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 \texttt{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.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.3</td>
<td>Formatting the Node Content</td>
<td>36</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Adding Images</td>
<td>38</td>
</tr>
<tr>
<td>2.3.5</td>
<td>Full Example with Frame</td>
<td>39</td>
</tr>
<tr>
<td>2.4</td>
<td>Tutorial: Descendants of the Grandparents (Connecting Trees)</td>
<td>43</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Descendants of the Two Grandparents</td>
<td>43</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Connected Diagram</td>
<td>44</td>
</tr>
<tr>
<td>2.5</td>
<td>Tutorial: Multi-Ancestors</td>
<td>46</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Triple Ancestor Example</td>
<td>46</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Adding Edges Manually</td>
<td>47</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Manual Position Adjustments</td>
<td>48</td>
</tr>
<tr>
<td>2.6</td>
<td>Tutorial: Externalization</td>
<td>49</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Externalization Process</td>
<td>49</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Document Setup</td>
<td>49</td>
</tr>
<tr>
<td>2.6.3</td>
<td>Marking Diagrams for Externalization</td>
<td>50</td>
</tr>
<tr>
<td>2.7</td>
<td>Tutorial: Conversion to Pixel Images</td>
<td>52</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Command Line Conversion with Ghostscript</td>
<td>52</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Command Line Conversion with ImageMagick</td>
<td>53</td>
</tr>
<tr>
<td>2.7.3</td>
<td>Conversion with the ‘standalone’ Package</td>
<td>53</td>
</tr>
<tr>
<td>2.7.4</td>
<td>Conversion during Externalization</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>Genealogy Tree Macros</td>
<td>55</td>
</tr>
<tr>
<td>3.1</td>
<td>Creating a Genealogy Tree</td>
<td>55</td>
</tr>
<tr>
<td>3.2</td>
<td>Using Tree Options</td>
<td>58</td>
</tr>
<tr>
<td>3.3</td>
<td>Accessing Information inside Nodes</td>
<td>59</td>
</tr>
<tr>
<td>4</td>
<td>Graph Grammar</td>
<td>61</td>
</tr>
<tr>
<td>4.1</td>
<td>Graph Structure</td>
<td>61</td>
</tr>
<tr>
<td>4.2</td>
<td>Subgraph 'parent'</td>
<td>63</td>
</tr>
<tr>
<td>4.3</td>
<td>Subgraph 'child'</td>
<td>65</td>
</tr>
<tr>
<td>4.4</td>
<td>Subgraph 'union'</td>
<td>67</td>
</tr>
<tr>
<td>4.5</td>
<td>Subgraph 'sandclock'</td>
<td>69</td>
</tr>
<tr>
<td>4.6</td>
<td>Node 'c'</td>
<td>71</td>
</tr>
<tr>
<td>4.7</td>
<td>Node 'p'</td>
<td>71</td>
</tr>
<tr>
<td>4.8</td>
<td>Node 'g'</td>
<td>71</td>
</tr>
<tr>
<td>4.9</td>
<td>Data 'input'</td>
<td>72</td>
</tr>
<tr>
<td>4.10</td>
<td>Control Sequence 'insert'</td>
<td>73</td>
</tr>
<tr>
<td>5</td>
<td>Option Setting</td>
<td>75</td>
</tr>
<tr>
<td>5.1</td>
<td>Option Priorities</td>
<td>75</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Option Priorities for Nodes</td>
<td>76</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Option Priorities for Families</td>
<td>77</td>
</tr>
<tr>
<td>5.2</td>
<td>Graph Growth Setting (Time Flow)</td>
<td>78</td>
</tr>
<tr>
<td>5.3</td>
<td>Graph Geometry</td>
<td>81</td>
</tr>
<tr>
<td>5.4</td>
<td>Identifiers</td>
<td>90</td>
</tr>
<tr>
<td>5.5</td>
<td>Node Options</td>
<td>93</td>
</tr>
<tr>
<td>5.6</td>
<td>Family Options</td>
<td>102</td>
</tr>
<tr>
<td>5.7</td>
<td>Subtree Options</td>
<td>105</td>
</tr>
<tr>
<td>5.8</td>
<td>Level Options</td>
<td>107</td>
</tr>
<tr>
<td>5.9</td>
<td>Tree Positioning Options</td>
<td>109</td>
</tr>
<tr>
<td>5.10</td>
<td>TikZ and Tcolorbox Options</td>
<td>112</td>
</tr>
<tr>
<td>5.11</td>
<td>Ignoring Input</td>
<td>115</td>
</tr>
<tr>
<td>5.12</td>
<td>Inserting Input</td>
<td>118</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>5.13</td>
<td>Phantom Nodes and Subtrees</td>
<td>122</td>
</tr>
<tr>
<td>5.14</td>
<td>Special and Auxiliary Options</td>
<td>125</td>
</tr>
<tr>
<td>6</td>
<td>Node Data (Content) Processing</td>
<td>127</td>
</tr>
<tr>
<td>6.1</td>
<td>Setting a Node Data Processing and Processor</td>
<td>128</td>
</tr>
<tr>
<td>6.2</td>
<td>Predefined Non-Interpreting Processings</td>
<td>129</td>
</tr>
<tr>
<td>6.2.1</td>
<td>fit</td>
<td>129</td>
</tr>
<tr>
<td>6.2.2</td>
<td>tcolorbox</td>
<td>133</td>
</tr>
<tr>
<td>6.2.3</td>
<td>tcbox</td>
<td>136</td>
</tr>
<tr>
<td>6.2.4</td>
<td>tcbox*</td>
<td>139</td>
</tr>
<tr>
<td>6.2.5</td>
<td>tikznod</td>
<td>142</td>
</tr>
<tr>
<td>6.3</td>
<td>Creating a Customized Non-Interpreting Processor</td>
<td>144</td>
</tr>
<tr>
<td>6.4</td>
<td>Content Interpreters</td>
<td>145</td>
</tr>
<tr>
<td>7</td>
<td>Database Processing</td>
<td>151</td>
</tr>
<tr>
<td>7.1</td>
<td>Database Concept</td>
<td>152</td>
</tr>
<tr>
<td>7.2</td>
<td>Example Settings</td>
<td>153</td>
</tr>
<tr>
<td>7.3</td>
<td>Data Keys</td>
<td>155</td>
</tr>
<tr>
<td>7.4</td>
<td>Input Format for Dates</td>
<td>161</td>
</tr>
<tr>
<td>7.5</td>
<td>Formatting the Node Data</td>
<td>163</td>
</tr>
<tr>
<td>7.6</td>
<td>Formatting Names</td>
<td>173</td>
</tr>
<tr>
<td>7.7</td>
<td>Formatting Dates</td>
<td>175</td>
</tr>
<tr>
<td>7.8</td>
<td>Formatting Places</td>
<td>179</td>
</tr>
<tr>
<td>7.9</td>
<td>Formatting Events</td>
<td>180</td>
</tr>
<tr>
<td>7.10</td>
<td>Formatting Lists of Events</td>
<td>182</td>
</tr>
<tr>
<td>7.11</td>
<td>Formatting Comments</td>
<td>184</td>
</tr>
<tr>
<td>7.12</td>
<td>Formatting Professions</td>
<td>185</td>
</tr>
<tr>
<td>7.13</td>
<td>Formatting Lists of Information</td>
<td>186</td>
</tr>
<tr>
<td>7.14</td>
<td>Formatting Sex</td>
<td>187</td>
</tr>
<tr>
<td>7.15</td>
<td>Formatting Images</td>
<td>188</td>
</tr>
<tr>
<td>8</td>
<td>Edges</td>
<td>191</td>
</tr>
<tr>
<td>8.1</td>
<td>Edge Settings</td>
<td>192</td>
</tr>
<tr>
<td>8.2</td>
<td>Edge Types</td>
<td>196</td>
</tr>
<tr>
<td>8.3</td>
<td>Edge Parameters</td>
<td>201</td>
</tr>
<tr>
<td>8.4</td>
<td>Edge Labels</td>
<td>205</td>
</tr>
<tr>
<td>8.5</td>
<td>Edge Labels Database</td>
<td>207</td>
</tr>
<tr>
<td>8.6</td>
<td>Adding and Removing Nodes from Edge Drawing</td>
<td>209</td>
</tr>
<tr>
<td>8.7</td>
<td>Extra Edges</td>
<td>214</td>
</tr>
<tr>
<td>8.8</td>
<td>Edge Shifting</td>
<td>219</td>
</tr>
<tr>
<td>9</td>
<td>Genealogy Symbols</td>
<td>225</td>
</tr>
<tr>
<td>9.1</td>
<td>Symbol Color Settings</td>
<td>225</td>
</tr>
<tr>
<td>9.1.1</td>
<td>Global Color Settings</td>
<td>225</td>
</tr>
<tr>
<td>9.1.2</td>
<td>Local Color Settings</td>
<td>226</td>
</tr>
<tr>
<td>9.2</td>
<td>List of Symbols</td>
<td>227</td>
</tr>
<tr>
<td>9.3</td>
<td>Legend to Symbols</td>
<td>230</td>
</tr>
<tr>
<td>9.3.1</td>
<td>Printing a Legend</td>
<td>230</td>
</tr>
<tr>
<td>9.3.2</td>
<td>Description Texts and Language Settings</td>
<td>231</td>
</tr>
<tr>
<td>10</td>
<td>Language and Text Settings</td>
<td>233</td>
</tr>
<tr>
<td>10.1</td>
<td>Preamble Settings</td>
<td>233</td>
</tr>
</tbody>
</table>
11 Debugging: Library  
11.1 Parser Debugging  
11.2 Processor Debugging  
11.3 Graphical Debugging  
11.4 Show Information  
12 Templates: Library  
12.1 Using Templates  
12.2 Template 'formal graph'  
12.3 Template 'signpost'  
12.4 Template 'symbol nodes'  
12.5 Template 'tiny boxes'  
12.6 Template 'tiny circles'  
12.7 Template 'directory'  
12.8 Template 'database pole'  
12.9 Template 'database pole reduced'  
12.10 Template 'database poleportrait'  
12.11 Template 'database poleportrait reduced'  
12.12 Template 'database portrait'  
12.13 Template 'database portrait reduced'  
12.14 Template 'database traditional'  
12.15 Template 'database traditional reduced'  
12.16 Template 'database sideways'  
12.17 Template 'database sideways reduced'  
12.18 Template 'database sidewaysportrait'  
12.19 Template 'database sidewaysportrait reduced'  
12.20 Template 'database relationship'  
12.21 Template 'ahnentafel 3'  
12.22 Template 'ahnentafel 4'  
12.23 Template 'ahnentafel 5'  
12.24 Predefined Colors of the Library  
12.25 Auxiliary Control Sequences  
13 Auto-Layout Algorithm  
13.1 Preliminaries  
13.1.1 Aesthetic Properties  
13.1.2 Genealogy Trees  
13.1.3 Graph Grammar  
13.2 Requirements  
13.2.1 Parent and Child Alignment  
13.2.2 Patchwork Families  
13.2.3 Graph Growing Direction  
13.3 Algorithmic Steps  
13.3.1 Recursive Family and Node Placement  
13.3.2 Contours  
13.3.3 Combining Subtrees  
13.4 Known Problems  
14 Example Graph Files  
14.1 example.option.graph
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 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.

