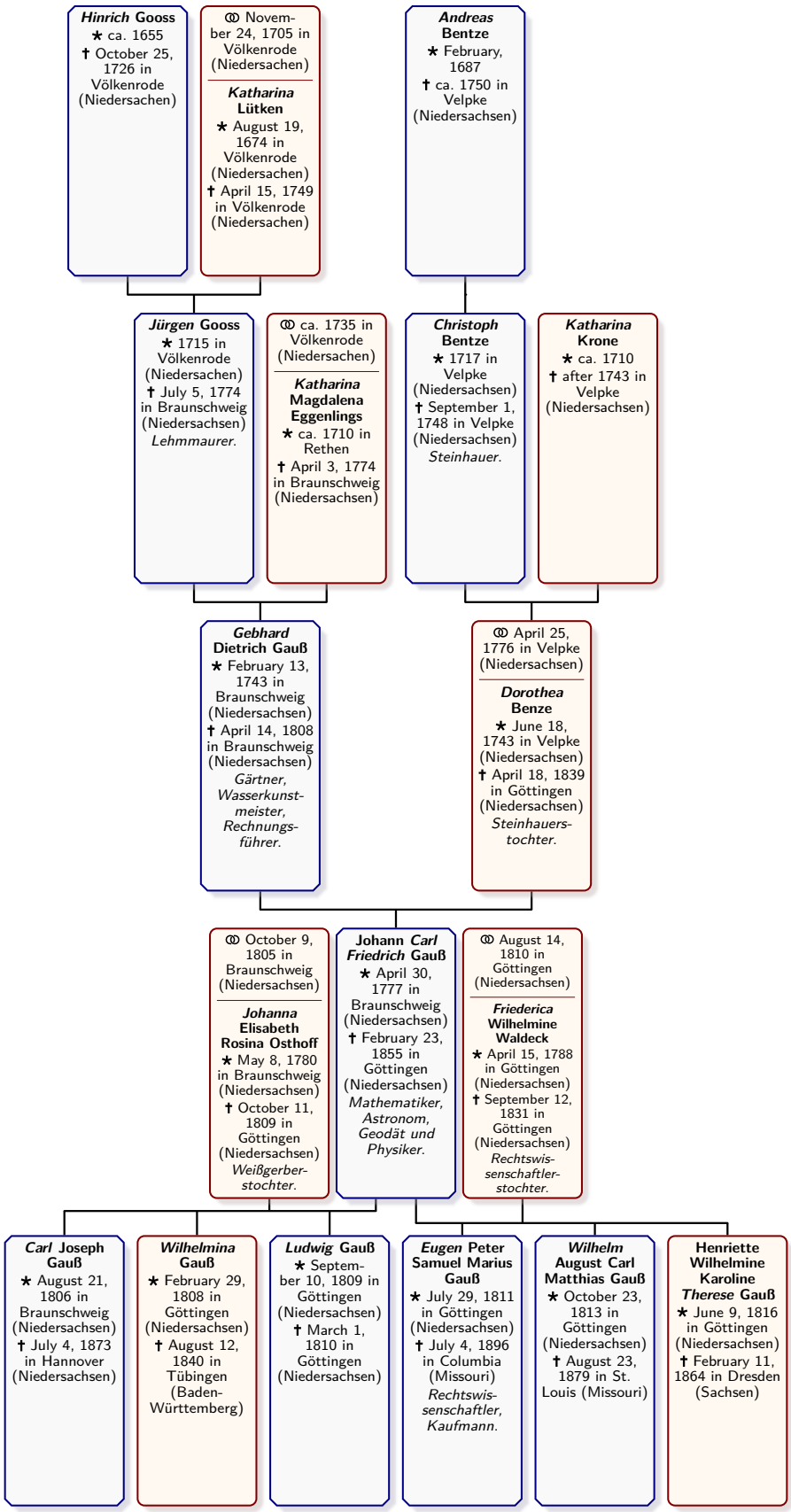
The following example uses a file documented in Section 14.2 on page 288.

```
\begin{genealogypicture}
  [template=database pole]
  input{example.database.graph}
\end{genealogypicture}
```



The next example uses the graph data from Section 2.3.5 on page 39.

```
\begin{genealogypicture}[template=database pole,tikzpicture={scale=0.9,transform shape}]
  input{example.gauss.graph}
\end{genealogypicture}
```



12.9 Template 'database pole reduced'

N 2017-09-15

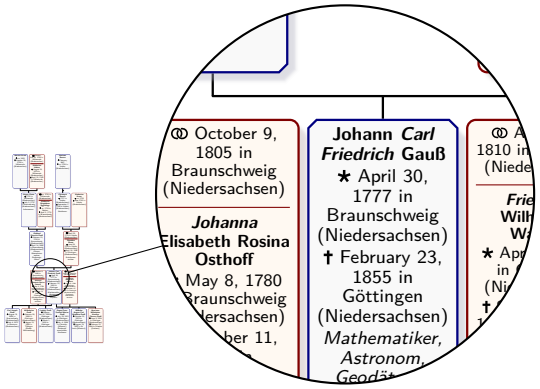
```
template=database pole reduced
```

This style is identical to **database pole** (Section 12.8 on page 251), but every size property is scaled by factor $\frac{1}{10}$. Therefore, the resulting graphs are only 10 percent of width and height of corresponding graphs made with **database pole**.

- TEX/LATEX length values are limited by about 575cm. Due to internal calculations, the maximum width and height of a graph may even be smaller. Using this *reduced* layout size, this limit is avoided to a certain degree.
- For virtual PDF 'paper' on a computer screen, the tiny layout irrelevant, since this vector format can be zoomed without loss.
- For printing, the PDF also can be zoomed or cut into several pages which can be zoomed.
- Note that a freely scalable text font is needed for the *reduced* layout size! The standard LATEX font is not scalable.
- Examples with more than 1500 nodes compiled successfully. Note that the compiler memory settings may have to be set to increased values for graphs with many nodes.

The following example uses the graph data from Section 2.3.5 on page 39.

```
\begin{tikzpicture}[
    spy using outlines={circle, magnification=10, size=5cm, connect spies}]
  \genealogytreeinput[template=database pole reduced]{example.gauss.graph}
  \spy on (0.6,-0.1) in node [left] at (7,1);
\end{tikzpicture}
```



12.10 Template 'database poleportrait'

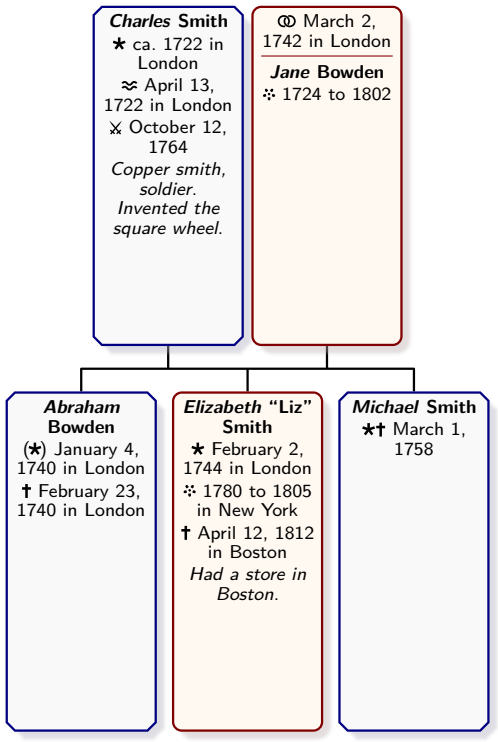
`template=database poleportrait`

N 2017-09-15

This style is based on `/gtr/processing` → P. 128`=database` and sets `/gtr/database format` → P. 162`=full marriage above`.

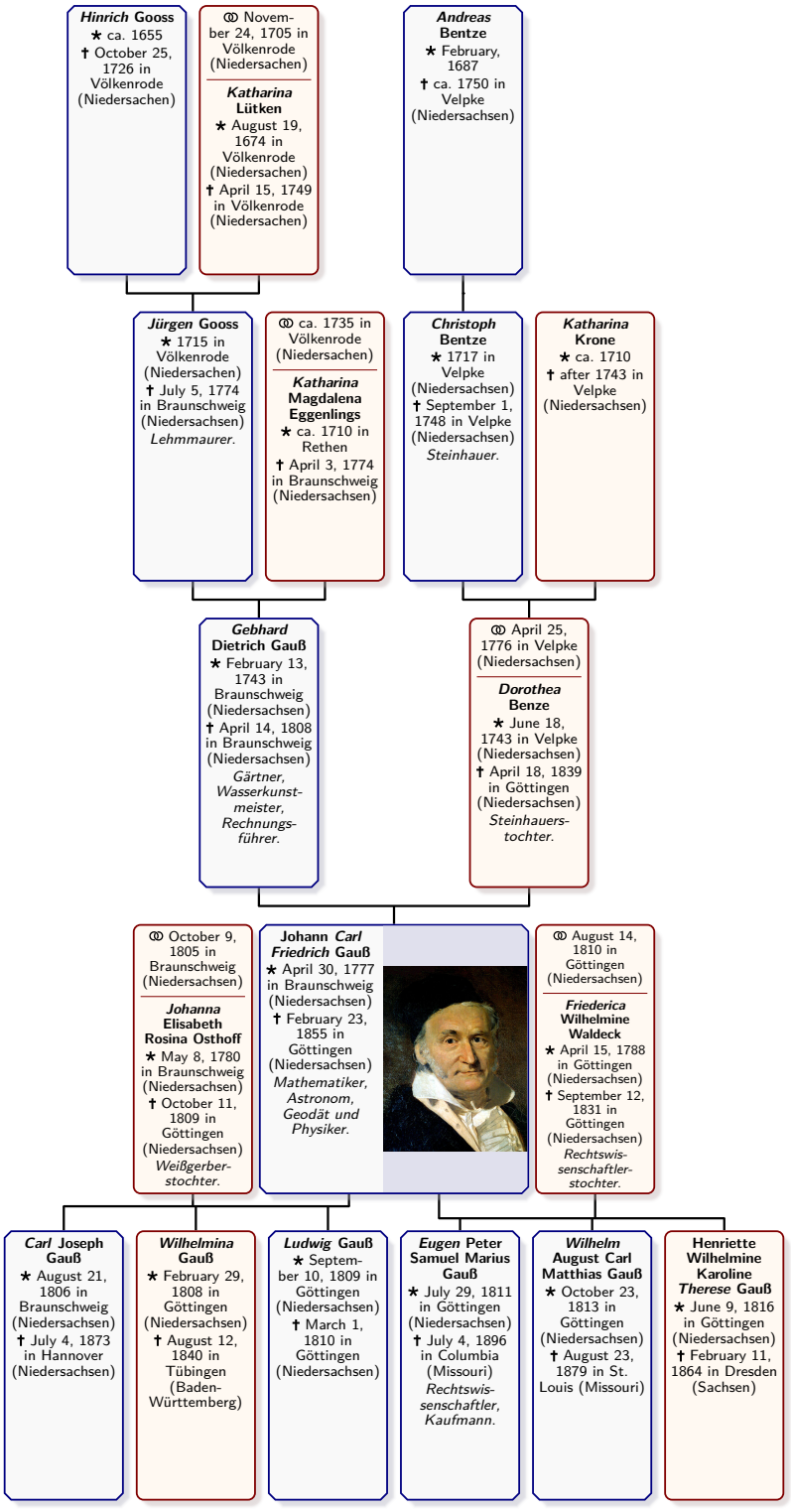
In contrast to `database pole` (Section 12.8 on page 251), portraits are drawn, if present.

```
\begin{genealogypicture}
  [template=database poleportrait]
  input{example.database.graph}
\end{genealogypicture}
```



The next example uses the graph data from Section 2.3.5 on page 39.

```
\begin{genealogypicture}[template=database poleportrait,
    tikzpicture={scale=0.8,transform shape}]
  input{example.gauss.graph}
\end{genealogypicture}
```



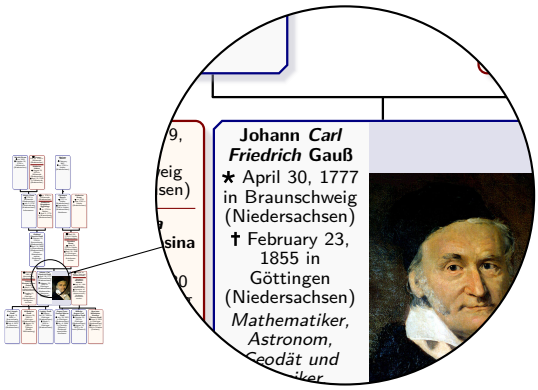
12.11 Template 'database poleportrait reduced'

`template=database poleportrait reduced` N 2017-09-15

This style is identical to **database poleportrait** (Section 12.10 on page 254), but every size property is scaled by factor $\frac{1}{10}$. See Section 12.9 on page 253 for more explanations.

The following example uses the graph data from Section 2.3.5 on page 39.

```
\begin{tikzpicture}[
    spy using outlines={circle, magnification=10, size=5cm, connect spies}]
  \genealogytreeinput[template=database poleportrait reduced]{example.gauss.graph}
  \spy on (0.6,-0.1) in node [left] at (7,1);
\end{tikzpicture}
```



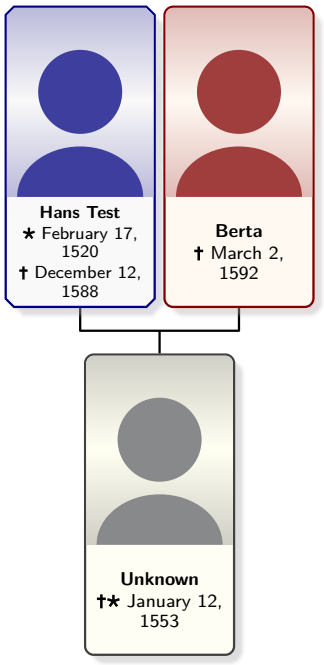
12.12 Template 'database portrait'

```
template=database portrait
```

This style is based on /gtr/processing →P. 128=database and sets /gtr/database format →P. 162=short no marriage.

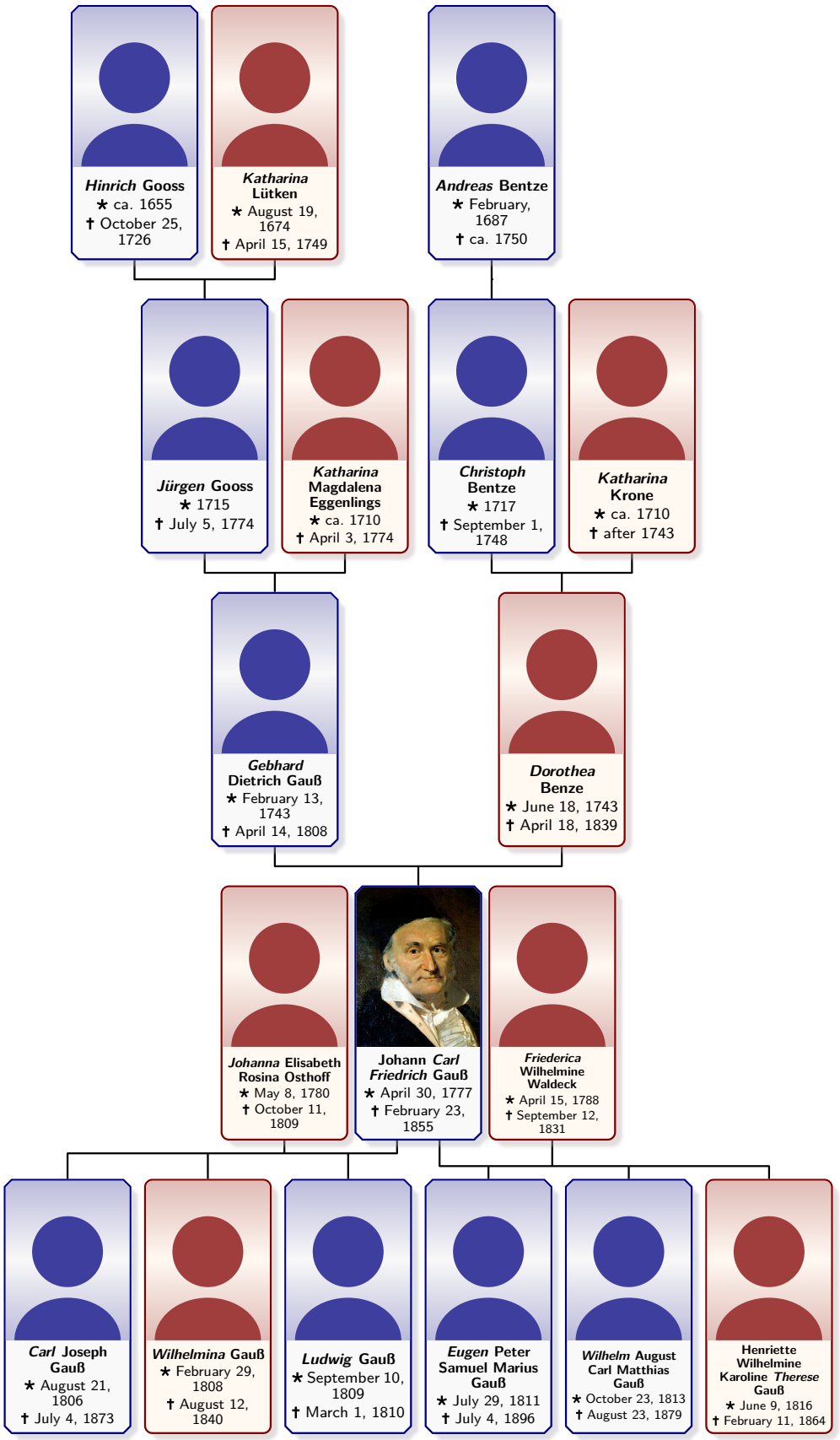
The boxes are quite small for placing many nodes horizontally. Also, many settings are adapted for this style. If /gtr/database/image →P. 156 is present, the corresponding image is inserted. Otherwise, a symbolic portrait is drawn.

```
\begin{genealogypicture}
  [template=database portrait]
  child{
    g{male,name=Hans Test,
      birth={1520-02-17}{Footown},
      death={1588-12-12}{Footown}}
    p{female,name=Berta,
      death={1592-03-02}{Footown}}
    c{name=Unknown,
      birth+={1553-01-12}{Footown}
        {stillborn}}
  }
\end{genealogypicture}
```



The next example uses the graph data from Section 2.3.5 on page 39.

```
\begin{genealogypicture}[template=database portrait]
  input{example.gauss.graph}
\end{genealogypicture}
```



12.13 Template 'database portrait reduced'

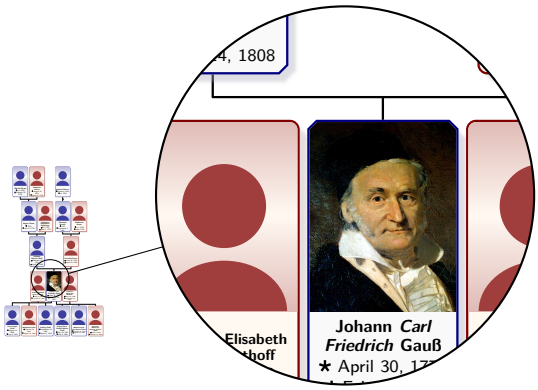
N 2017-09-15

```
template=database portrait reduced
```

This style is identical to `database portrait` (Section 12.12 on page 257), but every size property is scaled by factor $\frac{1}{10}$. See Section 12.9 on page 253 for more explanations.

The following example uses the graph data from Section 2.3.5 on page 39.

```
\begin{tikzpicture}[
    spy using outlines={circle, magnification=10, size=5cm, connect spies}]
  \genealogytreeinput[template=database portrait reduced]{example.gauss.graph}
  \spy on (0.6,-0.1) in node [left] at (7,1);
\end{tikzpicture}
```



12.14 Template 'database traditional'

```
template=database traditional
```

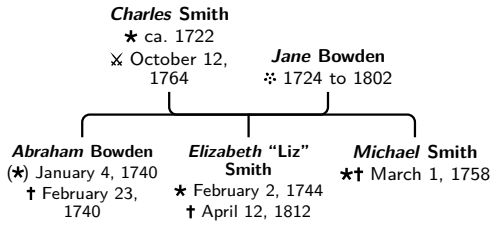
This style is based on /gtr/processing →P. 128=database, sets /gtr/database format →P. 162=short no marriage and /gtr/timeflow →P. 78=down.

Using this template, a sober black-and-white drawing with only short information is created. The box content is not framed.

- For p nodes, the content is bottom aligned.
- For c nodes, the content is top aligned.
- For g nodes, the content is center aligned. While this is usally reasonable, g nodes in families without childs or parents may have to be adapted manually. The root node is treated automatically.

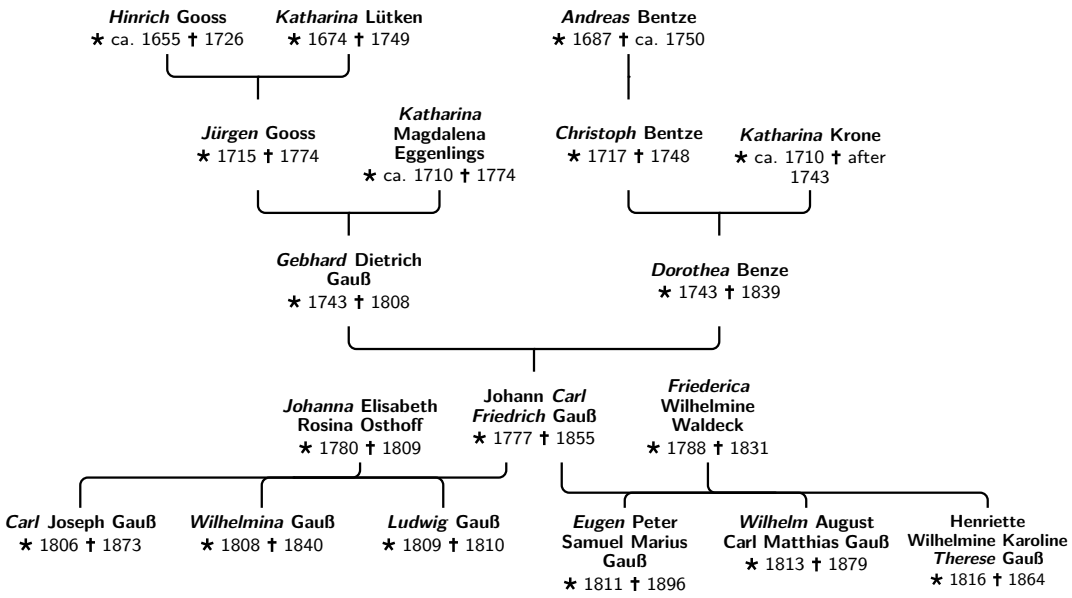
The following example uses a file documented in Section 14.2 on page 288.

```
\begin{genealogypicture}[
    template=database traditional,
    level size=1.3cm  ]
  input{example.database.graph}
\end{genealogypicture}
```



The next example uses the graph data from Section 2.3.5 on page 39.

```
\begin{genealogypicture}[template=database traditional,
  level size=1.2cm,node size=2.2cm,date format=yyyy,list separators={\par}{ }{}{}]
  input{example.gauss.graph}
\end{genealogypicture}
```



12.15 Template 'database traditional reduced'

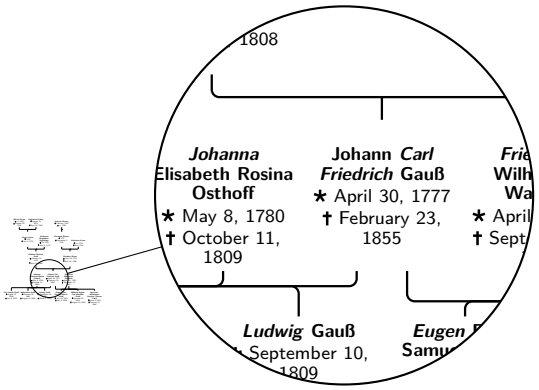
N 2017-09-15

```
template=database traditional reduced
```

This style is identical to `database traditional` (Section 12.14 on page 260), but every size property is scaled by factor $\frac{1}{10}$. See Section 12.9 on page 253 for more explanations.

The following example uses the graph data from Section 2.3.5 on page 39.

```
\begin{tikzpicture}[
    spy using outlines={circle, magnification=10, size=5cm, connect spies}]
  \genealogytreeinput[template=database traditional reduced]{example.gauss.graph}
  \spy on (0.6,-0.1) in node [left] at (7,1);
\end{tikzpicture}
```



12.16 Template 'database sideways'

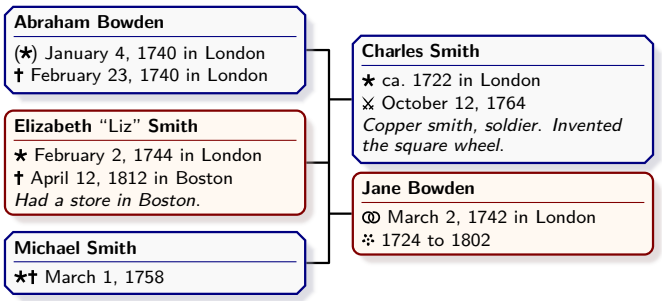
```
template=database sideways
```

This style is based on /gtr/processing→ P. 128=database, sets /gtr/database format→ P. 162=medium and /gtr/timeflow→ P. 78=left.

Here, the boxes are positioned sideways and have a large variety of height. Therefore, the content will seldom be resized.

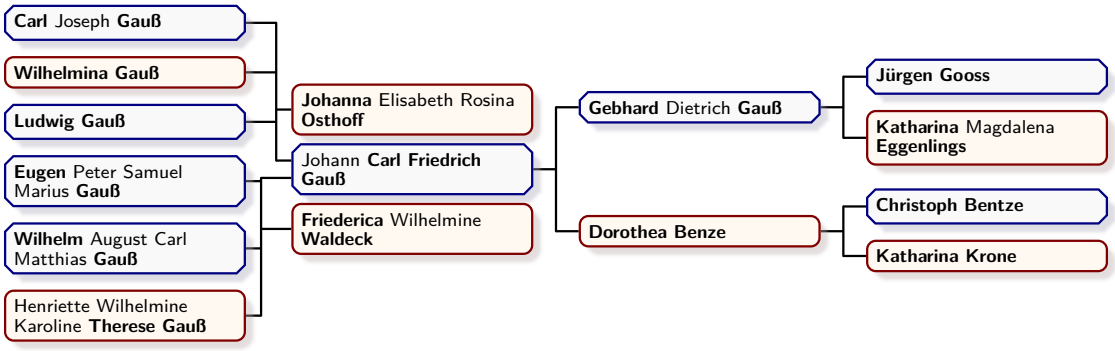
The following example uses a file documented in Section 14.2 on page 288.

```
\begin{genealogypicture}[template=database sideways]
  input{example.database.graph}
\end{genealogypicture}
```

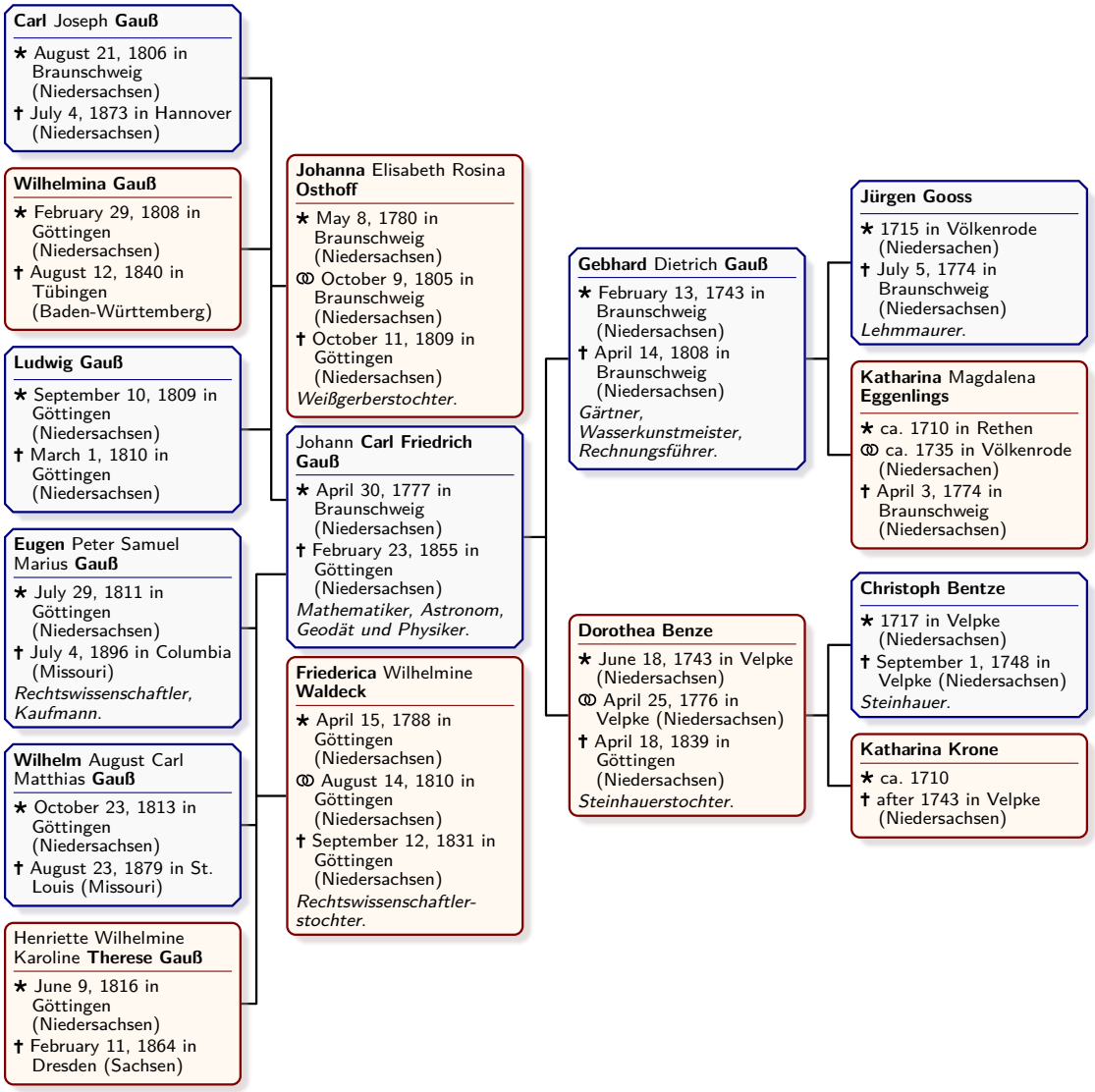


The next example uses the graph data from Section 2.3.5 on page 39.

```
\begin{genealogypicture}[template=database sideways,
  level size=3.2cm,ignore level=3,database format=name]
  input{example.gauss.graph}
\end{genealogypicture}
```



```
\begin{genealogypicture}[template=database sideways,
  level size=3.2cm,ignore level=3]
  input{example.gauss.graph}
\end{genealogypicture}
```



12.17 Template 'database sideways reduced'

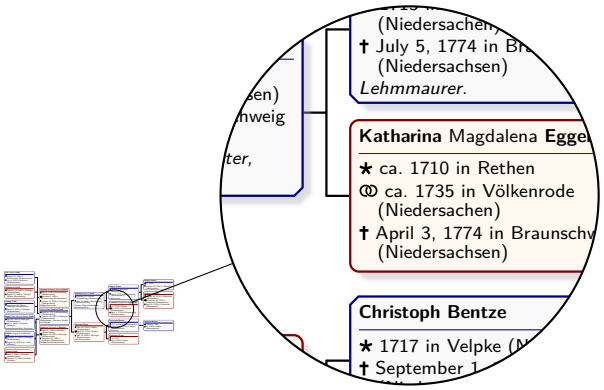
`template=database sideways reduced`

N 2017-09-15

This style is identical to **database sideways** (Section 12.16 on page 262), but every size property is scaled by factor $\frac{1}{10}$. See Section 12.9 on page 253 for more explanations.