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 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 295 is used numerous times inside this document for different diagrams.

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 TikZ [4] and \texttt{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 191.

• 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 TikZ 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\textsuperscript{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.

\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

 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 235.

/gtr/library/templates

 Loads additional code for templates. These are styles to set various options by one key; see Chapter 12 on page 255.

/gtr/library/all

 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 .

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 255, 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 TikZ \cite{tikz} and tcolorbox \cite{tcolorbox}, since \genealogytree is based on both.
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:

```latex
\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\kZ \cite{tikz}[4] package. \texttt{\genealogytree} \textsuperscript{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 \texttt{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 \textit{\texttt{/gtr/template}} \textsuperscript{P.255}. We put this to the option list of \texttt{\genealogytree} \textsuperscript{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 \texttt{/gtr/male} \textsuperscript{P.99} and \texttt{/gtr/female} \textsuperscript{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}
  }
  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 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. The options inside /gtr/box 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 or inside the option list of \genealogytree.

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

Now, highlight can be used to apply /gtr/pivot and /gtr/box settings with one key word:

\%
\%... g[highlight,female]{first child}
\%
2.1.6 Changing the Timeflow

A genealogy tree may be grown in four directions depending on the given \textit{/gtr/timeflow} \textsuperscript{P.78}. Now, we will let the time flow to the left. Additionally, we replace the \textit{/gtr/template} \textsuperscript{P.255} setting by individual settings for \textit{/gtr/processing} \textsuperscript{P.128}, \textit{/gtr/level size} \textsuperscript{P.82}, \textit{/gtr/node size from} \textsuperscript{P.84}, and \textit{/gtr/box} \textsuperscript{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}},
]

\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}},
]

\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}},
]

\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}},
]
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 is used for quick setup.

\begin{tikzpicture}
genealogytree{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 values.

\begin{tikzpicture}
genealogytree{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 values for later reference.

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

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`:

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` and `/gtr/edge/background` (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=n11,female]{i_1}
c[id=n12,female]{i_2}
c[id=n13,female]{i_3}
c[id=n14,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 297. That is everything inside \texttt{\genealogytree} \textsuperscript{P.55} without the options of \texttt{\genealogytree} \textsuperscript{P.55}. Using the \texttt{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 /\texttt{gtr/id} \textsuperscript{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 /\texttt{gtr/id} \textsuperscript{P.90} value, the /\texttt{gtr/show id} \textsuperscript{P.251} setting from the \texttt{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 \( ne_3, nb_3, na_3, na_1 \) and \( na_2, na_4, nd_4, ni_2 \). 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 \texttt{genealogytree} without changing the diagram data directly.

1. \texttt{/gtr/box} sets options to wash out all nodes.
2. \texttt{/gtr/edges} sets options to wash out all edges.
3. \texttt{/gtr/options for node} sets box options to all nodes along the selected path to display them emphasized.
4. \texttt{/gtr/extra edges for families} 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` values through `/gtr/options for node` values through `/gtr/options for node` values through `/gtr/options for node`.

All given `/gtr/id` values are also TikZ nodes. Therefore, a genealogy tree can easily be annotated and extended by TikZ instructions.
2.2.5 Coloring Subtrees

For the given example data, the descendants of the root family should now be 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 $\text{fam}_B$ is colored in red by `/gtr/options for subtree` P.105. Analogously, $\text{fam}_C$ 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={\text{fam}_B}{box={colback=red!20!white}},
\options for subtree={\text{fam}_C,\text{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.253 from the \texttt{lib} 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={\text{fam}_B}{box={colback=red!20!white}},
\options for subtree={\text{fam}_C,\text{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 \texttt{\textbackslash gtrnodetype} or \texttt{\textbackslash grifpnode}. We use this to set up a \texttt{tcolorbox} style \texttt{bleach p} which wash out the in-law nodes, when \texttt{\textbackslash grifpnode} expands to \texttt{(true)}. This style is formulated locally by \texttt{/gtr/tcbset}:

\begin{center}
\begin{verbatim}
\%...
tcbset={bleach p/.code={%
  \gtrifpnode{\tcbset{enhanced jigsaw,opacityback=0.2}}{}
},
%
\}
\end{verbatim}
\end{center}

This gives:

\begin{verbatim}
\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}}
\end{tikzpicture}
\end{verbatim}

\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}}
\end{tikzpicture}

\begin{verbatim}
\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 \texttt{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 \texttt{a}_4};
\end{verbatim}

\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 genealognpicture\textsuperscript{\textsuperscript*P.57} to create genealogy trees. This is a handy combination of tikzpicture and \genealogytree\textsuperscript{\textsuperscript*P.55}.

\\begin{tikzpicture}[x=1.5cm, y=1.5cm]
\node (g) at (0,0) {\text{ proband }};
\end{tikzpicture}

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

\\begin{tikzpicture}[x=1.5cm, y=1.5cm]
\node (g) at (0,0) {\text{ proband }};
\node (p) at (-1,-1) {a};
\node (b) at (-1,-2) {b};
\node (c) at (-1,-3) {c};
\node (A) at (1,-1) {A};
\node (B) at (1,-2) {B};
\draw (g) -- (p);
\draw (p) -- (b);
\draw (p) -- (c);
\draw (g) -- (A);
\draw (g) -- (B);
\end{tikzpicture}
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{Gauß}, 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 \texttt{biblatex} or \texttt{bibtex}.

So, we tell genealogytree that we want to use such a database concept by setting \texttt{/gtr/processing=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 \texttt{/gtr/database format=medium marriage below} to some predefined value. Everything can be customized later.

The basic information for a person is \texttt{/gtr/database/name\texttt{P. 155}}, \texttt{/gtr/database/male\texttt{P. 155}} or \texttt{/gtr/database/female\texttt{P. 155}}, \texttt{/gtr/database/birth\texttt{P. 158}} and \texttt{/gtr/database/death\texttt{P. 159}}.

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

In the example above, we also added a \texttt{/gtr/database/profession\texttt{P. 156}} which appears in the output, and an \texttt{/gtr/database/image\texttt{P. 156}} which is not used. Note the markup with \texttt{\pref\texttt{P. 173}} and \texttt{\surn\texttt{P. 173}} inside the \texttt{/gtr/database/name\texttt{P. 155}} which marks preferred name parts and the surname. There is no name parsing as known from \texttt{bib(la)tex}.

As \texttt{/gtr/id\texttt{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 wives 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 \textsuperscript{P. 83}, /gtr/level size \textsuperscript{P. 82}, and /gtr/level distance \textsuperscript{P. 81} set size and distance values.

With /gtr/box \textsuperscript{P. 96}, we set tcolorbox options for the appearance of the node box. Note that \texttt{\gtrDBsex} is set to male by the database values inside the node content. There are predefined /tcb/male \textsuperscript{P. 99} and /tcb/female \textsuperscript{P. 99} styles, but with /gtr/tcbset \textsuperscript{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 \texttt{\pref\{Carl Friedrich\} \texttt{\surn\{Gau\ss\}})),
    birth={1777-04-30\{Braunschweig (Niedersachsen)\)},
    death={1855-02-23\{Göttingen (Niedersachsen)\)},
    profession={Mathematiker, Astronom, Geodät und Physiker},
    image={Carl_Friedrich_Gauss.jpg},
}
}
\end{genealogypicture}
As second step, we adapt the format of the given data inside the node output.

\begin{genealogypicture}
\begin{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,
\end{processing}
\gtrDBsex
\end{genealogypicture}

With \texttt{/gtr/list separators hang} \textsuperscript{P.183}, the event list is formatted with hanging indent. \texttt{/gtr/name font} \textsuperscript{P.174} and \texttt{/gtr/surn code} \textsuperscript{P.173} are used to format the name of the person. \texttt{/gtr/place text} \textsuperscript{P.179} inserts a \texttt{\newline} before the place of an event is printed in our example. Finally, \texttt{/gtr/date format} \textsuperscript{P.175} is used to change the way dates are printed.

\begin{genealogypicture}
\begin{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},
},
\end{processing}
\sandclock
{child{
  g[id=GauxCarl1777]{
    male,
    name={Johann \textsuperscript{pref}{Carl Friedrich} \textsuperscript{surn}{Gau\ss{}}},
    birth={1777-04-30}{Braunschweig (Niedersachsen)},
    death={1855-02-23}{Gö\öt\ö̈ngen (Niedersachsen)},
    profession={Mathematiker, Astronom, Geodä\ö̈t und Physiker},
    image={Carl_Friedrich_Gauss.jpg},
  }
}
\end{genealogypicture}
2.3.4 Adding Images

The predefined /gtr/database format \( ^{\text{P. 163}} \) options do not consider images. But we can add image code easily to be /gtr/box \( ^{\text{P. 96}} \) definition which accepts tcolorbox settings.

/tcb/if image defined \( ^{\text{P. 188}} \) 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 \(^{\text{1}}\) by the database values inside the node content. Here, it is accessed by /tikz/fill overzoom DBimage \( ^{\text{P. 189}} \).

Options from The tcolorbox package \(^{\text{3}}\) are used to enlarge the box width by 25mm and fill the space with this image:

\begin{genealogypicture}
\begin{itemize}
  \item processing=database,
  \item database format=medium marriage below,
  \item node size=2.4cm,
  \item level size=3.5cm,
  \item level distance=6mm,
  \item list separators hang,
  \item name font=\bfseries,
  \item surn code=\textcolor{red!50!black}{#1},
  \item place text={\newline}{},
  \item date format=d/mon/yyyy,
  \item tcbset={male/.style={colframe=blue,colback=blue!5},
          female/.style={colframe=red,colback=red!5}},
  \item 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 DBimage]
          ([xshift=-24mm]interior.south east) rectangle (interior.north east);
          end[tcbclipinterior]}},
}\end{itemize}
\end{genealogypicture}

Johann Carl Friedrich Gauß
\[30\text{ Apr} / 1777\]
\[\text{Braunschweig (Niedersachsen)}\]
\[\uparrow 23\text{ Feb} / 1855\]
\[\text{Göttingen (Niedersachsen)}\]
\[\text{Mathematiker, Astronom, Geodät und Physiker}\].

\(^{1}\)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»:

```
sandclock{
  child{id=GauaOsth1805}{
    p{id=OsthJoha1780}{
      female,
      name={\text{\textbf{Johanna}} Elisabeth Rosina \textit{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={\text{\textbf{Johann Carl Friedrich}} 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={\text{\textbf{Carl Joseph}} Gau\ss{}},
      birth={1806-08-21}{Braunschweig (Niedersachsen)},
      death={1873-07-04}{Hannover (Niedersachsen)},
    }
    c[id=GauxWilh1808]{
      female,
      name={\text{\textbf{Wilhelmina}} Gau\ss{}},
      birth={1808-02-29}{G"ottingen (Niedersachsen)},
      death={1840-08-12}{T"ubingen (Baden-W"urttemberg)},
    }
    c[id=GauxLudw1809]{
      male,
      name={\text{\textbf{Ludwig}} Gau\ss{}},
      birth={1809-09-10}{G"ottingen (Niedersachsen)},
      death={1810-03-01}{G"ottingen (Niedersachsen)},
    }
  }union{id=GauaWald1810}{
    p{id=WaldFrie1788}{
      female,
      name={\text{\textbf{Friederica}} Wilhelmine 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={\text{\textbf{Eugen Peter Samuel Marius}} Gau\ss{}},
      birth={1811-07-29}{G"ottingen (Niedersachsen)},
      death={1896-07-04}{Columbia (Missouri)},
      profession={Rechtswissenschaftler, Kaufmann},
    }
    c[id=GauxWilh1813]{
      male,
      name={\text{\textbf{Wilhelm August Carl Matthias}} Gau\ss{}},
      birth={1813-10-23}{G"ottingen (Niedersachsen)},
    }
}
```
As final polish, \textit{gtr/edges}^{P.192} are set to be \textit{rounded} and the used symbols are recorded by \textit{gtr/symbols record reset}^{P.230} and displayed by \textit{\textbackslash gtrSymbolsLegend}^{P.231} inside \textit{gtr/after tree}^{P.113}.

Finally, the whole diagram is put into a titled \texttt{tcolorbox} to exhibit the example:

```latex
\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 Gauss (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,
\textit{\textbackslash gtrDBsex},drop fuzzy shadow,
if image defined={add to width=25mm,right=25mm,
underlay={\begin{tcbclipinterior}\path[fill overzoom DBimage]
{(xshift=-24mm)interior.south east} rectangle (interior.north east);\end{tcbclipinterior}}},
edges=rounded,
symbols record reset,
\textit{after tree}={\node[font=\scriptsize\textit{itspace},text width=1.8cm,below left,\fill=white,fill opacity=0.4,text opacity=1]
at (current bounding box.north east) \{\textit{\textbackslash gtrSymbolsLegend}\};},
}]
input{example.gauss.graph}
\end{genealogypicture}
\end{tcolorbox}
```
The Family of Carl Friedrich Gauß (1777–1855)

**Carl Friedrich Gauß**
- **Born**: 30/Apr/1777 (Braunschweig, Niedersachsen)
- **Died**: 23/Feb/1855 (Göttingen, Niedersachsen)
- **Mathematiker, Astronom, Geodät und Physiker**.

**Johanna**
- **Born**: 8/May/1780 (Braunschweig, Niedersachsen)
- **Died**: 24/Nov/1705 (Völkenrode, Niedersachsen)

**Elisabeth**
- **Born**: 11/Oct/1809 (Göttingen, Niedersachsen)
- **Died**: 9/Oct/1805 (Braunschweig, Niedersachsen)

**Osthoff**
- **Born**: ca. 1715 (Völkenrode, Niedersachsen)
- **Died**: 5/Jul/1774 (Braunschweig, Niedersachsen)

**Weißgerberstochter**
- **Born**: ca. 1735 (Völkenrode, Niedersachsen)

**Johann**
- **Born**: 30/Apr/1777 (Braunschweig, Niedersachsen)
- **Died**: 23/Feb/1855 (Göttingen, Niedersachsen)

**Mathematiker, Astronom, Geodät und Physiker**.

**Johanna Elisabeth Rosina Osthoff**
- **Born**: 8/May/1780 (Braunschweig, Niedersachsen)
- **Died**: 11/Oct/1809 (Göttingen, Niedersachsen)

**Johann Carl Friedrich Gauß**
- **Born**: 30/Apr/1777 (Braunschweig, Niedersachsen)
- **Died**: 23/Feb/1855 (Göttingen, Niedersachsen)

**Mathematiker, Astronom, Geodät und Physiker**.

**Dorothea Bentze**
- **Born**: 25/Apr/1776 (Völkenrode, Niedersachsen)

**Karoline Gauß**
- **Born**: 9/Jun/1816 (Göttingen, Niedersachsen)
- **Died**: 11/Feb/1864 (Dresden, Sachsen)

**Eugen Peter Samuel Marius Gauß**
- **Born**: 29/Jul/1811 (Göttingen, Niedersachsen)
- **Died**: 4/Jul/1896 (Columbia, Missouri)

**Rechtswissenschaftler, Kaufmann**.

**Wilhelm August Carl Matthias Gauß**
- **Born**: 23/Oct/1813 (Göttingen, Niedersachsen)
- **Died**: 25/Aug/1879 St. Louis (Missouri)

**Henriette Wilhelmine Karoline Therese Gauß**
- **Born**: 9/Jun/1816 (Göttingen, Niedersachsen)

**=born, =married, =died.**
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 suffix of $c1\Theta a$, because we added an /gtr/id suffix of $\Theta a$ to every id value of our first tree.

\begin{tikzpicture}
\genealogytree
\end{tikzpicture}

The other settings in this example are less important, but one may observe that the /tcb/male and /tcb/female 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$^\text{P.90}$ of c1@b, because we added an /gtr/id suffix$^\text{P.92}$ 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}

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{tree.png}
\end{figure}

\subsection{Connected Diagram}

After the preparations, the \genealogytree$^\text{P.55}$ diagrams can easily be put together.

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

In the first \genealogytree$^\text{P.55}$, one sees a /gtr/phantom$^\ast$\textsuperscript{P.124} option which makes the first family $c_1,\ldots,c_5$ invisible but still space reserving.

Using /gtr/id suffix$^\text{P.92}$ or /gtr/id prefix$^\text{P.92}$ allows to distinguish nodes with the same id value in different trees. Otherwise, the id values would have to be changed manually.
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 ancestors 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).

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

Now, we add the missing connections. For this, `/gtr/id` values are added to all involved nodes and families. Then, the connections are drawn using `/gtr/add parent` 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.

```latex
\begin{genealogypicture}[template=formal graph,timeflow=left,
add parent=a_5 to AB\_fam,
add parent=b_6 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=a_5]\{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=b_6]\{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 \texttt{/gtr/tikz} option with some \texttt{TikZ} shift operations. These final shiftings do not influence the auto-layout algorithm, but the edges move with the nodes.

Alternatively, \texttt{/gtr/distance}, \texttt{/gtr/pivot}, and \texttt{/gtr/pivot shift} can be used to influence the auto-layout algorithm. \texttt{/gtr/pivot shift} 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{genealypicture}[template=formal graph,timeflow=left,
add parent=a5 to AB\_fam,
add parent=b6 to BC\_fam,
]
\[\text{child}{ g[male\{}{X} \ p[female]\{}{Y}
\text{child}{g[male\{}{a_1}
\text{child}{ g[female]\{}{a_2}
\text{child}{g[male]\{}{a_3}
\text{child}{g[male]\{}{a_4}
\text{child}{g[female, id=a5]{a_5}}
})}]
\text{child}{g[male]\{}{b_1}
\text{child}{g[male]\{}{b_2}
\text{child}{g[male]\{}{b_3}
\text{child}{g[male]\{}{b_4}
\text{child}{id=AB\_fam}{g[male]\{}{b_5}
\text{child}{g[female, id=b6, tikz={xshift=-6.5mm, yshift=5.5mm}]{b_6}}}
})]
\text{child}{g[male]\{}{c_1}
\text{child}{g[female]\{}{c_2}
\text{child}{g[male]\{}{c_3}
\text{child}{g[female]\{}{c_4}
\text{child}{id=BC\_fam, pivot shift=-8.25mm}{g[male]\{}{c_5}
\text{child}{g[male]\{}{Z}}
})}]
\]
\end{genealypicture}
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:

- TikZ externalization: Here, the whole original document is compiled in a sophisticated way. The external library of TikZ can automatically externalize all \texttt{tikzpicture} environments, see [4].

- \texttt{tcolorbox} externalization: Here, marked code snippets are compiled in a not so sophisticated but more robust way. The external library of \texttt{tcolorbox} only externalizes marked and named snippets. Besides \texttt{tikzpicture} environments, \texttt{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 \texttt{tcolorbox}. This library is already included by the \texttt{genealogytree} package.

2.6.2 Document Setup

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

\begin{verbatim}
\documentclass{article}
\usepackage[all]{genealogytree}
\tcbEXTERNALIZE
\begin{document}
% ...
\section{My heading}
% ...
Your main document containing texts, diagrams, figures, etc.
% ... 
\end{document}
\end{verbatim}

To use the externalization options, the compiler has to be called with the \texttt{\-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 \texttt{tikzpicture} and the other one the \texttt{genealogypicture} \textsuperscript{P.57} shortcut.

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 \texttt{tikzpicture} by \texttt{extikzpicture} and add a \textit{unique name} as additional parameter.

- Replace \texttt{genealogypicture} \textsuperscript{P.57} by \texttt{exgenealogypicture} \textsuperscript{P.57} and add a \textit{unique name} as additional parameter.

By default, these \textit{unique names} are the names of the external files inside an \texttt{external} subdirectory. Depending on the operation system, the sub-directory \texttt{external} may have to be generated manually.
This is the first externalized example:
\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}

Now follows the second externalized example:
\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:

\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:
\begin{exgenealogypicture}{second_example}[template=tiny boxes]
child{
  g-p-c-
  child{g-p-c-c-}
  child{g-p-c-c-c-}
}
\end{exgenealogypicture}

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:

```
"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 adaptation 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 \texttt{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.

```latex
\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}[]
\parent{
g{male}
\insert{gtrparent4}
}
\end{exgenealogypicture}

\end{document}
```

- If the document above is compiled with the \texttt{-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 adaptation for your system. Here, a 64 bit Windows installation with a specific version of Ghostscript is used.
- Adapt \texttt{-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 \texttt{tcolorbox}. This hack may become useless in the future. Especially, for a single image, the \texttt{standalone} package should be preferred.
3 Genealogy Tree Macros

3.1 Creating a Genealogy Tree

\genealogytree\{\textit{options}\}\{\textit{tree content}\}

This is the main genealogy tree drawing macro of the package. The \textit{tree content} has to obey to the tree grammar rules documented in Chapter 4 on page 61. The \textit{options} control how the drawing is done. These \textit{options} are \texttt{pgf} keys with the key tree path \texttt{/gtr/} and they are described in the following.

The actual drawing is done with help of the Ti\textit{k}Z package. Therefore, every \genealogytree has to be placed into a \texttt{tikzpicture} environment. It is possible to put several \genealogytree macros into the same \texttt{tikzpicture} and interconnect them.

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

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

A \textit{family} is a set of \textit{parent} and \textit{child} nodes which has to contain exactly one \textit{genealogy} node (\texttt{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:

- \textbf{parent}: A \textit{parent} family may contain other \textit{parent} families. Trees with this construction grow into \textit{ancestor direction}.
- \textbf{child}: A \textit{child} family may contain other \textit{child} families or \textit{union} families. Trees with this construction grow into \textit{descendant direction}.
- \textbf{union}: A \textit{union} ties a second child-type family to a \texttt{g} node as parent of this family.
- \textbf{sandclock}: A \textit{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[\langle options \rangle]{\langle file name \rangle}

Uses the content of the file denoted by \langle file name \rangle to create a \genealogytree^P.55 with the given \langle options \rangle. See Section 14.1 on page 295 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\textsuperscript{P.55} inside a \tikzpicture. For ⟨options⟩ and ⟨tree content⟩ see \genealogytree\textsuperscript{P.55}. This environment allows more compact source code, but one cannot combine several trees and adding additional TikZ commands has to be done by /gtr/tikzpicture\textsuperscript{P.112} or /gtr/after tree\textsuperscript{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\textsuperscript{[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\langle\text{options}\rangle

Sets \langle\text{options}\rangle for every following \texttt{\textbackslash genealogytree} inside the current \TeX\ group. These \langle\text{options}\rangle are \texttt{pgf} keys with the key tree path /gtr/ and they are described in the following.

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

\gtrkeys\langle\text{hook}\rangle\langle\text{key list}\rangle

Auxiliary macro which appends a \langle\text{key list}\rangle (options) to a \langle\text{hook}\rangle 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 251 for displaying these values.

\texttt{\textbackslash gtrnodetype}

Holds the node type \texttt{g}, \texttt{p}, or \texttt{c}.

\texttt{\textbackslash gtrnodeid}

Holds the /gtr/id\textsuperscript{\textbackslash P.90} value of the node.

\texttt{\textbackslash gtrnodenumber}

Holds the internal node number.

\texttt{\textbackslash gtrnodefamily}

Holds the internal family number this node belongs to.

\texttt{\textbackslash 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 & \textbackslash gtrnodetype \\
id & \textbackslash gtrnodeid \\
number & \textbackslash gtrnodenumber \\
family & \textbackslash gtrnodefamily \\
level & \textbackslash gtrnodelevel \\
\end{tabular}
}]
\begin{tabular}{@{}rr}
type & p \\
id & Jim \\
number & 5 \\
family & 2 \\
level & 2 \\
\end{tabular}
\begin{tabular}{@{}rr}
type & g \\
id & Bob \\
number & 4 \\
family & 2 \\
level & 2 \\
\end{tabular}
\begin{tabular}{@{}rr}
type & p \\
id & ??? \\
number & 6 \\
family & 2 \\
level & 2 \\
\end{tabular}
\begin{tabular}{@{}rr}
type & p \\
id & ??? \\
number & 7 \\
family & 1 \\
level & 1 \\
\end{tabular}
\end{tikzpicture}

\texttt{\textbackslash gtrifnodeid\{\textbackslash true\}\{\textbackslash false\}}

Expands to \texttt{\langle true\rangle}, if /gtr/id\textsuperscript{\textbackslash P.90} was set, and to \texttt{\langle false\rangle} otherwise.

\texttt{\textbackslash gtrifgnode\{\textbackslash true\}\{\textbackslash false\}}

Expands to \texttt{\langle true\rangle}, if the node type is \texttt{g}, and to \texttt{\langle false\rangle} otherwise.

\texttt{\textbackslash gtrifcnode\{\textbackslash true\}\{\textbackslash false\}}

Expands to \texttt{\langle true\rangle}, if the node type is \texttt{c}, and to \texttt{\langle false\rangle} otherwise.

\texttt{\textbackslash gtrifpnode\{\textbackslash true\}\{\textbackslash false\}}

Expands to \texttt{\langle true\rangle}, if the node type is \texttt{p}, and to \texttt{\langle false\rangle} 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.
## 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`. 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.

```latex
\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}
```
\texttt{\gtrparserdebug} \textsuperscript{P. 236} can help to detect structural errors. Here, we get:

\begin{verbatim}
\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\(parent\)}
  }
}
\end{verbatim}

Genealogytree Parser Debugger

Start: Parent Family 1, Level 1

Child: Individual 1, Family 1, Level 0
Options: id=pB
Content: B\(child\)

Child: Individual 2, Family 1, Level 0
Options: id=pA
Content: A\(proband\)

Child: Individual 3, Family 1, Level 0
Options: id=pC
Content: C\(child\)

Child: Individual 4, Family 1, Level 0
Options: id=pD
Content: D\(child\)

Parent: Individual 5, Family 1, Level 1
Options: id=pE
Content: E\(parent\)

Parent: Individual 6, Family 1, Level 1
Options: id=pF
Content: F\(parent\)

End: Parent Family 1, Level 1
End of Genealogytree Parser Debugger
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**

```latex
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.

```latex
\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}
```
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\(child\)}
}

Genealogytree Parser Debugger
Start: Child Family 1, Level 0
Parent: Individual 1, Family 1, Level 0
Options: id=pA
Content: A\(proband\)

Parent: Individual 2, Family 1, Level 0
Options: id=pB
Content: B\(parent\)

Child: Individual 3, Family 1, Level -1
Options: id=pC
Content: C\(child\)

Child: Individual 4, Family 1, Level -1
Options: id=pD
Content: D\(child\)

Child: Individual 5, Family 1, Level -1
Options: id=pE
Content: E\(child\)