The following example uses the graph data from Section 2.3.5 on page 39.

```
\begin{tikzpicture}[
    spy using outlines={circle, magnification=10, size=5cm, connect spies}]
  \genealogytreeinput[template=database sideways reduced]{example.gauss.graph}
  \spy on (0.6,-0.5) in node [left] at (7,1);
\end{tikzpicture}
```



12.18 Template 'database sidewaysportrait'

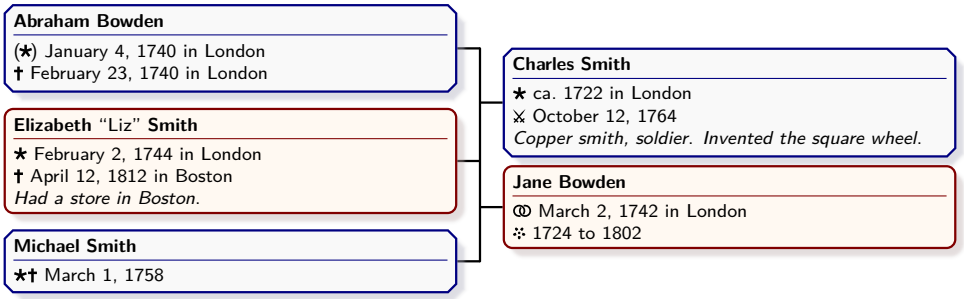
N 2017-09-15

```
template=database sidewaysportrait
```

This style is based on /gtr/processing → P. 128=database, sets /gtr/database format → P. 162=medium and /gtr/timeflow → P. 78=left.

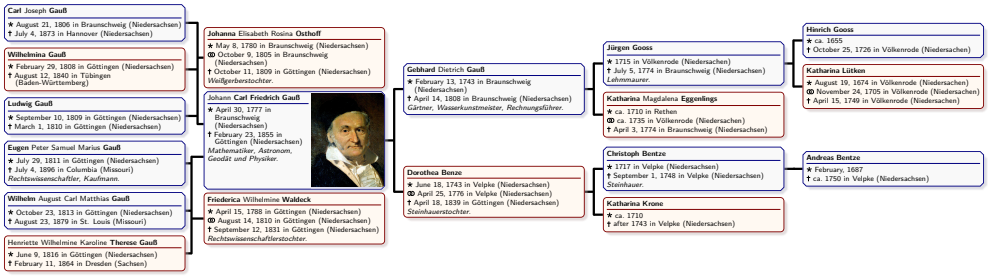
In contrast to database sideways (Section 12.16 on page 262), portraits are drawn, if present.

```
\begin{genealogypicture}
  [template=database sidewaysportrait]
  input{example.database.graph}
\end{genealogypicture}
```



The next example uses the graph data from Section 2.3.5 on page 39.

```
\begin{genealogypicture}[template=database sidewaysportrait,
    tikzpicture={scale=0.4,transform shape}]
  input{example.gauss.graph}
\end{genealogypicture}
```



12.19 Template 'database sidewaysportrait reduced'

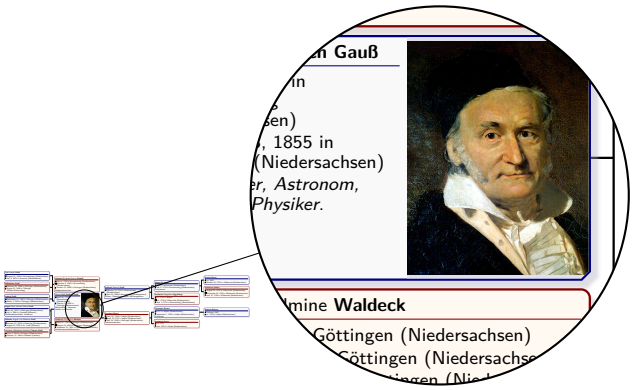
`template=database sidewaysportrait reduced`

N 2017-09-15

This style is identical to **database sidewaysportrait** (Section 12.18 on page 265), but every size property is scaled by factor $\frac{1}{10}$. See Section 12.9 on page 253 for more explanations.

The following example uses the graph data from Section 2.3.5 on page 39.

```
\begin{tikzpicture}[
    spy using outlines={circle, magnification=10, size=5cm, connect spies}]
  \genealogytreeinput[template=database sidewaysportrait reduced]{example.gauss.graph}
  \spy on (-0.2,-0.5) in node [left] at (7,1);
\end{tikzpicture}
```



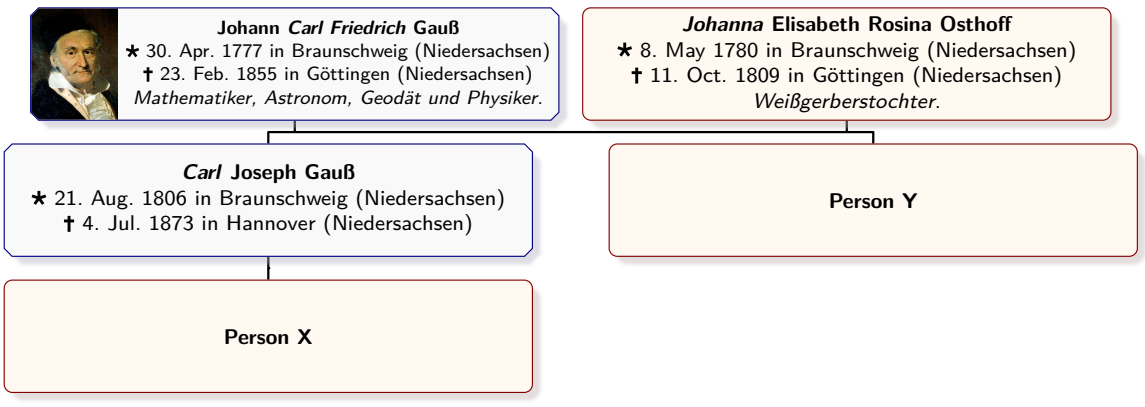
12.20 Template 'database relationship'

```
template=database relationship
```

This style is based on /gtr/processing→ P. 128=database, sets /gtr/database format→ P. 162=medium no marriage and /gtr/timeflow→ P. 78=down.

This template is intended to be used for diagrams which show the relationship of a person X to a person Y with common ancestors. If /gtr/database/image→ P. 156 is present, the corresponding image is inserted.

```
\begin{genealogypicture}[ template=database relationship, node size=7cm ]
child{
  g{male,
    name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
    birth={1777-04-30}{Braunschweig (Niedersachsen)},
    death={1855-02-23}{G\"ottingen (Niedersachsen)},
    profession={Mathematiker, Astronom, Geod\"at und Physiker},
    image={Carl_Friedrich_Gauss.jpg},
  }
  p{female,
    name={\pref{Johanna} Elisabeth Rosina \surn{Osthoff}},
    birth={1780-05-08}{Braunschweig (Niedersachsen)},
    marriage={1805-10-09}{Braunschweig (Niedersachsen)},
    death={1809-10-11}{G\"ottingen (Niedersachsen)},
    comment={Wei\ss{}gerberstochter},
  }
  child{
    g{male,
      name={\pref{Carl} Joseph \surn{Gau\ss{}}},
      birth={1806-08-21}{Braunschweig (Niedersachsen)},
      death={1873-07-04}{Hannover (Niedersachsen)},
    }
    c{ female, name={Person X} }
  }
  c{ female, name={Person Y} }
}
\end{genealogypicture}
```



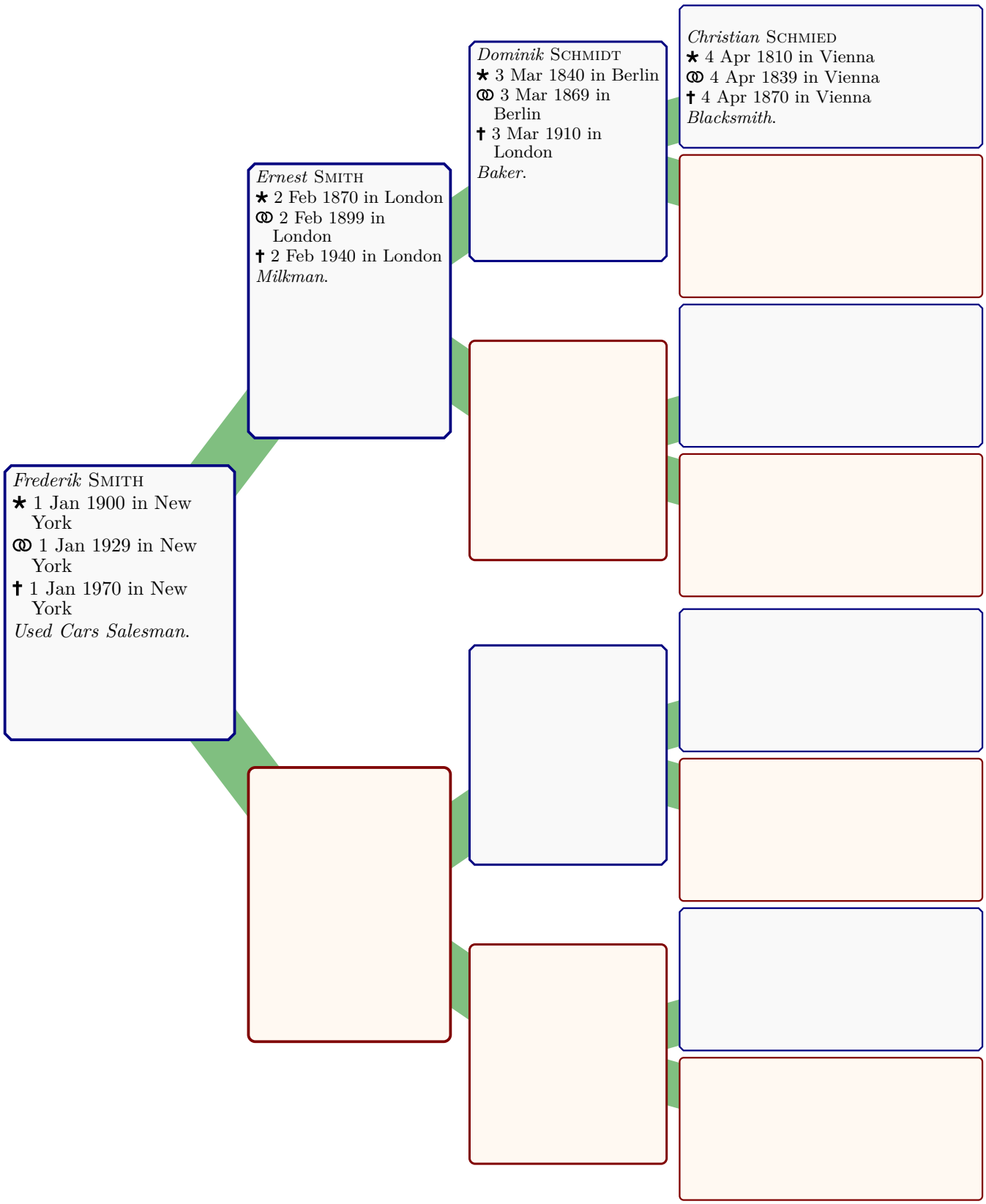
12.21 Template 'ahnentafel 3'

```
template=ahnentafel 3
```

This style is based on /gtr/processing→ P. 128=database and sets /gtr/timeflow→ P. 78=left.

Note that this style is very restrictive and its sole intended use is to easily set up predefined ancestor tables with three generations of ancestors. One should apply only parent, p, and g constructs which gives a binary tree.

```
\begin{genealogypicture}[template=ahnentafel 3,empty name text={},
  date format=d mon yyyy
]
  parent{
    g{male,name=\pref{Frederik} \surn{Smith},
      birth={1900-01-01}{New York},death={1970-01-01}{New York},
      marriage={1929-01-01}{New York},comment={Used Cars Salesman}}
    parent{
      g{male,name=\pref{Ernest} \surn{Smith},
        birth={1870-02-02}{London},death={1940-02-02}{London},
        marriage={1899-02-02}{London},comment={Milkman}}
      parent{
        g{male,name=\pref{Dominik} \surn{Schmidt},
          birth={1840-03-03}{Berlin},death={1910-03-03}{London},
          marriage={1869-03-03}{Berlin},comment={Baker}}
        p{male,name=\pref{Christian} \surn{Schmied},
          birth={1810-04-04}{Vienna},death={1870-04-04}{Vienna},
          marriage={1839-04-04}{Vienna},comment={Blacksmith}}
        p{female}
      }
      parent{ g{female} insert{gtrparent1} }
    }
    parent{ g{female} insert{gtrparent2} }
  }
\end{genealogypicture}
```


Frederik SMITH
★ 1 Jan 1900 in New York
⚭ 1 Jan 1929 in New York
† 1 Jan 1970 in New York
Used Cars Salesman.
Ernest SMITH
★ 2 Feb 1870 in London
⚭ 2 Feb 1899 in London
† 2 Feb 1940 in London
Milkman.
Dominik SCHMIDT
★ 3 Mar 1840 in Berlin
⚭ 3 Mar 1869 in Berlin
† 3 Mar 1910 in London
Baker.
Christian SCHMIED
★ 4 Apr 1810 in Vienna
⚭ 4 Apr 1839 in Vienna
† 4 Apr 1870 in Vienna
Blacksmith.

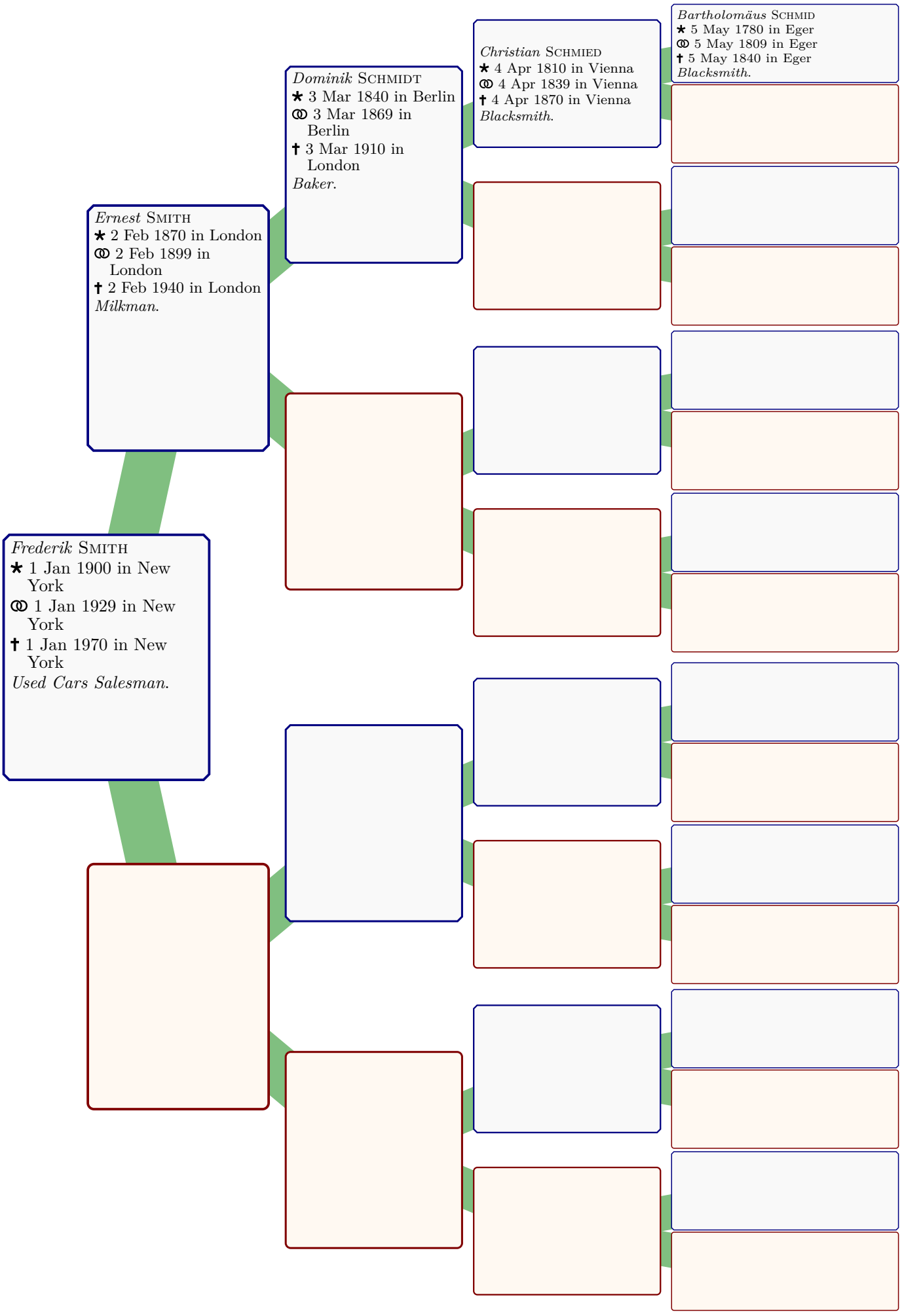
12.22 Template 'ahnentafel 4'

```
template=ahnentafel 4
```

This style is based on /gtr/processing→ P. 128=database and sets /gtr/timeflow→ P. 78=left.

Note that this style is very restrictive and its sole intended use is to easily set up predefined ancestor tables with four generations of ancestors. One should apply only parent, p, and g constructs which gives a binary tree. Since the first parent generation is shifted, the diagram should always contain mother and father of the proband to avoid overlapping.

```
\begin{genealogypicture}[template=ahnentafel 4,empty name text={},
  date format=d mon yyyy
]
  parent{
    g{male,name=\pref{Frederik} \surn{Smith},
      birth={1900-01-01}{New York},death={1970-01-01}{New York},
      marriage={1929-01-01}{New York},comment={Used Cars Salesman}}
    parent{
      g{male,name=\pref{Ernest} \surn{Smith},
        birth={1870-02-02}{London},death={1940-02-02}{London},
        marriage={1899-02-02}{London},comment={Milkman}}
      parent{
        g{male,name=\pref{Dominik} \surn{Schmidt},
          birth={1840-03-03}{Berlin},death={1910-03-03}{London},
          marriage={1869-03-03}{Berlin},comment={Baker}}
        parent{
          g{male,name=\pref{Christian} \surn{Schmied},
            birth={1810-04-04}{Vienna},death={1870-04-04}{Vienna},
            marriage={1839-04-04}{Vienna},comment={Blacksmith}}
          p{male,name=\pref{Bartholom\"aus} \surn{Schmid},
            birth={1780-05-05}{Eger},death={1840-05-05}{Eger},
            marriage={1809-05-05}{Eger},comment={Blacksmith}}
          p{female}
        }
        parent{ g{female} insert{gtrparent1} }
      }
      parent{ g{female} insert{gtrparent2} }
    }
    parent{ g{female} insert{gtrparent3} }
  }
\end{genealogypicture}
```


Frederik Smith
★ 1 Jan 1900 in New York
⚭ 1 Jan 1929 in New York
† 1 Jan 1970 in New York
Used Cars Salesman.
Ernest Smith
★ 2 Feb 1870 in London
⚭ 2 Feb 1899 in London
† 2 Feb 1940 in London
Milkman.
Dominik Schmidt
★ 3 Mar 1840 in Berlin
⚭ 3 Mar 1869 in Berlin
† 3 Mar 1910 in London
Baker.
Christian Schmied
★ 4 Apr 1810 in Vienna
⚭ 4 Apr 1839 in Vienna
† 4 Apr 1870 in Vienna
Blacksmith.
Bartholomäus Schmid
★ 5 May 1780 in Eger
⚭ 5 May 1809 in Eger
† 5 May 1840 in Eger
Blacksmith.

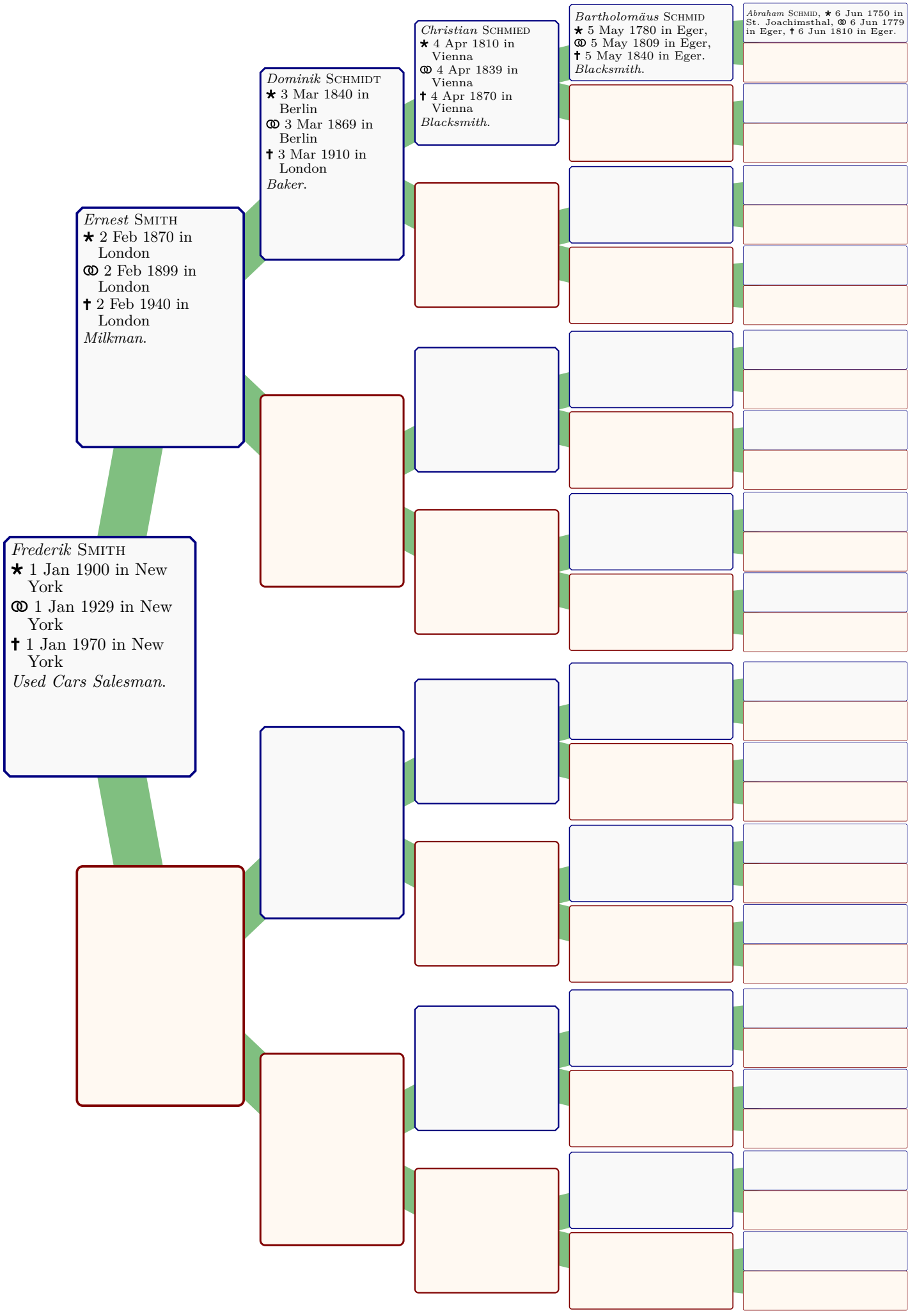
12.23 Template 'ahnentafel 5'

```
template=ahnentafel 5
```

This style is based on /gtr/processing→ P. 128=database and sets /gtr/timeflow→ P. 78=left.

Note that this style is very restrictive and its sole intended use is to easily set up predefined ancestor tables with five generations of ancestors. One should apply only parent, p, and g constructs which gives a binary tree. Since the first parent generation is shifted, the diagram should always contain mother and father of the proband to avoid overlapping.

```
\begin{genealogypicture}[template=ahnentafel 5,empty name text={},
  date format=d mon yyyy
]
  parent{
    g{male,name=\pref{Frederik} \surn{Smith},
      birth={1900-01-01}{New York},death={1970-01-01}{New York},
      marriage={1929-01-01}{New York},comment={Used Cars Salesman}}
    parent{
      g{male,name=\pref{Ernest} \surn{Smith},
        birth={1870-02-02}{London},death={1940-02-02}{London},
        marriage={1899-02-02}{London},comment={Milkman}}
      parent{
        g{male,name=\pref{Dominik} \surn{Schmidt},
          birth={1840-03-03}{Berlin},death={1910-03-03}{London},
          marriage={1869-03-03}{Berlin},comment={Baker}}
        parent{
          g{male,name=\pref{Christian} \surn{Schmied},
            birth={1810-04-04}{Vienna},death={1870-04-04}{Vienna},
            marriage={1839-04-04}{Vienna},comment={Blacksmith}}
          parent{
            g{male,name=\pref{Bartholom\"aus} \surn{Schmid},
              birth={1780-05-05}{Eger},death={1840-05-05}{Eger},
              marriage={1809-05-05}{Eger},comment={Blacksmith}}
            p{male,name=\pref{Abraham} \surn{Schmid},
              birth={1750-06-06}{St. Joachimsthal},death={1810-06-06}{Eger},
              marriage={1779-06-06}{Eger},comment={Miner}}
            p{female}
          }
          parent{ g{female} insert{gtrparent1} }
        }
        parent{ g{female} insert{gtrparent2} }
      }
      parent{ g{female} insert{gtrparent3} }
    }
    parent{ g{female} insert{gtrparent4} }
  }
\end{genealogypicture}
```


Frederik Smith
✱ 1 Jan 1900 in New York
⚭ 1 Jan 1929 in New York
† 1 Jan 1970 in New York
Used Cars Salesman.
Ernest Smith
✱ 2 Feb 1870 in London
⚭ 2 Feb 1899 in London
† 2 Feb 1940 in London
Milkman.
Dominik Schmidt
✱ 3 Mar 1840 in Berlin
⚭ 3 Mar 1869 in Berlin
† 3 Mar 1910 in London
Baker.
Christian Schmied
✱ 4 Apr 1810 in Vienna
⚭ 4 Apr 1839 in Vienna
† 4 Apr 1870 in Vienna
Blacksmith.
Bartholomäus Schmid
✱ 5 May 1780 in Eger,
⚭ 5 May 1809 in Eger,
† 5 May 1840 in Eger.
Blacksmith.
Abraham Schmid, ✱ 6 Jun 1750 in St. Joachimsthal, ⚭ 6 Jun 1779 in Eger, † 6 Jun 1810 in Eger.

12.24 Predefined Colors of the Library

The following colors are predefined. They are used as default colors in some templates.

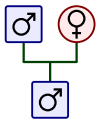
gtr_Blue_1 | gtr_Blue_2 | gtr_Blue_3
gtr_Blue_4 | gtr_Blue_5 | gtr_Blue_6
gtr_Bright_Red | gtr_Peach_Back | gtr_Peach_Frame
gtr_Yellow_Back | gtr_Yellow_Frame

12.25 Auxiliary Control Sequences

`\gtrparent1`

This control sequence inserts a pair of parents with content `male` and `female`.

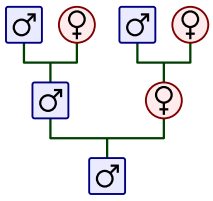
```
\begin{genealogypicture}
  [template=symbol nodes]
  parent{
    g{male}
    insert{gtrparent1}
  }
\end{genealogypicture}
```



`\gtrparent2`

This control sequence inserts two generations of parents with content `male` and `female`.

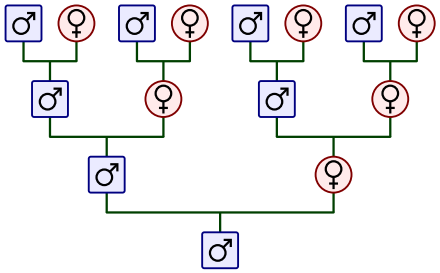
```
\begin{genealogypicture}
  [template=symbol nodes]
  parent{
    g{male}
    insert{gtrparent2}
  }
\end{genealogypicture}
```



`\gtrparent3`

This control sequence inserts three generations of parents with content `male` and `female`.

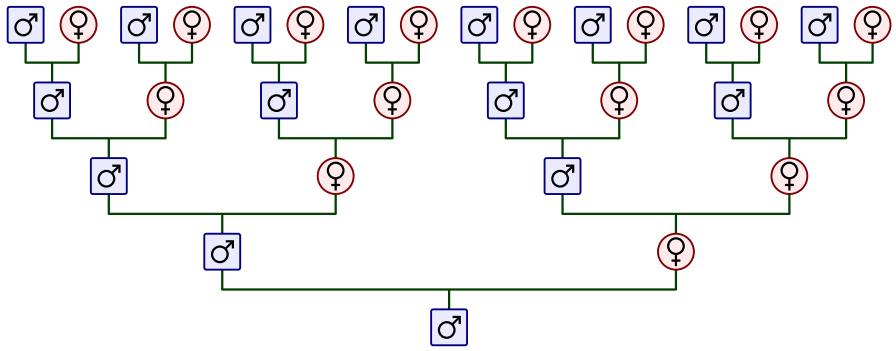
```
\begin{genealogypicture}
  [template=symbol nodes]
  parent{
    g{male}
    insert{gtrparent3}
  }
\end{genealogypicture}
```



\gtrparent4

This control sequence inserts four generations of parents with content `male` and `female`.

```
\begin{genealogypicture}
  [template=symbol nodes]
  parent{
    g{male}
    insert{gtrparent4}
  }
\end{genealogypicture}
```



\gtrparent5

This control sequence inserts five generations of parents with content `male` and `female`.

\gtrparent6

This control sequence inserts six generations of parents with content `male` and `female`.

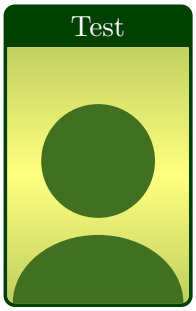
\gtrparent7

This control sequence inserts seven generations of parents with content `male` and `female`.

\gtrDrawSymbolicPortrait

Inserts Ti*k*Z code to draw a symbolic portrait. The colors are frame and back color of a `tcolorbox`. Therefore, the intended application is inside a `tcolorbox`.

```
\begin{tcolorbox}[enhanced,width=2.5cm,height=4cm,
  title=Test,halign title=center,
  colframe=green!25!black,colback=yellow!50,
  underlay={\begin{tcbclipinterior}%
      \path[fill overzoom picture=\gtrDrawSymbolicPortrait]
      (interior.south west) rectangle (interior.north east);
    \end{tcbclipinterior}} ]
\end{tcolorbox}
```



13

Auto-Layout Algorithm

13.1 Preliminaries

As discussed before in Chapter 1 on page 9 and Chapter 4 on page 61, *genealogy trees* can be considered as rooted ordered trees of unbounded degree with annotations. Therefore, an auto-layout algorithm for *genealogy trees* should be some extension of a known algorithm for tree layout which considers the family-centric approach.