End: Child Family 1, Level 0
End of Genealogytree Parser Debugger
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}
\texttt{\textbackslash gtrparserdebug}\ P. 236 can help to detect structural errors. Here, we get:
### 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⟩} \hspace{1em} \text{optional; zero or many times}
    p[⟨node options⟩] {⟨node content⟩} \hspace{1em} \text{optional; zero or many times}
    child[⟨child options⟩] {⟨subtree content⟩} \hspace{1em} \text{mandatory; one or many times}
    parent[⟨parent options⟩] {⟨subtree content⟩} \hspace{1em} \text{optional; zero or many times}
    input{⟨file name⟩} \hspace{1em} \text{optional; zero or many times}
    insert{⟨csname⟩} \hspace{1em} \text{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 \(\text{(child)}\)}
        child{
            g[id=pA] {A \(\text{(proband)}\)}
            c[id=pa] {a \(\text{(child)}\)}
            c[id=pb] {b \(\text{(child)}\)}
            p[id=pX] {X \(\text{(partner)}\)}
        }
        p[id=pC] {C \(\text{(parent)}\)}
        parent{
            g[id=pD] {D \(\text{(parent)}\)}
            c[id=pE] {E \(\text{(child)}\)}
            p[id=pF] {F \(\text{(parent)}\)}
        }
    }
}
\end{tikzpicture}
Genealogytree Parser Debugger

Start: Sandclock Family 1, Level 1

Child: Individual 1, Family 1, Level 0
Options: id=pB
Content: B\(\text{(child)}\)

Start: Child Family 2, Level 0
Parent: Individual 2, Family 2, Level 0
Options: id=pA
Content: A\(\text{(proband)}\)

Child: Individual 3, Family 2, Level -1
Options: id=pa
Content: a\(\text{(child)}\)

Child: Individual 4, Family 2, Level -1
Options: id=pb
Content: b\(\text{(child)}\)

Parent: Individual 5, Family 2, Level 0
Options: id=pX
Content: X\(\text{(partner)}\)
End: Child Family 2, Level 0

Parent: Individual 6, Family 1, Level 1
Options: id=pC
Content: C\(\text{(parent)}\)
Start: Parent Family 3, Level 2
Child: Individual 7, Family 3, Level 1
Options: id=pD
Content: D\(\text{(parent)}\)
Child: Individual 8, Family 3, Level 1
Options: id=pE
Content: E\(\text{(child)}\)
Parent: Individual 9, Family 3, Level 2
Options: id=pF
Content: F\(\text{(parent)}\)
End: Parent Family 3, Level 2
End: Sandclock Family 1, Level 1
End of Genealogytree Parser Debugger
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_{\gtrownodenumber}\},
]
    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'**

```plaintext
input{(file name)}
```

The following example reads a `parent` subgraph from a file. See Section 14.1 on page 295 for the file contents.

```latex
\begin{tikzpicture}
genealogytree[template=signpost]
{
parent{
g{Puppy}
input{example.option.graph}
p{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 \texttt{\mytest}:

```latex
\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}
```
Option Setting

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

Some of the following examples use a standard graph file which is documented in Section 14.1 on page 295.

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 \texttt{pgf} key-value syntax:

- as parameter of \texttt{gtrset P.58} for setting (global) options,
- as optional parameter of \texttt{genealogytree P.55} or \texttt{genealogypicture P.57},
- as optional parameter of a family identifier like \texttt{parent} or \texttt{child},
- as optional parameter of a node identifier like \texttt{g}, \texttt{p}, or \texttt{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 \texttt{/gtr/pivot P.95}, an option setting with higher priority overwrites an option setting with lower priority.
- For options like \texttt{/gtr/box P.96}, an option setting with higher appends to an option setting with lower priority. Thus, \texttt{/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

```latex
\gtrset{
  ...
  \genealogytree[
    options for node={A}{box={...}},
    options for family={fam_a}{box={...}},
    options for subtree={fam_a}{box={...}},
    level 2/.style={
      node={box={...}},
      family={box={...}},
      subtree={box={...}},
    },
    level/.code={\ifnum#1=2\relax
      \gtrset{
        node={box={...}},
        family={box={...}},
        subtree={box={...}}\fi},
  box={...},
  ]%}
%
{...
  ...
  parent[ id=fam_a, % family with id 'fam_a'
    family={box={...}},
    subtree={box={...}},
  ]
  { ...
    ...
    p[id=A, % node with id 'A'
      box={...}]\{A\}
    ...
    ...
}
```

The priorities for options regarding nodes:

1. `/gtr/options for node` has the highest priority. The node has to be identified by a given `/gtr/id`. `/gtr/options for node` should be given using `\gtrset` or as option of `\genealogytree`.
2. Optional parameter of a node identifier like `g`, `p`, or `c`.
3. Option `/gtr/node` inside `/gtr/level n`.
4. Option `/gtr/node` inside `/gtr/level 1`.
5. `/gtr/options for family`; the family has to be identified by a given `/gtr/id`.
6. `/gtr/family` as optional parameter of a family identifier like `parent` or `child`.
7. Option `/gtr/family` inside `/gtr/level n`.
8. Option `/gtr/family` inside `/gtr/level 1`.
9. `/gtr/options for subtree`; the subtree has to be identified by a given `/gtr/id`.
10. `/gtr/subtree` as optional parameter of a family identifier like `parent` or `child`.
11. Option `/gtr/subtree` inside `/gtr/level n`.
13. Setting as parameter of `\genealogytree` or `\gtrset` has the lowest priority.
5.1.2 Option Priorities for Families

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

\begin{verbatim}
\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={...}}}
  \fi},
  edges={...}, % priority (9) lowest
  ...
}
\end{verbatim}

The priorities for options regarding families

1. \texttt{\gtr/options for family} \cite{P.102} has the highest priority. The family has to be identified by a given \texttt{\gtr/id} \cite{P.90}. \texttt{\gtr/options for family} \cite{P.102} should be given using \texttt{\gtrset} \cite{P.58} or as option of \texttt{\genealogytree} \cite{P.55}.

2. Optional \texttt{\gtr/family} \cite{P.103} parameter of a family identifier like \texttt{parent} or \texttt{child}.

3. Option \texttt{\gtr/family} \cite{P.103} inside \texttt{\gtr/level} \cite{P.108}.

4. Option \texttt{\gtr/family} \cite{P.103} inside \texttt{\gtr/level} \cite{P.107}.

5. \texttt{\gtr/options for subtree} \cite{P.105}; the subtree has to be identified by a given \texttt{\gtr/id} \cite{P.90}.

6. Optional \texttt{\gtr/subtree} \cite{P.106} parameter of a family identifier like \texttt{parent} or \texttt{child}.

7. Option \texttt{\gtr/subtree} \cite{P.106} inside \texttt{\gtr/level} \cite{P.108}.

8. Option \texttt{\gtr/subtree} \cite{P.106} inside \texttt{\gtr/level} \cite{P.107}.

9. Setting as parameter of \texttt{\genealogytree} \cite{P.55} or \texttt{\gtrset} \cite{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.

\[ \text{/gtr/timeflow=\langle direction \rangle} \]

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 \langle direction \rangle is set to down, a child graph will grow down, but a parent graph will grow up. Feasible values are:

- down
- up
- left
- right

See Section 14.1 on page 295 for the included example graph file.
\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}

\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}
\begin{tikzpicture}
\genealogytree[template=signpost, timeflow=right, node size=1.2cm, level size=3cm]
\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 \(^{\text{P.107}}\) separately. Inside a level, they are fixed.

\(\text{/gtr/level distance} = \langle \text{length} \rangle\) (no default, initially 5\text{mm})

The given \(\langle \text{length} \rangle\) defines the distance between two following generations. This distance can be set in dependency of the /gtr/level \(^{\text{P.107}}\).

The \(\text{/gtr/level distance}\) can be specified for individual level numbers, e.g.

\[
\text{\gtrset} \{ \\
\quad \text{level 0/.style={level distance=5mm}}, \\
\quad \text{level -1/.style={level distance=10mm}} \\
\} \ldots
\]

---

---
The given \( \langle \text{length} \rangle \) defines one dimension of a node.

- If \( /gtr/timeflow \uparrow \) is \text{up} or \text{down}, then \( /gtr/level \text{ size} \) sets the height of a node.
- If \( /gtr/timeflow \downarrow \) is \text{left} or \text{right}, then \( /gtr/level \text{ size} \) sets the width of a node.

The \( /gtr/level \text{ size} \) be set in dependency of the \( /gtr/level \uparrow \).

Some actual node implementations may not respect the given \( /gtr/level \text{ size} \). Note that the placement algorithm ignores deviations and assumes that the restrictions hold.
The given \texttt{(length)} defines one dimension of a node.

- If \texttt{/gtr/timeflow} is \texttt{up} or \texttt{down}, then \texttt{/gtr/level size} sets the width of a node.
- If \texttt{/gtr/timeflow} is \texttt{left} or \texttt{right}, then \texttt{/gtr/level size} sets the height of a node.

The \texttt{/gtr/node size} can be set in dependency of the \texttt{/gtr/level}. Note that the \texttt{/gtr/node size} may be ignored by nodes boxes which set the width individually or depending from the content width.

If the size should be changed for an individual node, use \texttt{/gtr/box} instead of \texttt{/gtr/node size}:

Some actual node implementations may not respect the given \texttt{/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 \textsuperscript{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 \textsuperscript{P.107}. For an individual node, this distance can be overruled by setting
/gtr/distance \textsuperscript{P.94}.