The basic ideas for aesthetic properties and implementation are taken from Reingold and Tilford [2], Walker [5], and Buchheim, Jünger, and Leipert [1]. To dampen expectations early, the actual implementation is some extended Reingold and Tilford algorithm and does not consider the aesthetic balancing of small subtrees as presented in more recent research. There are multi-fold reasons for this ranging from performance and implementation complexity considerations in pure LaTeX to the simply question, if balancing is needed or even obstructive for this special application. We will come back to this later.

13.1.1 Aesthetic Properties

First, let us consider aesthetic properties which are usually desired when drawing trees. The following wording is intended for vertically (mainly top-down) oriented trees:

(A1) The y coordinate of a node is given by its level.

(A2) The edges do not cross each other and nodes on the same level have a minimal horizontal distance.

(A3) Isomorphic subtrees are drawn identically up to translation.

(A4) The order of the children of a node is displayed in the drawing.

(A5) The drawing of the reflection of a tree is the reflected drawing of the original tree.

Some of these properties cannot be guaranteed by the implementation and some are even violated deliberately.

13.1.2 Genealogy Trees

In supplement to typical graph theory notions, there is the additional *family* term for *genealogy trees*:

(G1) A family is an ordered set of parent nodes and child nodes.

(G2) All parent nodes of a family are connected with edges to all child nodes of the same family.

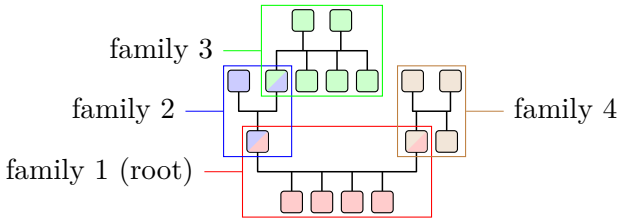
(G3) A node is child to zero or none family and is parent to zero or arbitrary many families.

These three properties alone would allow to construct *genealogy graphs* since they do not restrict to a tree-like structure.

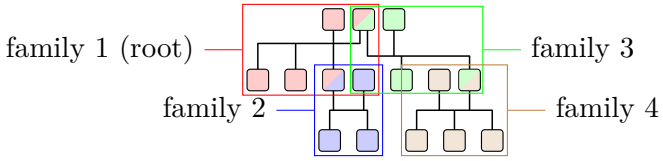
A parent node of a family is called a *leaf*, if it is not child to another family. A child node of a family is called a *leaf*, if it is not parent to another family.

For *genealogy trees*, the graph is required to be *connected* and to comply with exactly *one* of the following requirements:

(G4a) All child nodes of a family are leaf nodes with exception of at most one (*parent tree*).



(G4b) All parent nodes of a family are leaf nodes with exception of at most one (*child tree*).



Finally, we always consider *rooted* graphs. If (G4a) is fulfilled, there has to be a *root family* where all child nodes of a family are leaf nodes. If (G4b) is fulfilled, there has to be a *root family* where all parent nodes of a family are leaf nodes.

It is quite obvious that there are *genealogy trees* fulfilling (G1)–(G4) which cannot comply with (A2). Edge crossing is quite likely, but should still be minimized. The minimal distance of nodes on the same level may be deliberately different to emphasize different node affiliations.

13.1.3 Graph Grammar

Genealogy trees fulfilling (G1)–(G4) are described by the graph grammar of Chapter 4 on page 61. This given grammar is certainly not without alternative. Also, there are child trees fulfilling (G1)–(G4b) which cannot be represented in this grammar.

- `parent` constructs including `g`, `c`, and `p` represent *parent trees* fulfilling (G1)–(G4a).
- `child` constructs including `g`, `c`, `p`, and `union` represent *child trees* fulfilling (G1)–(G4b).
- `sandclock` constructs are an extension. They are a handy combination of a *parent tree* and a *child tree*.
- Nodes which are parent to one family and child to another family are `g`-nodes.
- The *root node* of a *parent tree* or *child tree* is the `g`-node of the *root family*.

13.2 Requirements

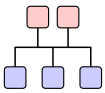
The aesthetic properties (A1)–(A5) are generally desired for the implemented auto-layout algorithm. While (A1), (A3), and (A4)[1] are considered to be fulfilled, (A2) cannot be guaranteed for *child tree*. This was discussed before and can be seen explicitly in Section 13.2.2 on page 280. Property (A5) is loosely considered in the following alignment requirements, but is not covered by special algorithmic efforts. Besides the effects known from the Reingold and Tilford [2] algorithm, there are additional violations of reflections for edge drawing; also see Section 13.2.2 on page 280 for this.

13.2.1 Parent and Child Alignment

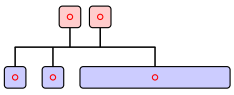
The following wording is intended for top-down oriented trees, but applied analogously for other growing directions.

For a family, the parent nodes and the child nodes should be placed centered to each other. This means that the center point of all parents should be vertically in congruence with the center point of all children.

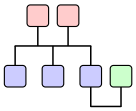
Here, every parent is parent of just one family. If a parent node is parent to more than one family, see Section 13.2.2 on page 280.



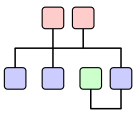
The parent nodes (red) are placed centered above the child nodes (blue).



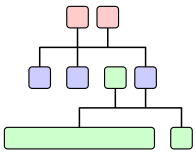
The center points for children and parents are computed as the half way between the center of the leftmost node and the center of the rightmost node.



Even, if a node is member of another family, only the center of the node itself is considered for alignment.



Even, if a node is member of another family, only the center of the node itself is considered for alignment.



Even, if a node is member of another family, only the center of the node itself is considered for alignment.

[1] One can argue about fulfillment of (A4). The graph grammar restricts children of `union` constructs to be grouped together while the children of the embedding `child` can be placed freely. The algorithm displays the order of the children as far as it is described by the grammar, but one could construct trees fulfilling (G1)–(G4b) which cannot by described by the grammar.

13.2.2 Patchwork Families

If a parent node is parent to more than one family, this is described by a `child` construct which embeds one or more `union` constructs. The `g`-node of the `child` family is also the implicit `g`-node of all `union` families. In this case, the combination of the `child` family with all directly embedded `union` families is called the *patchwork family* of the `g`-node which is parent to all children of the patchwork family.

The parent and child alignment considered in Section 13.2.1 on page 279 is now done for the whole patchwork family. This means that the center point of all parents should be vertically in congruence with the center point of all children of the patchwork family.

While node placement is a straightforward extension to Section 13.2.1 on page 279, edge placement is more difficult since edge crossing is quite likely. Therefore, the interconnections are to be separated vertically. This does not hinder crosspoints, but reduces the probability for lines lying on other lines.

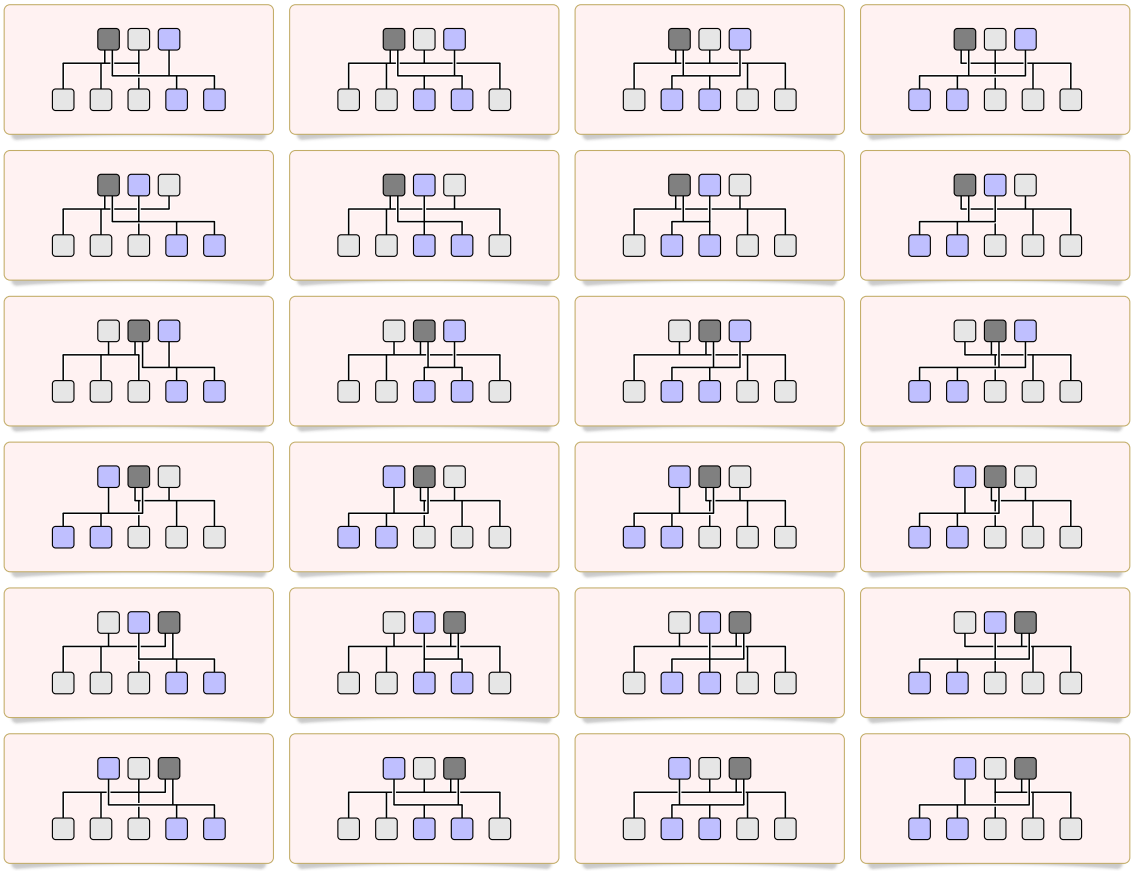
The fulfillment of this requirement results in a best-effort algorithm. The following still small example gives a highlight on the vast amount of possible edge configurations for complex patchwork families.

Edge Varieties for Families with Unions

Consider a `child` family with three children together with a `union` with two children, e.g.

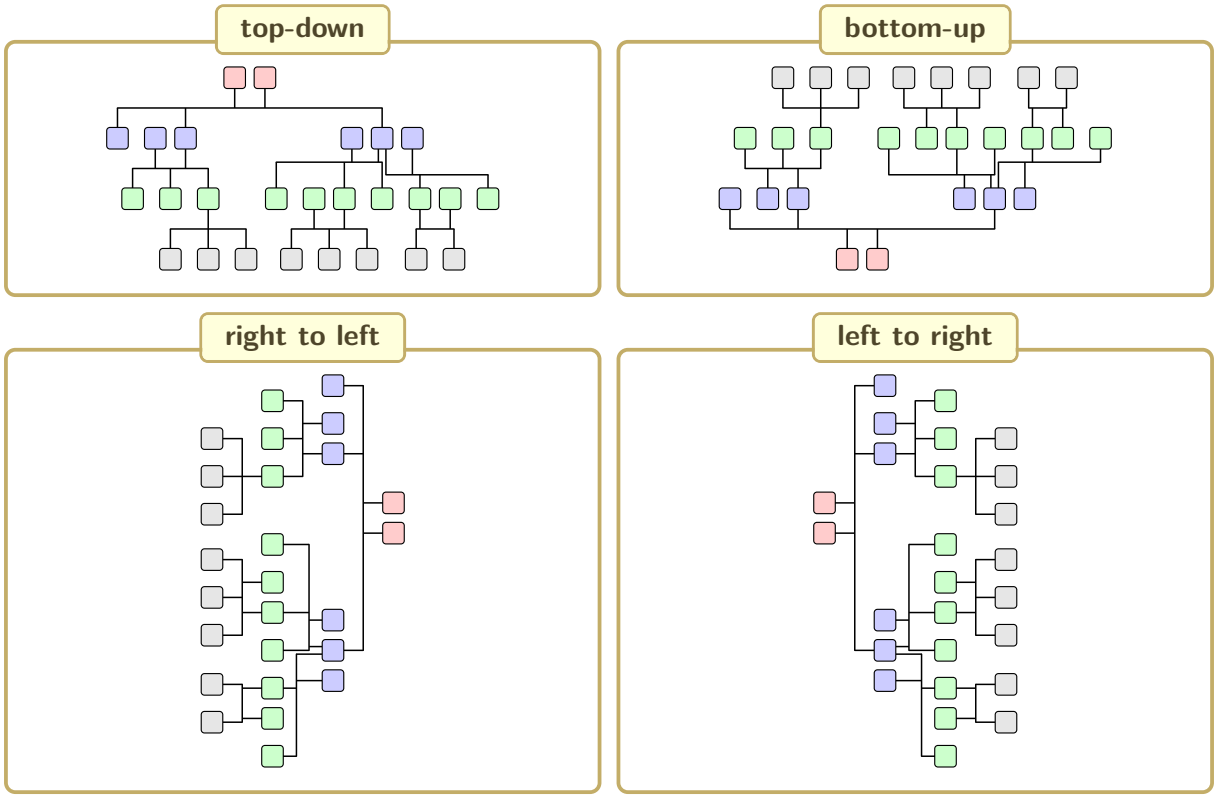
```
child{
  g-p-c-c-c-
  union{p-c-c-}
}
```

Depending on the order of the nodes, here, there are 16 different edge configurations:



13.2.3 Graph Growing Direction

The wording used in this chapter applies to top-down oriented trees, but the auto-layout algorithm should consider all four standard directions for graph growing.



The graph growing direction can be selected by setting the `/gtr/timeflow` → P. 78 option appropriately.

13.3 Algorithmic Steps

The following steps use the notations for *child trees* which grow top-down.

13.3.1 Recursive Family and Node Placement

The tree is constructed recursively in a bottom-up flow. The y-coordinate of a family is given by the current level while the x-coordinate is computed as relative *offset* to the enclosing family. This *offset* is initially 0pt.

During processing for a `child` family, every enclosed `child` and `union` is worked on recursively to construct their corresponding subtrees independently. This results in a list of children subtrees where the direct children of the original `child` family are the root `g`-nodes. This list also contains leaf child nodes and all direct leaf child nodes and child subtrees for all enclosed `union` families.

After the construction of this children list with all their subtrees, each leaf or subtree is placed step by step as close as possible to the right of the already placed last leaf or subtree. The placement is stored into the *offset* value of a subtree or directly for a leaf node.

The same procedure applies to the parent nodes of the original `child` family but with reduced complexity since all parents are leaf nodes.

Finally, the center points (*pivot* points) of all placed children and analogously of all placed parents are computed. All parents are shifted to get both points to congruence[2].

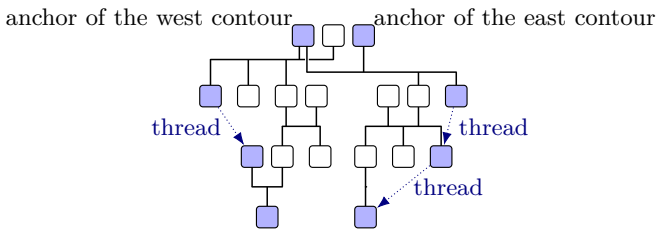
This concludes the computation for the current `child` family.

13.3.2 Contours

The core of the algorithm is to place one subtree (or leaf node) as close as possible to the right of another subtree (or leaf node). This is done following the ideas of Reingold and Tilford [2] by housekeeping contours for subtrees. Every `child` family keeps an anchor to a starting node for its *west contour* and its *east contour*. The *west contour* is the sequence of all leftmost nodes in the whole subtree, while the *east contour* is the sequence of all rightmost nodes in the whole subtree.

Every node itself has a *west contour value* and an *east contour value* describing the relative x-coordinate of the left and right border of this node in relation to its enclosing family.

When a *west contour* is followed starting from its anchor, the *next* node in the *west contour* after the current contour node is the leftmost leaf (patchwork) child or the parent node of the very first (patchwork) child, if the current node is no leaf node. Otherwise, a *thread* is used to note the next node plus a *thread gap* which is saved for housekeeping.

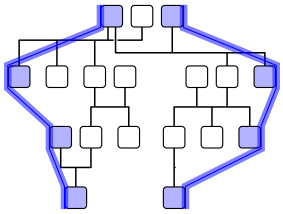


[2] As described in detail in this document, this algorithm can be adapted with various option settings, e.g. to change the pivot alignment procedure

Analogously, when an *east contour* is followed starting from its anchor, the *next* node in the *east contour* after the current contour node is the rightmost leaf (patchwork) child or the parent node of the very last (patchwork) child, if the current node is no leaf node. Otherwise, a *thread* is used to note the next node plus a *thread gap*.

For direct children or child families, the relative position is known or computed by the family *offset* values and the stored contour values. For nodes on a thread, this cannot be done and, therefore, the *thread gap* is needed.

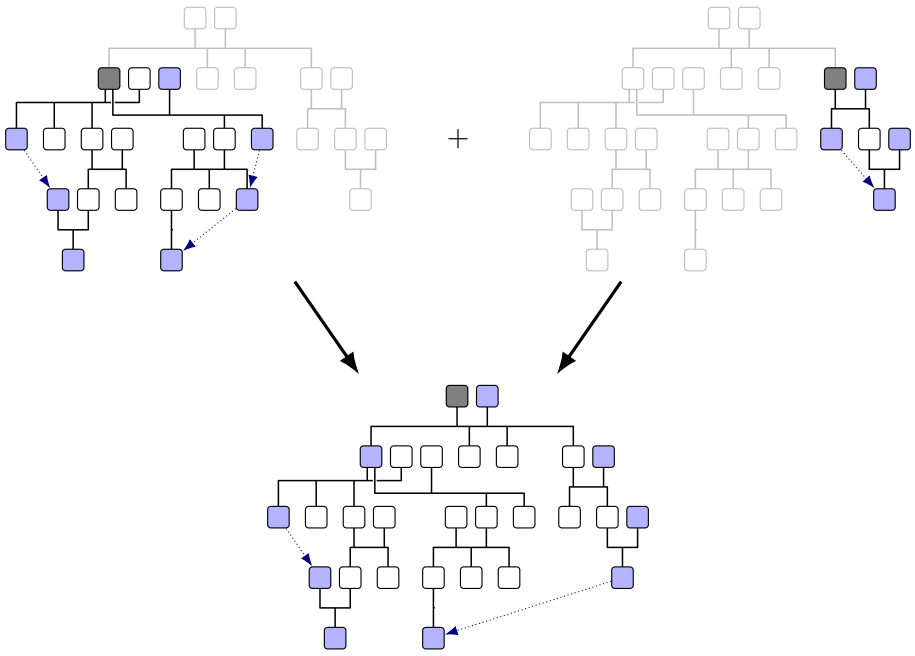
The LIB debug library documented in Chapter 11 on page 227, provides the `\gtrprocessordebug` → P. 230 command which displays the *offset* value and the different contour values. Also, `\gtrdebugdrawcontour` → P. 239 depicts the contours.



13.3.3 Combining Subtrees

The combining or sewing of two adjacent subtrees traverses the east contour of the left subtree and the west contour of the right subtree. The distance comparisons of every two contour nodes on the same level gives the required *offset* value for the right subtree. The combined forest of the two trees inherits the west contour of the left subtree and the east contour of the right subtree. If one of them is shorter than the other, it is prolonged by a *thread* as required.

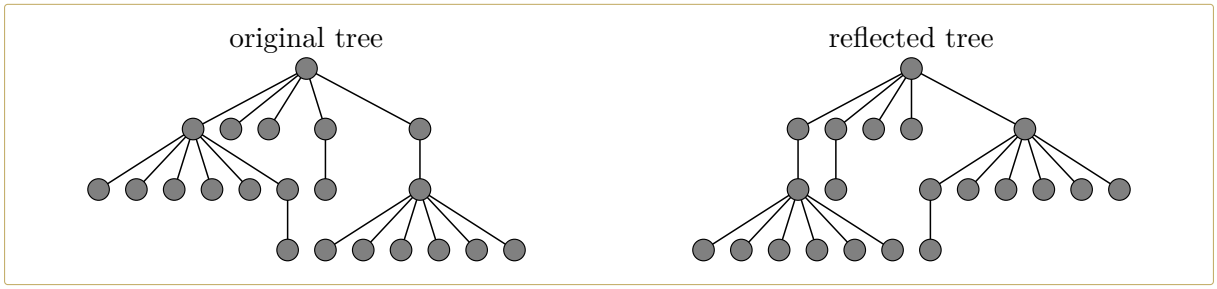
Sewing Subtrees



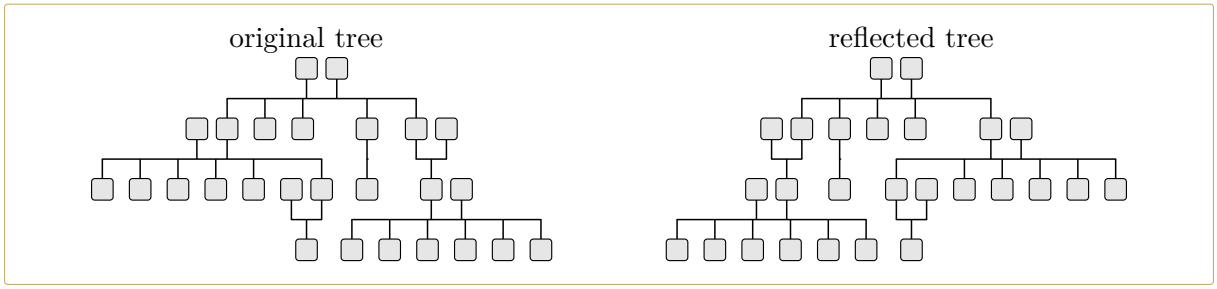
This is a shortened depiction of combining two leaf children step by step, followed by the rightmost subtree. Also, the two parent nodes of the new root family are added which are the new anchors for the east contour and the west contour.

13.4 Known Problems

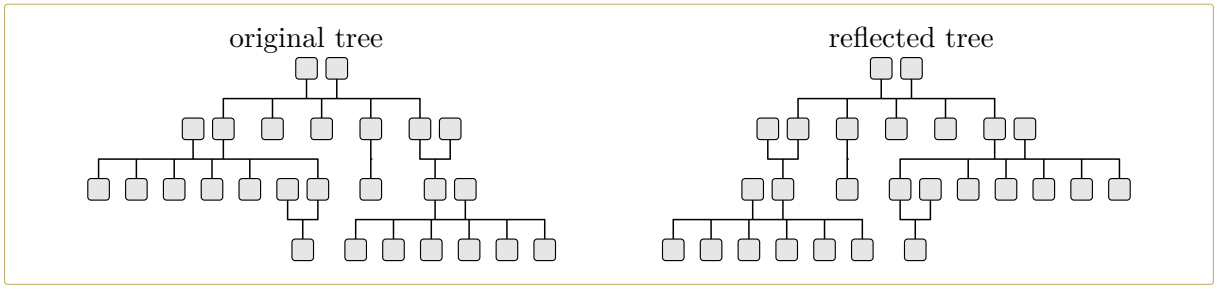
As was already mentioned before, the aesthetic property (A5) is not guaranteed by the auto-layout algorithm. The classic example for this is depicted below using the implemented auto-layout algorithm. Note that the small inner subtrees are not evenly spread but are crowded on the left-hand side.



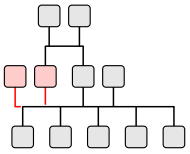
Next, the classic example is translated to *genealogy trees*. The effect is the same but arguable may be seen more negligible or at least acceptable. To avoid this automatically, some technique from [2, 5] would be needed.



Luckily, the algorithm is implemented in LaTeX with a lot of intervention points using options. If (A5) is really needed for aesthetic reasons, one can simply cheat by adding some `/gtr/distance` → P. 94 options at the crucial small subtrees:

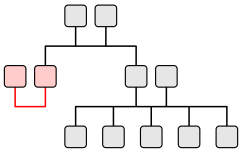


Another known problem is edge crossing which violates (A2), but this is for some patchwork families sheer unavoidable as even the small examples from Section 13.2.2 on page 280 show. Edge crossing can also happen for childless families, if the usual perpendicular edge drawing is used.



The edge between the two parent nodes (red) of the childless family is overlapped with the edge of the sibling family.

To solve the problem manually, a child with `/gtr/phantom` → P. 122 option can be added to the childless family:



The childless family (red) was given a child with the `phantom` option. This invisible child reserves the space needed for edge drawing.

14

Example Graph Files

The following example graph files are used for various examples inside this document.

14.1 example.option.graph

File «example.option.graph»

```
parent[id=SmithDoe]{
  g[id=Arth2008,male]{Arthur\\\gtrsymBorn\,2008}
  c[id=Bert2010,female]{Berta\\\gtrsymBorn\,2010}
  c[id=Char2014,male]{Charles\\\gtrsymBorn\,2014}
  parent[id=Smith]{
    g[id=John1980,male]{John Smith\\\gtrsymBorn\,1980}
    p[id=GpSm1949,male]{Grandpa Smith\\\gtrsymBorn\,1949}
    p[id=GmSm1952,female]{Grandma Smith\\\gtrsymBorn\,1952}
  }
  parent[id=Doe]{
    g[id=Jane1982,female]{Jane Doe\\\gtrsymBorn\,1982}
    c[id=Harr1987,male]{Uncle Harry\\\gtrsymBorn\,1987}
    p[id=GpDo1955,male]{Grandpa Doe\\\gtrsymBorn\,1955}
    p[id=GmDo1956,female]{Grandma Doe\\\gtrsymBorn\,1956}
  }
}
```

14.2 example.database.graph

Also see Section 7.2 on page 153.

File «example.database.graph»

```
child[id=SmitBowd1742]{
  g[id=SmitChar1722]{
    male,
    name       = {\pref{Charles} \surn{Smith}},
    birth      = {(caAD)1722}{London},
    baptism    = {1722-04-13}{London},
    death+     = {1764-10-12}{}{killed},
    profession = {Copper smith, soldier},
    comment    = {Invented the square wheel},
  }
  p[id=BowdJane1724]{
    female,
    name     = {\pref{Jane} \surn{Bowden}},
    floruit- = {1724/1802},
    marriage = {1742-03-02}{London},
  }
  c[id=BowdAbra1740]{
    male,
    name   = {\pref{Abraham} \surn{Bowden}},
    birth+ = {1740-01-04}{London}{out of wedlock},
    death  = {1740-02-23}{London}
  }
  c[id=SmitEliz1744]{
    female,
    name    = {\pref{Elizabeth} \nick{Liz} \surn{Smith}},
    birth   = {1744-02-02}{London},
    floruit = {1780/1805}{New York},
    death   = {1812-04-12}{Boston},
    comment = {Had a store in Boston},
  }
  c[id=SmitMich1758]{
    male,
    name   = {\pref{Michael} \surn{Smith}},
    birth+ = {1758-03-01}{}{died},
  }
}
```

14.3 example.formal.graph

File «example.formal.graph»

```
child[id=fam_A]{
  g[id=na1,male]{a_1}
  p[id=na2,female]{a_2}
  child[id=fam_B]{
    p[id=nb1,male]{b_1}
    g[id=na3,female]{a_3}
    c[id=nb2,male]{b_2}
    child[id=fam_E]{
      p[id=ne1,male]{e_1}
      g[id=nb3,female]{b_3}
      c[id=ne2,male]{e_2}
      c[id=ne3,female]{e_3}
    }
  }
  child[id=fam_C]{
    g[id=na4,male]{a_4}
    p[id=nc1,female]{c_1}
    child[id=fam_F]{
      g[id=nc2,male]{c_2}
      p[id=nf1,female]{f_1}
      c[id=nf2,male]{f_2}
      c[id=nf3,female]{f_3}
      c[id=nf4,male]{f_4}
    }
    union[id=fam_D]{
      p[id=nd1,female]{d_1}
      child[id=fam_G]{
        p[id=ng1,male]{g_1}
        g[id=nd2,female]{d_2}
        c[id=ng2,male]{g_2}
        c[id=ng3,female]{g_3}
        union[id=fam_H]{
          p[id=nh1,male]{h_1}
          c[id=nh2,male]{h_2}
        }
      }
      c[id=nd3,male]{d_3}
      child[id=fam_I]{
        g[id=nd4,male]{d_4}
        p[id=ni1,female]{i_1}
        c[id=ni2,female]{i_2}
        c[id=ni3,female]{i_3}
        c[id=ni4,female]{i_4}
      }
    }
  }
  c[id=na5,female]{a_5}
}
```

15

Stack Macros

The genealogytree package provides an elementary stack mechanism which is used internally, but may also be applied elsewhere. This elementary stack stores and retrieves expanded text first-in-last-out (FILO). There are no safe-guarding mechanisms implemented.

15.1 Creating a Stack

\gtrnewstack{⟨*name*⟩}

Creates a new empty stack ⟨*name*⟩.

```
\gtrnewstack{foo}              % new empty stack
Stack size: \gtrstacksize{foo}
```

Stack size: 0

\gtrstacksize{⟨*name*⟩}

Returns the current stack size.

```
\gtrnewstack{foo}
Stack size: \gtrstacksize{foo}
\gtrstackpush{foo}{a}\par
Stack size: \gtrstacksize{foo}
```

Stack size: 0
Stack size: 1

15.2 Push to a Stack

\gtrstackpush{⟨*name*⟩}{⟨*content*⟩}

Pushes ⟨*content*⟩ to a stack ⟨*name*⟩. The ⟨*content*⟩ is expanded during pushing.

```
\def\myx{X}
\gtrnewstack{foo}
\gtrstackpush{foo}{\myx}
\gtrstackpop{foo}
```

X

15.3 Pop from a Stack

`\gtrstackpop{`⟨*name*⟩`}`

Pops content from a stack ⟨*name*⟩. The last pushed content is popped first.

```
\gtrnewstack{foo}
\gtrstackpush{foo}{This}
\gtrstackpush{foo}{is}
\gtrstackpush{foo}{a}
\gtrstackpush{foo}{hello}
\gtrstackpush{foo}{world}
\gtrstackpop{foo} \gtrstackpop{foo} \gtrstackpop{foo}
\gtrstackpop{foo} \gtrstackpop{foo}
```

world hello a is This

`\gtrstackpopto{`⟨*name*⟩`}{`⟨*macro*⟩`}`

Pops content from a stack ⟨*name*⟩ into a ⟨*macro*⟩.

```
\gtrnewstack{foo}
\gtrstackpush{foo}{My}
\gtrstackpush{foo}{test}
\gtrstackpopto{foo}{\myA}
\gtrstackpopto{foo}{\myB}
'\myA' and '\myB'.
```

'test' and 'My'.

15.4 Peek into a Stack

`\gtrstackpeek{`⟨*name*⟩`}`

Reads from a stack ⟨*name*⟩ without reducing the stack content.

```
\gtrnewstack{foo}
\gtrstackpush{foo}{First entry}
'\gtrstackpeek{foo}', '\gtrstackpeek{foo}';
\gtrstackpush{foo}{Second entry}
'\gtrstackpeek{foo}', '\gtrstackpeek{foo}';
```

'First entry', 'First entry'; 'Second entry', 'Second entry';

`\gtrstackpeekto{`⟨*name*⟩`}{`⟨*macro*⟩`}`

Peeks content from a stack ⟨*name*⟩ into a ⟨*macro*⟩.

```
\gtrnewstack{foo}
\gtrstackpush{foo}{My}
\gtrstackpush{foo}{test}
\gtrstackpeekto{foo}{\myA}
\gtrstackpeekto{foo}{\myB}
'\myA' and '\myB'.
```

'test' and 'test'.