```latex
\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}
  }
}
...
```

```
\genealogytree
```

```
child distance\n```

```
P_1  P_2
   /\  \\
  C_1 A   B
```

child distance
The given \textit{\{length\}} defines the minimum distance of two children of a family in a child graph. The \textit{/gtr/child distance in child graph} can be set in dependency of the \textit{/gtr/level} \textsuperscript{P.107}. For an individual node, this distance can be overruled by setting \textit{/gtr/distance} \textsuperscript{P.94}.

\begin{verbatim}
\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}
    }
  }
\end{verbatim}

\begin{verbatim}
/gtr/child distance=(length) (no default, style)
This is an abbreviation for setting \textit{/gtr/child distance in parent graph} \textsuperscript{P.84} and \textit{/gtr/child distance in child graph} to the same \textit{\{length\}}.
The given \((\text{length})\) 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[\text{id}=A]\{A\} \\
p[\text{id}=B]\{B\} \\
g\{C_1\} \\
c\{C_2\} \\
c\{C_3\} \\
} \\
} \\
...
The given \( \langle \text{length} \rangle \) defines the minimum distance of two parents of a family in a \textit{child} graph. The \textit{/gtr/parent distance in parent graph} \textsuperscript{P.86} can be set in dependency of the \textit{/gtr/level} \textsuperscript{P.107}. For an individual node, this distance can be overruled by setting \textit{/gtr/distance} \textsuperscript{P.94}.

\[
\begin{align*}
\text{parent distance in child graph} &= \langle \text{length} \rangle \\
\text{parent distance in parent graph} &= \langle \text{length} \rangle
\end{align*}
\]

This is an abbreviation for setting \textit{/gtr/parent distance in parent graph} \textsuperscript{P.86} and \textit{/gtr/parent distance in child graph} to the same \( \langle \text{length} \rangle \).
The given \textit{length} defines the minimum distance of two nodes which are not parents or children of the same family. The \texttt{/gtr/further distance} can be set in dependency of the \texttt{/gtr/level}\footnote{P.107}. For an individual node, this distance can be overruled by setting \texttt{/gtr/distance}\footnote{P.94}.

\begin{verbatim}
\gtrset{
  further distance=5mm
}
\end{verbatim}
If /gtr/further distance 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.103*, `/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 TiKZ node `<name>` is automatically created for later usage.

```
...% family with id 'family_a'
child[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'
}
...% family with id 'family_a'
```

For example, let us consider the Smith-Doe graph used many times in this document, see Section 14.1 on page 295. 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}
```

![Genealogy tree diagram](image-url)
One could inspect the source code in Section 14.1 on page 295 to see the given identifiers. For a large dataset, this may become inconvenient. A good alternative is to use /gtr/show id"P.251 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)

The given \textit{(text)} is prefixed to every /gtr/id. This option is intended to be used as part of an option list for a \genealogytree or genealogypicture. If not used there, it will be set to empty by \genealogytree.

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

/gtr/id suffix=(text)

The given \textit{(text)} is suffixed to every /gtr/id. This option is intended to be used as part of an option list for a \genealogytree or genealogypicture. If not used there, it will be set to empty by \genealogytree.

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

\texttt{/gtr/options for node=\{id list\}\{options\}\{} 

The given \{options\} are set for all nodes with \texttt{/gtr/id=P.90} values from the given \{id list\}. If an \texttt{/gtr/id=P.90} value is not existing, the setting is silently ignored. The intended spot for using \texttt{/gtr/options for node} is before \texttt{\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 \texttt{/gtr/options for node}. 

\begin{tikzpicture}
\node[red, fill, font=\scriptsize] {Arthur \star 2008};
\node[red, fill, font=\scriptsize] {Berta \star 2010};
\node[red, fill, font=\scriptsize] {Charles \star 2014};
\node[blue, fill, font=\scriptsize] {John Smith \star 1980};
\node[blue, fill, font=\scriptsize] {Jane Doe \star 1982};
\node[blue, fill, font=\scriptsize] {Uncle Harry \star 1987};
\node[blue, fill, font=\scriptsize] {Grandpa Smith \star 1949};
\node[blue, fill, font=\scriptsize] {Grandma Smith \star 1952};
\node[blue, fill, font=\scriptsize] {Grandpa Doe \star 1955};
\node[blue, fill, font=\scriptsize] {Grandma Doe \star 1956};
\end{tikzpicture}
The given \textit{options} are set for all nodes within the current scope. This scope is primarily intended to be a \texttt{/gtr/level \textsuperscript{P.107}} or \texttt{/gtr/level n \textsuperscript{P.108}} definition. For other spots, where \texttt{/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}}}}]
\end{tikzpicture}

\begin{tikzpicture}
genealogytree[template=signpost,]
\end{tikzpicture}

A non-negative \texttt{length} replaces the default minimum distance to the preceding sibling. The default settings are given by \texttt{/gtr/child distance in parent graph \textsuperscript{P.84}} etc.

\begin{tikzpicture}
genealogytree[template=signpost, options for node={GpDo1955}{distance=2cm}]
\end{tikzpicture}
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 \texttt{/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. Depending on
the selected processor, the ⟨options⟩ are usually \texttt{tcolorbox} options which describe how a
node box is drawn. If a processor is not based on the \texttt{tcolorbox} package, the ⟨options⟩
can be different, e.g. \texttt{TikZ} options.

\begin{tikzpicture}
\genealogytree[template=formal graph, options for node={node_B}{box={colback=green!50}},]
\node node_B at (0,0) {B};
\child{g{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.

\begin{tikzpicture}
\genealogytree[template=formal graph, level -1/.style={node box={colback=red!50}},]
\node node_B at (0,0) {B};
\child{g{A} p{B} c{C} c{D} c{E}}
\end{tikzpicture}
This is an abbreviation for placing /gtr/box P.96 inside /gtr/family P.103.

\begin{tikzpicture}
genealogytree[template=formal graph]
\%
\begin{verbatim}
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{verbatim}
\end{tikzpicture}

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

\begin{tikzpicture}
genealogytree[template=formal graph]
\%
\begin{verbatim}
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{verbatim}
\end{tikzpicture}
This is a special \texttt{/gtr/box\_\textsuperscript{P.96}} style to rotate the content a node. Typically, all nodes of a \texttt{/gtr/level \textsuperscript{P.108}} may be rotated together.

Feasible \textit{(option)} values are:

- \texttt{off}: no rotation.
- \texttt{right}: rotation by 90 degrees.
- \texttt{upsidedown}: rotation by 180 degrees.
- \texttt{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}
The following three options are \texttt{tcbox} options which are declared by the \texttt{genealogytree} package. They can be redefined for customization.

\texttt{/tcb/male} 
A \texttt{tcbox} option defined as

\texttt{\tcboxset{male/.style={colframe=blue}}}

\texttt{/tcb/female} 
A \texttt{tcbox} option defined as

\texttt{\tcboxset{female/.style={colframe=red}}}

\texttt{/tcb/neuter} 
A \texttt{tcbox} option defined as

\texttt{\tcboxset{neuter/.style={}}}

The following three options are \texttt{genealogytree} options which are shortcuts for setting the three options above inside a \texttt{/gtr/box}. \footnote{P. 96.}

\texttt{/gtr/male} 
This is an abbreviation for placing \texttt{/tcb/male} inside \texttt{/gtr/box}. \footnote{P. 96.}

\texttt{/gtr/female} 
This is an abbreviation for placing \texttt{/tcb/female} inside \texttt{/gtr/box}. \footnote{P. 96.}

\texttt{/gtr/neuter} 
This is an abbreviation for placing \texttt{/tcb/neuter} inside \texttt{/gtr/box}. \footnote{P. 96.}

Also see \texttt{/gtr/database/sex}. \footnote{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}
Applys extra TiKZ \textit{options} for drawing a node. These \textit{options} are \textit{not} used for the node box content, even if /gtr/\texttt{processing} \textit{P.128} is set to \texttt{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 \textit{not} aware of such movements, but edge drawing will follow the new positioning.

For node movements which influence the auto-layout algorithm, see /gtr/\texttt{distance} \textit{P.94}, /gtr/\texttt{pivot} \textit{P.95}, and /gtr/\texttt{pivot shift} \textit{P.104}. For node content option settings like coloring, see /gtr/\texttt{box} \textit{P.96}. 
5.6 Family Options

\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}

\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}

\texttt{\gtrsetoptionsforfamily\{(id list)\}\{\{options\}\}}

Identical to using \texttt{/gtr/options for family}. 

The given \texttt{\{options\}} are set for all families with \texttt{/gtr/id} \textsuperscript{\texttt{P.90}} values from the given \texttt{\{id list\}}. If an \texttt{/gtr/id} \textsuperscript{\texttt{P.90}} value is not existing, the setting is silently ignored. The intended spot for using \texttt{/gtr/options for family} is before \texttt{\genealogytree} \textsuperscript{\texttt{P.55}} or inside its option list. Also see Section 5.1.1 on page 76 and Section 5.1.2 on page 77.
The given \texttt{options} are set for all nodes and edges within the current scope. This scope is intended to be a \texttt{/gtr/level} \textsuperscript{P.107} or \texttt{/gtr/level n} \textsuperscript{P.108} definition or an option of a family identifier like \texttt{parent} or \texttt{child}. Also see Section 5.1.1 on page 76 and Section 5.1.2 on page 77.
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 /\texttt{pivot shift}, there is a shift of the given \texttt{length} between these two points. Note that this works for \texttt{child}, \texttt{parent}, and \texttt{sandclock}, but not for \texttt{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

\texttt{\textasciitilde{gtr}/options for subtree=(\langle id list\rangle)\{\langle options\rangle\}} (style, no default)

The given \texttt{\langle options\rangle} are set for all subtrees with \texttt{\textasciitilde{gtr}/id=P.90} values from the given \texttt{\langle id list\rangle}. Subtrees are identified by the \texttt{\textasciitilde{gtr}/id=P.90} of the root family of the subtree. If an \texttt{\textasciitilde{gtr}/id=P.90} value is not existing, the setting is silently ignored. The intended spot for using \texttt{\textasciitilde{gtr}/options for subtree} is before \texttt{\genealogytree}\textasciitilde{.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}

\texttt{\textasciitilde{gtrsetoptionsforsubtree}=(\langle id list\rangle)\{\langle options\rangle\}}

Identical to using \texttt{\textasciitilde{gtr}/options for subtree}.
The given \textit{options} are set for all families and their nodes and edges within the current scope. This scope is intended to be a \texttt{gtr/level \*P.107} or \texttt{gtr/level n \*P.108} definition or an option of a family identifier like \texttt{parent} or \texttt{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 \textsuperscript{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 \textsuperscript{P.94} to set options for nodes.
- /gtr/family \textsuperscript{P.103} to set options for families.
- /gtr/subtree \textsuperscript{P.106} to set options for subtrees.
- Also see /gtr/ignore \textsuperscript{P.115} and /gtr/ignore level \textsuperscript{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 \textit{(number)} as parameter. This style can be redefined.
At each level with the level number \( n \) this style is applied after \texttt{/gtr/level} \textsuperscript{*}. 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`\(^\text{+P.107}\) 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`\(^\text{+P.78}\), 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`\(^\text{+P.110}\): place a node relative to a position (respecting the node dimensions).
- `/gtr/adjust node`\(^\text{+P.111}\): 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`\(^\text{+P.90}\).

```
\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}
```
Adjusts the current graph such that a \langle node \rangle of the graph is placed in the given \langle direction \rangle relative to the given \langle position \rangle with a given \langle distance \rangle in this direction and an optional \langle shift \rangle orthogonal to the direction. The \langle node \rangle is identified by a /gtr/id. Feasible values for the \langle direction \rangle 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}
Adjusts the current graph such that a \texttt{node} of the graph is placed in the given \texttt{direction} relative to the given \texttt{reference node} (a TikZ node) with a given \texttt{distance} in this direction and an optional \texttt{shift} orthogonal to the direction. The \texttt{node} is identified by a \texttt{/gtr/id} \footnote{P. 90}.

Feasible values for the \texttt{direction} are:

- \texttt{right} (right of \texttt{reference node}. \texttt{east})
- \texttt{left} (left of \texttt{reference node}. \texttt{west})
- \texttt{above} (above of \texttt{reference node}. \texttt{north})
- \texttt{below} (below of \texttt{reference node}. \texttt{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 \texttt{/gtr/tikz} \cite[p.101]{gtr}.  

\begin{verbatim}
\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}
\end{verbatim}

\begin{verbatim}
\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}
\end{verbatim}
/gtr/after tree=\langle \text{tikz code} \rangle 
(no default, initially empty)
Used to insert \langle \text{tikz code} \rangle 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,  
amfter tree={ \draw[very thick,blue,-Latex] (node_A) to[out=180,in=120] (node_C);  
}]
\begin{genealogypicure} [template=formal graph,  
amfter tree={ \draw[very thick,blue,-Latex] (node_A) to[out=180,in=120] (node_C);  
}]
\end{genealogypicture}

/gtr/tcbset=\langle \text{tcolorbox options} \rangle 
(no default, initially empty)
Used to insert \langle \text{tcolorbox options} \rangle 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}  
}]
\begin{genealogypicure} [template=formal graph,  
tcbset={  
male/.style={colframe=blue,colback=blue!5},  
female/.style={colframe=red,colback=red!5}  
}]
\end{genealogypicture}
/tikz/fit to family=(id) (style, no default)
This is an extension to the fit library of TikZ. 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 TikZ 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 TikZ. All nodes of the subtree identified by (id) are given to the fit option of a TikZ 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 235 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.

\texttt{/gtr/ignore=true|false} (default \texttt{true}, initially \texttt{false})

The \texttt{/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 \texttt{/gtr/ignore} inside a node ignores this node.
- Using \texttt{/gtr/ignore} inside a family means that the whole subtree becomes ignored.
- Using \texttt{/gtr/ignore} inside \texttt{/gtr/level n} 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 \texttt{/gtr/ignore level}.

To ignore a node or subtree by its \texttt{/gtr/id}, use \texttt{/gtr/ignore node} or \texttt{/gtr/ignore subtree}.

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

/gtr/ignore node={⟨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}

/gtr/ignore subtree={⟨id list⟩}
Identical to using /gtr/ignore subtree.
\texttt{/gtr/ignore level=\textit{number}} \hspace{1cm} \text{(style, no default)}

The level with the given \textit{number} is ignored. This also removes unconnected nodes and families. \textbf{Note that /gtr/ignore level should never be used, if /gtr/proband level\textsuperscript{\textit{\textgreater}P.109} was set!}

This style sets \texttt{/gtr/level n\textsuperscript{\textless}P.108} options to remove all unwanted nodes and families. Depending on the algebraic sign of \textit{number} the implementation differs. Zero has no effect.

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

\begin{itemize}
\item[\texttt{\gtrset{ignore level=4}}]
\textit{\% is equal to}\n\texttt{\gtrset{level 4/.style={node=ignore},level 5/.style={ignore}}}
\item[\texttt{\gtrset{ignore level=-4}}]
\textit{\% is equal to}\n\texttt{\gtrset{level -4/.style={ignore,node=ignore}}}
\end{itemize}
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 \textsuperscript{P.236} ignores inserting options, while \gtrprocessordebug \textsuperscript{P.238} 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 \textsuperscript{P.119} should never be used to insert data after the root element of an inserted node or family.

\texttt{/gtr/insert after node}={⟨id⟩}{⟨input data⟩} \textsuperscript{(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.
/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}
The `\gtr/insert at end family` command inserts \textit{(input data)} into the graph data of the family with the given \textit{(id)}, after the content of the family is processed. If no family with \textit{(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.
5.13 Phantom Nodes and Subtrees

A /gtr/subtree style which makes the whole current subtree invisible. This style can also be applied for single nodes. If a \langle length \rangle value is used, the /gtr/node size for all nodes of the subtree is replaced by \langle length \rangle (width for vertical time flow and height for horizontal time flow).

\begin{tikzpicture}
genealogytree [template=formal graph, content interpreter content={\gtrifnodeid {\gtrnodeid}n_{\gtrnodenumber}}],

\begin{itemize}
\item child{ g-p-
  \begin{itemize}
  \item child{ g-p-
    \begin{itemize}
    \item c[id=A]-% invisible phantom
    \item c[phantom=2cm]-
    \end{itemize}
  \end{itemize}
  \begin{itemize}
  \item phantom; borders made visible
  \item child[phantom,subtree box={show bounding box}]{
    \begin{itemize}
    \item g-p-c-c-
    \end{itemize}
  }
  \end{itemize}
  \begin{itemize}
  \item c[id=B]-
  \end{itemize}
  \end{itemize}
  \end{itemize}
\end{itemize}
\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*]- },

\begin{itemize}
\item parent{
  \begin{itemize}
  \item g-c[id=A]-c[id=B]-
  \item p-p-
  \end{itemize}
  \end{itemize}
\end{itemize}
\end{tikzpicture}
\begin{tikzpicture}
genealogytree[{template=\texttt{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}
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 \langle code\rangle 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}

\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}
Every node in a \texttt{genealogytree}\textsuperscript{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 \texttt{/gtr/node processor}\textsuperscript{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 \textit{as-is} or changed in some way before the node processor displays it. This is done by an element called \texttt{/gtr/content interpreter}\textsuperscript{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 \textit{node data processing} in the following.

Two classes of node processings can be distinguished:

- Non-interpreting node data processings take their content text \textit{as-is} and just format it with colors, fonts, frames, etc; see Section 6.2 on page 129.

- Interpreting node data processings use some \texttt{/gtr/content interpreter}\textsuperscript{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.

\input{node-processing}

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.

/gtr/processing=\langle processing \rangle (no default, initially fit)

Defines the base procedure for processing the content of a node. Feasible values for \langle processing \rangle are

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

Values given to /gtr/box will be interpreted according to the defined \langle processing \rangle. For \texttt{tcolorbox}, the values have to be \texttt{tcolorbox} settings; for \texttt{tikznode}, the values have to be \texttt{tikz} settings.
6.2 Predefined Non-Interpreting Processings

6.2.1 fit

The preset processing is based on \tcboxfit of the tcolorbox package [3]. Options given to \gtrbox have to be tcolorbox options which are used by \tcboxfit.

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

- Full observance of /gtr/level size, /gtr/node size, and /gtr/node size from. 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}]
  \hspace{2cm}
  \begin{verbatim}
  \hspace{1cm}child{ g[root]  
    c[turn=left]{Abc}  
    c[turn=upsidedown]{Bcd}  
    c[turn=right]{Cde}  
  }
  \end{verbatim}
\end{genealogypicture}
6.2.2 \texttt{tcolorbox}

\begin{verbatim}
/gtr/processing\^P.128=tcolorbox
\end{verbatim}

This processing is based on the \texttt{tcolorbox} environment of the \texttt{tcolorbox} package \cite{3}. Options given to \texttt{/gtr/box }\^P.96 have to be \texttt{tcolorbox} options.

The \texttt{/gtr/no content interpreter} \^P.147 is used. The main characteristics of the applied node data processor are:

- For \texttt{/gtr/timeflow} \^P.78 settings \texttt{up} and \texttt{down}, full observance of \texttt{/gtr/node size} \^P.83, but no observance of \texttt{/gtr/node size from} \^P.84. The \texttt{/gtr/level size} \^P.82 is observed, but content which is too large may overflow.

- For \texttt{/gtr/timeflow} \^P.78 settings \texttt{left} and \texttt{right}, full observance of \texttt{/gtr/level size} \^P.82. \texttt{/gtr/node size} \^P.83 and \texttt{/gtr/node size from} \^P.84 are both observed, but content which is too large may overflow.

- Using the option \texttt{natural height}, the height of a node box can be freely adapted to its content. This may be especially useful for \texttt{/gtr/timeflow} \^P.78 settings \texttt{left} and \texttt{right}, but with some limited use for \texttt{/gtr/timeflow} \^P.78 settings \texttt{up} and \texttt{down}.

- Extremely customizable using options.

- To observe node and level settings as far as possible, the dimensions can be set by \texttt{/tcb/gtrNodeDimensions} or \texttt{/tcb/gtrNodeDimensionsLandscape}. \texttt{/tcb/gtrNodeDimensions} is initially set.
6.2.3 \texttt{tcbox}

\texttt{/gtr/processing} \texttt{tcbox}

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

The \texttt{/gtr/no content interpreter} is used. The main characteristics of the applied
node data processor are:

- For \texttt{/gtr/timeflow} settings \texttt{up} and \texttt{down}, no observance of \texttt{/gtr/node size} and
  \texttt{/gtr/node size from}, but full observance of \texttt{/gtr/level size} if
  \texttt{/tcb/gtrNodeDimensions} is set.

- For \texttt{/gtr/timeflow} settings \texttt{left} and \texttt{right}, no observance of \texttt{/gtr/level size}.
  \texttt{/gtr/node size} and \texttt{/gtr/node size from} are both observed,
  if \texttt{/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
  \texttt{/tcb/gtrNodeDimensions} or \texttt{/tcb/gtrNodeDimensionsLandscape}.
  \texttt{/tcb/gtrNodeDimensions} is initially \texttt{not} set, but \texttt{/gtr/turn} 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}
6.2.4 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}\begin{tcbox*}\begin{align*}
\text{root} & & X \\
Y & & A & & D & & E & & Z \\
B & & C & & F & & G & & H
\end{align*}\end{tcbox*}\end{genealogypicture}

\begin{genealogypicture}\begin{tcbox*}
\text{root} & & \text{This is some longer text.} \\
Y & & A & & D & & E & & Z \\
B & & C & & F & & G & & H
\end{tcbox*}\end{genealogypicture}
This processing is based on \node of the \tikz package \cite{4}. Options given to /gtr/box\cite{96} have to be \tikz options which are used by \node.

The /gtr/no content interpreter\cite{147} is used. The main characteristics of the applied node data processor are:

- No observance of /gtr/level size\cite{82}, /gtr/node size\cite{83}, and /gtr/node size from\cite{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\cite{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}
6.3 Creating a Customized Non-Interpreting Processor

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

But using \texttt{/gtr/node processor}^\textsuperscript{P.128}, also a new node processor can be defined.

\begin{verbatim}
\gtrset{node processor=myprocessor}
\end{verbatim}

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

- \texttt{\gtrBoxContent} contains the (already interpreted or not interpreted) node content.
- \texttt{\gtrNodeMinWidth} contains the current target minimum node width as defined by the various tree settings.
- \texttt{\gtrNodeMaxWidth} contains the current target maximum node width as defined by the various tree settings.
- \texttt{\gtrNodeMinHeight} contains the current target minimum node height as defined by the various tree settings.
- \texttt{\gtrNodeMaxHeight} contains the current target maximum node height as defined by the various tree settings.
- \texttt{\gtrNodeBoxOptions} contains the option settings for the current node. These are the assembled \texttt{/gtr/box}^\textsuperscript{P.96} settings as comma separated key-value list for the current node.

For demonstration, a simple processor based on the \texttt{minipage} environment is constructed:

\begin{verbatim}
\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}
\end{verbatim}
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 \textsuperscript{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 \textsuperscript{P.\ 128}.

/gtr/content interpreter=(macro) (no default)

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

\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}

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.

\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}

\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}
Sets \texttt{\langle code \rangle} for interpreting the content of a node. This \texttt{\langle code \rangle} can use a parameter \texttt{\#1} (the original box content) and has to define a new parameterless macro \texttt{\gtrBoxContent} which should store the content which is given to the current \texttt{\gtr/node processor} 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}

Sets \texttt{\langle code \rangle} for interpreting the content of a node. This \texttt{\langle code \rangle} is the definition for \texttt{\gtrBoxContent}. The \texttt{\langle code \rangle} can use a parameter \texttt{\#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{A} p{B}
    child{ p{C} g{D} c{E} c{F} }
    c{G}
  }
}
\end{tikzpicture}

\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/no content interpreter  (no value, initially set)
Virtually removes any content interpreter. The node content is given directly to the current /gtr/processing for further compilation. Actually, this defines \gtrBoxContent to contain the original box content.

/gtr/deletion content interpreter  (no value, initially set)
Deletes any box content. This leads to empty boxes.
This is the content interpreter for database processing, see Chapter 7 on page 151.

\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}

The box content is not only used as-is but is also set as /gtr/id \textsuperscript{P. 90} of the node. This implies that no macro code is used inside the nodes.
Sets \( \langle \text{code} \rangle \) for interpreting the content of a node. This \( \langle \text{code} \rangle \) is the definition for \gtrBoxContent\textsuperscript{P.144}. Also, the \( \langle \text{id} \rangle \) for the node is set. The \( \langle \text{code} \rangle \) and \( \langle \text{id} \rangle \) can use a parameter \#1 (the original box content). Note that \( \langle \text{id} \rangle \) 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{
      \text{g-}\text{P-}
      \child{ \text{p-g-c-c-} }
      \text{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}
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 =database
```