15.5 Creating Stack Shortcuts

`\gtrmakestack{`⟨*name*⟩`}`

Creates a new empty stack ⟨*name*⟩ and creates new macros ⟨*name*⟩`size`, ⟨*name*⟩`push`, ⟨*name*⟩`popto`, ⟨*name*⟩`pop`, ⟨*name*⟩`peekto`, and ⟨*name*⟩`peek`. These macros serve as shortcuts to the corresponding stack macros from above.

```
\gtrmakestack{foo}
\foopush{First}
\foopush{Second}
\foopush{Third}
The stack contains \foosize\ entries. The last one is '\foopeek'.

\foopopto\myA
The stack contains \foosize\ entries after '\myA' was removed.

The remaining entries are '\foopop' and '\foopop'.
Now, the stack contains \foosize\ entries.

Never pop an empty stack: \foopop \foosize
```

The stack contains 3 entries. The last one is 'Third'.
The stack contains 2 entries after 'Third' was removed.
The remaining entries are 'Second' and 'First'. Now, the stack contains 0 entries.
Never pop an empty stack: -1

```
\gtrmakestack{foo}
\foopush{Mary}\foopush{had}\foopush{a}\foopush{little}\foopush{lamb}

\loop\ifnum\foosize>0
  \foopop,
\repeat
```

lamb, little, a, had, Mary,

16

Version History

v1.21 (2017/09/15)

- Italian translation provided by Andrea Vaccari – new value `italian` for `/gtr/language` → P. 225.
- `\gtrsymFloruit` → P. 221 symbol rotated by 36 degrees for an enhanced optical differentiation from `\gtrsymBorn` → P. 219.
- New `/gtr/template` → P. 247 styles, see Chapter 12 on page 247, namely `database pole reduced` (Section 12.9 on page 253), `database poleportrait` (Section 12.10 on page 254), `database poleportrait reduced` (Section 12.11 on page 256), `database portrait reduced` (Section 12.13 on page 259), `database traditional reduced` (Section 12.15 on page 261), `database sideways reduced` (Section 12.17 on page 264), `database sidewaysportait` (Section 12.18 on page 265), `database sidewaysportait reduced` (Section 12.19 on page 266).

v1.20 (2017/07/18)

- Dutch translation provided by Dirk Bosmans – new value `dutch` for `/gtr/language` → P. 225.
- New event «floruit» with symbol `\gtrsymFloruit` → P. 221 proposed by Mikkel Eide Eriksen. Accompanying keys are `/gtr/database/floruit` → P. 158, `/gtr/database/floruit+` → P. 158, `/gtr/database/floruit-` → P. 158, `/gtr/symlang/Floruit` → P. 224, and `/gtr/event prefix/floruit` → P. 180. The standard options for `/gtr/database format` → P. 162 were extended for «floruit».

v1.10 (2017/01/29)

- Danish translation provided by Mikkel Eide Eriksen – new value `danish` for `/gtr/language` → P. 225.
- French translation provided by Denis Bitouzé – new value `french` for `/gtr/language` → P. 225.
- Separate document `genealogytree-languages` added to give a short survey of language settings.

- New values for /gtr/date format→P. 174: typical, dd mon.yyyy, d mon.yyyy, dd/mm yyyy, yyyymmdd. The initial setting is changed to be typical.
- New data key /gtr/database/profession→P. 156 accompanied by \gtrDBprofession, \gtrPrintProfession→P. 184, \gtrifprofessiondefined→P. 184, and /gtr/profession code→P. 184.
- Option settings for /gtr/database format→P. 162 updated to process /gtr/database/comment→P. 155 and /gtr/database/profession→P. 156.
- New environment gtrinfolist→P. 185 and new corresponding option /gtr/info separators→P. 185.
- Implementation for level sensitive variables changed to support extension packages.
- Implementation for \gtruselibrary→P. 13 changed to support third party extension packages.

v1.01 (2016/07/29)

- Family options like /gtr/pivot shift→P. 104 are now applicable for /gtr/options for family→P. 102 and \gtrsetoptionsforfamily→P. 102.
- New option /gtr/tikz→P. 101.
- New option /tikz/genealogytree extra edges scope→P. 216.
- New tutorial for multi-ancestors, see Section 2.5 on page 46.

v1.00 (2015/09/21)

- Library loading made compatible with expl3.
- New /gtr/template→P. 247 called database relationship (Section 12.20 on page 267).

v0.91 beta (2015/06/22)

- /gtr/phantom→P. 122 and /gtr/phantom*→P. 124 switch off more settings of tcolorbox.
- New options /gtr/id prefix→P. 92 and /gtr/id suffix→P. 92.
- New /gtr/template→P. 247 called database sideways (Section 12.16 on page 262).
- New tutorials “Externalization” (Section 2.6 on page 49) and “Conversion” (Section 2.7 on page 52).
- Settings for /gtr/date range full→P. 176 and /gtr/date range separator→P. 177 changed.
- /gtr/date format→P. 174 complemented with many more formats.

v0.90 beta (2015/05/22)

- First functional beta release.
- Full genealogy tree customization, tree positioning, input insertion and deletion, edge customization.
- Database processing.
- Genealogy symbols.
- Internationalization.
- Templates library.
- Tutorials.

v0.10 alpha (2015/01/12)

- Initial public release (alpha version).
- Grammar and Debugger as preview release.

v0.00 (2013-2014)

- Pre publication development.

Bibliography

[1] Christoph Buchheim, Michael Jünger, and Sebastian Leipert. "Improving Walker's Algorithm to Run in Linear Time". In: *Graph Drawing*. 10th International Symposium, GD 2002. Volume 2528. Lecture Notes in Computer Science. Springer, Nov. 8, 2002, pages 344–353.

[2] Edward M. Reingold and John S. Tilford. "Tidier drawings of trees". In: *IEEE Transations on Software Engineering* 7.2 (Mar. 1981), pages 223–228.

[3] Thomas F. Sturm. *The tcolorbox package. Manual for version 3.96*. Nov. 18, 2016. http://mirror.ctan.org/macros/latex/contrib/tcolorbox/tcolorbox.pdf.

[4] Till Tantau. *The TikZ and PGF Packages. Manual for version 3.0.1a*. Aug. 29, 2015. http://sourceforge.net/projects/pgf/.

[5] John Q. Walker II. "A Node-positioning Algorithm for General Trees". In: *Software. Practice and Experience* 20.7 (July 1990), pages 685–705.

Index

above value, 110, 111
AD value, 160
add child key, 207
add parent key, 208
adjust node key, 111
adjust position key, 110
after parser key, 109
after tree key, 113
ahnentafel 3 value, 268
ahnentafel 4 value, 270
ahnentafel 5 value, 272
all key, 13
all value, 176
all but AD value, 160, 176
anchoring key, 200

background key, 200
baptism key, 157, 179
baptism+ key, 157
baptism- key, 157
Baptized key, 224
BC value, 160
below value, 110, 111
birth key, 157, 179
birth+ key, 157
birth- key, 157
Born key, 224
Bornoutofwedlock key, 224
both value, 95, 211, 241
box key, 96
box clear key, 96
burial key, 159, 180
burial+ key, 159
burial- key, 159
Buried key, 224

c grammar, 17, 19, 21, 55, 59, 60, 62, 71, 75, 76, 115, 260, 278
ca value, 160
caAD value, 160
caBC value, 160
calendar print key, 176
calendar text for key, 176
center value, 200
child grammar, 16, 25, 26, 34, 46, 55, 60, 62, 65, 67, 69, 75–78, 85, 87, 103, 104, 106, 205, 239, 278–280, 282
child value, 95, 211
child distance key, 85
child distance in child graph key, 85
child distance in parent graph key, 84
code key, 125
Colors
    gtr_Blue_1, 274
    gtr_Blue_2, 274
    gtr_Blue_3, 274
    gtr_Blue_4, 274
    gtr_Blue_5, 274
    gtr_Blue_6, 274
    gtr_Bright_Red, 274
    gtr_Peach_Back, 274
    gtr_Peach_Frame, 274
    gtr_Yellow_Back, 274
    gtr_Yellow_Frame, 274
    gtrsymbol, 217, 218
comment key, 155
comment code key, 183
content interpreter key, 145
content interpreter code key, 146
content interpreter content key, 146
content interpreter id and content key, 149
contour key, 241
cremated key, 180
cremated value, 159
custom key, 197

d M yyyy value, 174
d mon yyyy value, 174
d mon.yyyy value, 174, 296
d month yyyy value, 174
d-M-yyyy value, 175
d-m-yyyy value, 174
d-mon-yyyy value, 175
d-month-yyyy value, 175
d.M.yyyy value, 174
d.m.yyyy value, 174
d.mon.yyyy value, 174
d.month yyyy value, 174
d/M/yyyy value, 174
d/m/yyyy value, 174
d/mon/yyyy value, 174
d/month/yyyy value, 174
danish value, 225, 295
database value, 35, 128, 151, 251, 254, 257, 260, 262, 265, 267, 268, 270, 272
database content interpreter key, 148
database format key, 162
database pole value, 251, 253, 254
database pole reduced value, 253, 295
database poleportrait value, 254, 256, 295
database poleportrait reduced value, 256, 295
database portrait value, 257, 259

database portrait reduced value, 259, 295
database relationship value, 267, 296
database sideways value, 262, 264, 265, 296
database sideways reduced value, 264, 295
database sidewaysportait value, 295
database sidewaysportait reduced value, 295
database sidewaysportrait value, 265, 266
database sidewaysportrait reduced value, 266
database traditional value, 260, 261
database traditional reduced value, 261, 295
database unknown key key, 159
date value, 180
date code key, 175
date format key, 174
date range after key, 177
date range before key, 176
date range full key, 176
date range separator key, 177
dd mm yyyy value, 174
dd mon yyyy value, 174
dd mon.yyyy value, 174, 296
dd-mm-yyyy value, 174
dd-mon-yyyy value, 175
dd-month-yyyy value, 175
dd.mm.yyyy value, 174
dd.mon.yyyy value, 174
dd/mm yyyy value, 174, 296
dd/mm/yyyy value, 174
dd/mon/yyyy value, 174
dd/month/yyyy value, 174
ddmonyyyy value, 175
death key, 158, 180
death+ key, 158
death- key, 158
debug key, 13
deletion content interpreter key, 147
Died key, 224
died key, 179
died value, 157
Diedonbirthday key, 224
directory value, 250
disconnect key, 211
distance key, 94
divorce key, 158, 180
divorce+ key, 158
divorce- key, 158
Divorced key, 224
down value, 78, 197, 260, 267
dutch value, 225, 295

east value, 241
edges key, 190
edges for family key, 192
edges for subtree key, 192
empty value, 162
empty name text key, 173
Engaged key, 224
engagement key, 158, 179
engagement+ key, 158
engagement- key, 158
english value, 223, 225
Environments
    exgenealogypicture, 57
    genealogypicture, 57
    gtreventlist, 181
    gtrinfolist, 185
    gtrprintlist, 181
    tcolorbox, 128, 133
    tikzpicture, 55, 57
error value, 159
event code key, 180
event format key, 180
event text key, 180
exgenealogypicture environment, 57
extra edges key, 212
extra edges for families key, 213
extra edges for family key, 213
extra edges prepend key, 214
extra edges prepend for families key, 215
extra edges prepend for family key, 215

family key, 103
family box key, 97
family database key, 205
family edges key, 191
family extra edges key, 212
family extra edges prepend key, 214
family id key, 240
family label key, 204
family number key, 240
Female key, 224
female key, 99, 155
female value, 155, 186
fit value, 128, 129, 248
fit to family key, 114
fit to subtree key, 114
Floruit key, 224
floruit key, 158, 180
floruit+ key, 158
floruit- key, 158
foreground key, 199
formal graph value, 247
french value, 225, 295
full value, 162, 172
full marriage above value, 162, 251, 254
full marriage below value, 162
full no marriage value, 162
Funeralurn key, 224
further distance key, 88

g grammar, 10, 17, 19, 25, 26, 34, 55, 59, 60, 62, 63, 65, 67, 69, 71, 75, 76, 115, 211, 245, 260, 268, 270, 272, 278, 280, 282
genealogypicture environment, 57
\genealogytree, 55
genealogytree edges scope key, 193
genealogytree extra edges scope key, 216

\genealogytreeinput, 56
german value, 224, 225
german-austrian value, 225
german-german value, 225
GR value, 160
Grammar
c, 17, 19, 21, 55, 59, 60, 62, 71, 75, 76, 115, 260, 278
child, 16, 25, 26, 34, 46, 55, 60, 62, 65, 67, 69, 75–78, 85, 87, 103, 104, 106, 205, 239, 278–280, 282
g, 10, 17, 19, 25, 26, 34, 55, 59, 60, 62, 63, 65, 67, 69, 71, 75, 76, 115, 211, 245, 260, 268, 270, 272, 278, 280, 282
input, 29, 72
insert, 73
p, 17, 19, 21, 32, 55, 59, 60, 62, 71, 75, 76, 115, 260, 268, 270, 272, 278
parent, 16, 17, 19, 20, 34, 55, 60, 62, 63, 69, 72, 75–78, 84, 86, 103, 104, 106, 205, 268, 270, 272, 278
sandclock, 34, 39, 55, 60, 62, 69, 104, 278
union, 26, 27, 55, 62, 65, 67, 104, 211, 239, 245, 278–280, 282
gtr_Blue_1 color, 274
gtr_Blue_2 color, 274
gtr_Blue_3 color, 274
gtr_Blue_4 color, 274
gtr_Blue_5 color, 274
gtr_Blue_6 color, 274
gtr_Bright_Red color, 274
gtr_Peach_Back color, 274
gtr_Peach_Frame color, 274
gtr_Yellow_Back color, 274
gtr_Yellow_Frame color, 274
\gtrBoxContent, 144
\gtrDBcomment, 155
\gtrDBimage, 38, 156
\gtrDBkekule, 156
\gtrDBname, 155
\gtrDBprofession, 156, 296
\gtrDBrelationship, 156
\gtrDBsex, 36, 155, 186
\gtrDBshortname, 155
\gtrDBuuid, 156
\gtrdebugdrawcontour, 239
\gtrDeclareDatabaseFormat, 170
\gtrDrawSymbolicPortrait, 275
\gtredgeset, 193
gtreventlist environment, 181
\gtrifchild, 60
\gtrifcnode, 59
\gtrifcommentdefined, 183
\gtrifdatedefined, 174
\gtrifeventdefined, 179
\gtriffemale, 186
\gtrifgnode, 59
\gtrifimagedefined, 187
\gtrifleaf, 60
\gtrifleafchild, 60
\gtrifleafparent, 60
\gtrifmale, 186
\gtrifnodeid, 59
\gtrifparent, 60
\gtrifplacedefined, 178
\gtrifpnode, 59
\gtrifprofessiondefined, 184
\gtrifroot, 60
\gtrignorenode, 116
\gtrignoresubtree, 116
gtrinfolist environment, 185
\gtrkeysappto, 58
\gtrkeysgappto, 58
\gtrlanguagename, 225
\gtrlistseparator, 181
\gtrloadlanguage, 226
\gtrmakestack, 293
\gtrnewstack, 291
\gtrNodeBoxOptions, 144
gtrNodeDimensions key, 129, 133, 136, 139, 142
gtrNodeDimensionsLandscape key, 129, 133, 136, 139, 142
\gtrnodefamily, 59
\gtrnodeid, 59
\gtrnodelevel, 59
\gtrNodeMaxHeight, 144
\gtrNodeMaxWidth, 144
\gtrNodeMinHeight, 144
\gtrNodeMinWidth, 144
\gtrnodenumber, 59
\gtrnodetype, 59
\gtrparent1, 274
\gtrparent2, 274
\gtrparent3, 274
\gtrparent4, 275
\gtrparent5, 275
\gtrparent6, 275
\gtrparent7, 275
\gtrParseDate, 161
\gtrparserdebug, 228
\gtrparserdebuginput, 229
\gtrPrintComment, 183
\gtrPrintDate, 174
\gtrPrintEvent, 179
\gtrPrintEventPrefix, 179
gtrprintlist environment, 181
\gtrPrintName, 172
\gtrPrintPlace, 178
\gtrPrintProfession, 184
\gtrPrintSex, 186
\gtrprocessordebug, 230
\gtrprocessordebuginput, 233
\gtrset, 58
\gtrsetoptionsforfamily, 102
\gtrsetoptionsfornode, 93
\gtrsetoptionsforsubtree, 105
\gtrstackpeek, 292
\gtrstackpeekto, 292

\gtrstackpop, 292
\gtrstackpopto, 292
\gtrstackpush, 291
\gtrstacksize, 291
\gtrsymBaptized, 219
gtrsymbol color, 217, 218
\gtrSymbolsFullLegend, 222
\gtrSymbolsLegend, 223
\gtrSymbolsRecordReset, 222
\gtrSymbolsSetCreate, 218
\gtrSymbolsSetCreateSelected, 218
\gtrSymbolsSetDraw, 218
\gtrsymBorn, 219
\gtrsymBornoutofwedlock, 219
\gtrsymBuried, 221
\gtrsymDied, 220
\gtrsymDiedonbirthday, 219
\gtrsymDivorced, 220
\gtrsymEngaged, 220
\gtrsymFemale, 221
\gtrsymFloruit, 221
\gtrsymFuneralurn, 221
\gtrsymKilled, 220
\gtrsymMale, 221
\gtrsymMarried, 220
\gtrsymNeuter, 221
\gtrsymPartnership, 220
\gtrsymStillborn, 219
\gtruselibrary, 13

hide single leg key, 201

id key, 90
id content interpreter key, 148
id prefix key, 92
id suffix key, 92
if image defined key, 187
ignore key, 115
ignore value, 159
ignore family database key, 205
ignore level key, 117
ignore node key, 116
ignore subtree key, 116
image key, 156
image prefix key, 187
info separators key, 185
input grammar, 29, 72
insert grammar, 73
insert after family key, 119
insert after node key, 118
insert at begin family key, 120
insert at end family key, 121
italian value, 225, 295

JU value, 160

kekule key, 156
Keys
    /gtr/
      add child, 207
      add parent, 208
      adjust node, 111
      adjust position, 110
      after parser, 109
      after tree, 113
      box, 96
      box clear, 96
      calendar print, 176
      calendar text for, 176
      child distance, 85
      child distance in child graph, 85
      child distance in parent graph, 84
      code, 125
      comment code, 183
      content interpreter, 145
      content interpreter code, 146
      content interpreter content, 146
      content interpreter id and content, 149
      database content interpreter, 148
      database format, 162
      database unknown key, 159
      date code, 175
      date format, 174
      date range after, 177
      date range before, 176
      date range full, 176
      date range separator, 177
      deletion content interpreter, 147
      disconnect, 211
      distance, 94
      edges, 190
      edges for family, 192
      edges for subtree, 192
      empty name text, 173
      event code, 180
      event format, 180
      event text, 180
      extra edges, 212
      extra edges for families, 213
      extra edges for family, 213
      extra edges prepend, 214
      extra edges prepend for families, 215
      extra edges prepend for family, 215
      family, 103
      family box, 97
      family database, 205
      family edges, 191
      family extra edges, 212
      family extra edges prepend, 214
      family label, 204
      female, 99
      further distance, 88
      id, 90
      id content interpreter, 148
      id prefix, 92
      id suffix, 92
      ignore, 115

ignore family database, 205
ignore level, 117
ignore node, 116
ignore subtree, 116
info separators, 185
insert after family, 119
insert after node, 118
insert at begin family, 120
insert at end family, 121
keysfrom, 125
label, 203
label database options, 205
label options, 203
language, 225
level, 107
level distance, 81
level n, 108
level size, 82
list separators, 182
list separators hang, 182
male, 99
name, 172
name code, 173
name font, 173
neuter, 99
nick code, 172
no content interpreter, 147
node, 94
node box, 96
node processor, 128
node size, 83
node size from, 84
options for family, 102
options for node, 93
options for subtree, 105
parent distance, 87
parent distance in child graph, 87
parent distance in parent graph, 86
phantom, 122
phantom*, 124
pivot, 95
pivot shift, 104
place text, 178
pref code, 172
proband level, 109
processing, 128
profession code, 184
remove child, 210
remove parent, 210
reset, 125
set position, 109
show, 243
show family, 245
show id, 243
show level, 244
show number, 244
show type, 245
subtree, 106
subtree box, 97
subtree edges, 192
subtree label, 204
surn code, 172
symbols record reset, 222
tcbset, 113
template, 247
tikz, 101
tikzpicture, 112
tikzset, 112
timeflow, 78
tree offset, 109
turn, 98
use family database, 205
/gtr/database/
baptism, 157
baptism+, 157
baptism-, 157
birth, 157
birth+, 157
birth-, 157
burial, 159
burial+, 159
burial-, 159
comment, 155
death, 158
death+, 158
death-, 158
divorce, 158
divorce+, 158
divorce-, 158
engagement, 158
engagement+, 158
engagement-, 158
female, 155
floruit, 158
floruit+, 158
floruit-, 158
image, 156
kekule, 156
male, 155
marriage, 158
marriage+, 158
marriage-, 158
name, 155
neuter, 155
profession, 156
relationship, 156
sex, 155
shortname, 155
uuid, 156
/gtr/debug/
contour, 241
family id, 240
family number, 240
/gtr/edge/
anchoring, 200
background, 200
custom, 197
foreground, 199

hide single leg, 201
mesh, 196
no background, 200
no foreground, 199
none, 198
perpendicular, 194
rounded, 194
swing, 195
xshift, 201
yshift, 202
/gtr/event prefix/
baptism, 179
birth, 179
burial, 180
death, 180
divorce, 180
engagement, 179
floruit, 180
marriage, 179
/gtr/event prefix/birth/
died, 179
out of wedlock, 179
stillborn, 179
/gtr/event prefix/burial/
cremated, 180
/gtr/event prefix/death/
killed, 180
/gtr/event prefix/marriage/
other, 180
/gtr/library/
all, 13
debug, 13
templates, 13
/gtr/symlang/
Baptized, 224
Born, 224
Bornoutofwedlock, 224
Buried, 224
Died, 224
Diedonbirthday, 224
Divorced, 224
Engaged, 224
Female, 224
Floruit, 224
Funeralurn, 224
Killed, 224
Male, 224
Married, 224
Neuter, 224
Partnership, 224
Stillborn, 224
/tcb/
female, 99
gtrNodeDimensions, 129, 133, 136, 139
gtrNodeDimensionsLandscape, 129, 133, 136, 139
if image defined, 187
image prefix, 187
male, 99
neuter, 99
/tikz/
fit to family, 114
fit to subtree, 114
genealogytree edges scope, 193
genealogytree extra edges scope, 216
gtrNodeDimensions, 142
gtrNodeDimensionsLandscape, 142
keysfrom key, 125
Killed key, 224
killed key, 180
killed value, 158

label key, 203
label database options key, 205
label options key, 203
language key, 225
left value, 78, 98, 110, 111, 262, 265, 268, 270, 272
level key, 107
level distance key, 81
level n key, 108
level size key, 82
list separators key, 182
list separators hang key, 182

m-d-yyyy value, 175
m.d.yyyy value, 175
m/d/yyyy value, 175
Male key, 224
male key, 99, 155
male value, 155, 186
marriage key, 158, 179
marriage value, 162, 205
marriage+ key, 158
marriage- key, 158
Married key, 224
medium value, 162, 262, 265
medium marriage above value, 162
medium marriage below value, 162
medium no marriage value, 162, 267
mesh key, 196
mm-dd-yyyy value, 175
mm.dd.yyyy value, 175
mm/dd/yyyy value, 175
mon d yyyy value, 175
mon.d yyyy value, 175
month d yyyy value, 175

name key, 155, 172
name value, 162
name code key, 173
name font key, 173
Neuter key, 224
neuter key, 99, 155
neuter value, 155
\nick, 172
nick code key, 172
no background key, 200

no content interpreter key, 147
no foreground key, 199
\node, 128, 142
node key, 94
node box key, 96
node processor key, 128
node size key, 83
node size from key, 84
none key, 198
none value, 95, 176, 241

off value, 98
options for family key, 102
options for node key, 93
options for subtree key, 105
other key, 180
other value, 158
out of wedlock key, 179
out of wedlock value, 157

p grammar, 17, 19, 21, 32, 55, 59, 60, 62, 71, 75, 76, 115, 260, 268, 270, 272, 278
parent grammar, 16, 17, 19, 20, 34, 55, 60, 62, 63, 69, 72, 75–78, 84, 86, 103, 104, 106, 205, 268, 270, 272, 278
parent value, 95, 211
parent distance key, 87
parent distance in child graph key, 87
parent distance in parent graph key, 86
Partnership key, 224
periphery value, 200
perpendicular key, 194
phantom key, 122
phantom* key, 124
pivot key, 95
pivot shift key, 104
place text key, 178
\pref, 172
pref code key, 172
prefix date value, 180
prefix date place value, 180
proband level key, 109
processing key, 128
profession key, 156
profession code key, 184

relationship key, 156
remove child key, 210
remove parent key, 210
reset key, 125
right value, 78, 98, 110, 111, 250
rounded key, 194
rounded value, 41

sandclock grammar, 34, 39, 55, 60, 62, 69, 104, 278
save value, 159
set position key, 109
sex key, 155
short value, 162, 172
short marriage above value, 162
short marriage below value, 162
short no marriage value, 162, 257, 260
shortname key, 155
show key, 243
show family key, 245
show id key, 243
show level key, 244
show number key, 244
show type key, 245
signpost value, 248
Stillborn key, 224
stillborn key, 179
stillborn value, 157
subtree key, 106
subtree box key, 97
subtree edges key, 192
subtree label key, 204
\surn, 172
surn code key, 172
swing key, 195
symbol value, 162
symbol nodes value, 248
symbols record reset key, 222

\tcbEXTERNALIZE, 49
\tcbox, 128, 136, 139
tcbox value, 128, 136
tcbox* value, 128, 139, 247–250
\tcboxfit, 128, 129, 151
tcbset key, 113
tcolorbox environment, 128, 133
tcolorbox value, 128, 133
template key, 247
templates key, 13
tikz key, 101
tikznode value, 101, 128, 142
tikzpicture environment, 55, 57
tikzpicture key, 112
tikzset key, 112
timeflow key, 78
tiny boxes value, 249
tiny circles value, 249
tree offset key, 109
turn key, 98
typical value, 174, 296

union grammar, 26, 27, 55, 62, 65, 67, 104, 211, 239, 245, 278–280, 282
up value, 78
upsidedown value, 98
use family database key, 205
uuid key, 156

Values
  above, 110, 111
  AD, 160
  ahnentafel 3, 268
  ahnentafel 4, 270
  ahnentafel 5, 272

all, 176
all but AD, 160, 176
BC, 160
below, 110, 111
both, 95, 211, 241
ca, 160
caAD, 160
caBC, 160
center, 200
child, 95, 211
cremated, 159
d M yyyy, 174
d mon yyyy, 174
d mon.yyyy, 174, 296
d month yyyy, 174
d-M-yyyy, 175
d-m-yyyy, 174
d-mon-yyyy, 175
d-month-yyyy, 175
d.M.yyyy, 174
d.m.yyyy, 174
d.mon.yyyy, 174
d.month yyyy, 174
d/M/yyyy, 174
d/m/yyyy, 174
d/mon/yyyy, 174
d/month/yyyy, 174
danish, 225, 295
database, 35, 128, 151, 251, 254, 257, 260, 262, 265, 267, 268, 270, 272
database pole, 251, 253, 254
database pole reduced, 253, 295
database poleportrait, 254, 256, 295
database poleportrait reduced, 256, 295
database portrait, 257, 259
database portrait reduced, 259, 295
database relationship, 267, 296
database sideways, 262, 264, 265, 296
database sideways reduced, 264, 295
database sidewaysportait, 295
database sidewaysportait reduced, 295
database sidewaysportrait, 265, 266
database sidewaysportrait reduced, 266
database traditional, 260, 261
database traditional reduced, 261, 295
date, 180
dd mm yyyy, 174
dd mon yyyy, 174
dd mon.yyyy, 174, 296
dd-mm-yyyy, 174
dd-mon-yyyy, 175
dd-month-yyyy, 175
dd.mm.yyyy, 174
dd.mon.yyyy, 174
dd/mm yyyy, 174, 296
dd/mm/yyyy, 174
dd/mon/yyyy, 174
dd/month/yyyy, 174
ddmonyyyy, 175
died, 157
directory, 250
down, 78, 197, 260, 267
dutch, 225, 295
east, 241
empty, 162
english, 223, 225
error, 159
female, 155, 186
fit, 128, 129, 248
formal graph, 247
french, 225, 295
full, 162, 172
full marriage above, 162, 251, 254
full marriage below, 162
full no marriage, 162
german, 224, 225
german-austrian, 225
german-german, 225
GR, 160
ignore, 159
italian, 225, 295
JU, 160
killed, 158
left, 78, 98, 110, 111, 262, 265, 268, 270, 272
m-d-yyyy, 175
m.d.yyyy, 175
m/d/yyyy, 175
male, 155, 186
marriage, 162, 205
medium, 162, 262, 265
medium marriage above, 162
medium marriage below, 162
medium no marriage, 162, 267
mm-dd-yyyy, 175
mm.dd.yyyy, 175
mm/dd/yyyy, 175
mon d yyyy, 175
mon.d yyyy, 175
month d yyyy, 175
name, 162
neuter, 155
none, 95, 176, 241
off, 98
other, 158
out of wedlock, 157
parent, 95, 211
periphery, 200
prefix date, 180
prefix date place, 180
right, 78, 98, 110, 111, 250
rounded, 41
save, 159
short, 162, 172
short marriage above, 162
short marriage below, 162

short no marriage, 162, 257, 260
signpost, 248
stillborn, 157
symbol, 162
symbol nodes, 248
tcbox, 128, 136
tcbox*, 128, 139, 247–250
tcolorbox, 128, 133
tikznode, 101, 128, 142
tiny boxes, 249
tiny circles, 249
typical, 174, 296
up, 78
upsidedown, 98
warn, 159
west, 241
yyyy, 175
yyyy mm dd, 175
yyyy mon dd, 175
yyyy month d, 175
yyyy-mm-dd, 175
yyyy-mon-d, 175
yyyy-mon-dd, 175
yyyy.m.d, 175
yyyy.M.d., 175
yyyy.m.d., 175
yyyy.mm.dd, 175
yyyy.mon.d., 175
yyyy.month d., 175
yyyy/m/d, 175
yyyy/mm/dd, 175
yyyymmdd, 175, 296
yyyymondd, 175

warn value, 159
west value, 241

xshift key, 201

yshift key, 202
yyyy value, 175
yyyy mm dd value, 175
yyyy mon dd value, 175
yyyy month d value, 175
yyyy-mm-dd value, 175
yyyy-mon-d value, 175
yyyy-mon-dd value, 175
yyyy.m.d value, 175
yyyy.M.d. value, 175
yyyy.m.d. value, 175
yyyy.mm.dd value, 175
yyyy.mon.d. value, 175
yyyy.month d. value, 175
yyyy/m/d value, 175
yyyy/mm/dd value, 175
yyyymmdd value, 175, 296
yyyymondd value, 175