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

The \texttt{/gtr/database content interpreter} 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 163 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 296).

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 values. They are essential as handle to access a singular node from an importing document without changing the database.
Charles Smith
★ ca. 1722
London
❖ April 13, 1722
London
× October 12, 1764
Copper smith, soldier. Invented the square wheel.

Jane Bowden
⊙ March 2, 1742
London
🕶 1724 to 1802

Abraham Bowden
★ January 4, 1740
London
† February 23, 1740
London

Elizabeth “Liz” Smith
★ February 2, 1744
London
🕶 1780 to 1805
New York
† April 12, 1812
Boston
Had a store in Boston.

Michael Smith
★ † March 1, 1758
7.3 Data Keys

\[\text{/gtr/database/name=⟨full name⟩} \quad \text{(no default, initially empty)}\]

This key holds the ⟨full name⟩ of a person presumably with markup. For customization, the markup should be done with \texttt{\textbackslash pref}~\texttt{P.173}, \texttt{\textbackslash surn}~\texttt{P.173}, \texttt{\textbackslash nick}~\texttt{P.173} instead of common \LaTeX{} font settings.

\begin{verbatim}
... name = {\texttt{\textbackslash pref}\{Elizabeth\} \texttt{\textbackslash nick}\{Liz\} \texttt{\textbackslash surn}\{Smith\}},
...
\end{verbatim}

- \texttt{\textbackslash pref}~\texttt{P.173} marks a preferred given name.
- \texttt{\textbackslash nick}~\texttt{P.173} marks a nickname.
- \texttt{\textbackslash surn}~\texttt{P.173} marks a surname.

The saved data is accessible by \texttt{\textbackslash gtrDBname}.

\[\text{/gtr/database/shortname=⟨short name⟩} \quad \text{(no default, initially empty)}\]

This key holds an optional ⟨short name⟩ of a person presumably with markup. For customization, the markup should be done with \texttt{\textbackslash pref}~\texttt{P.173}, \texttt{\textbackslash surn}~\texttt{P.173}, \texttt{\textbackslash nick}~\texttt{P.173} instead of common \LaTeX{} font settings.

\begin{verbatim}
... shortname = {\texttt{\textbackslash nick}\{Liz\} \texttt{\textbackslash surn}\{Smith\}},
...
\end{verbatim}

The saved data is accessible by \texttt{\textbackslash gtrDBshortname}.

\[\text{/gtr/database/sex=⟨sex⟩} \quad \text{(no default, initially neuter)}\]

This key holds the ⟨sex⟩ of a person. Feasible values are \texttt{male} and \texttt{female}. \texttt{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 \texttt{\textbackslash gtrDBsex}.

\begin{verbatim}
... sex = female,
...
\end{verbatim}

\texttt{/gtr/database/female} \quad \text{(style, no value)}

Shortcut for \texttt{/gtr/database/sex=female}.

\texttt{/gtr/database/male} \quad \text{(style, no value)}

Shortcut for \texttt{/gtr/database/sex=male}.

\texttt{/gtr/database/neuter} \quad \text{(style, no value)}

Shortcut for \texttt{/gtr/database/sex=neuter}.

\[\text{/gtr/database/comment=⟨text⟩} \quad \text{(no default, initially empty)}\]

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

\begin{verbatim}
... comment = {Had a store in Boston},
...
\end{verbatim}
/gtr/database/profession={text}  
This key holds some ⟨text⟩ about the profession a person. The saved data is accessible by \\
\gtrDBprofession.  

%...  
\texttt{profession} = \{Copper smith, soldier\},  
%...  

/gtr/database/image={file name}  
This key holds an image ⟨file name⟩ of a person’s portrait. The saved data is accessible by \\
\gtrDBimage.  

%...  
\texttt{image} = Marry_Smith_1720.jpg,  
%...  

/gtr/database/imageopt={⟨options⟩}  
This key holds some ⟨options⟩ to be applied for including an image with \\
/gtr/database/image. These ⟨options⟩ should be valid for \includegraphics. The saved data is accessible by \\
\gtrDBimageopt.  

%...  
\texttt{imageopt}={\texttt{viewport}=30pt 50pt 150pt 180pt,\texttt{clip}},  
%...  

/gtr/database/viewport={a b c d}  
(style, no default)  
This style is a shortcut for calling /gtr/database/imageopt with an \includegraphics viewport of ⟨a b c d⟩ plus clip option. The saved data is accessible by \\
\gtrDBimageopt.  

%...  
\texttt{viewport}=30pt 50pt 150pt 180pt,  
%...  

/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.  

%...  
\texttt{uuid} = 1021aa0c-2508-488c-9760-f9f84b4df1dd,  
%...
This key holds the Kekulé number of a person. The saved data is accessible by `\gtrDBkekule`.

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

This key holds a relationship `<text>` describing the person. The saved data is accessible by `\gtrDBrelationship`.

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

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`.

```
```
The following data keys hold events. Every event consists of:

- a date, see Section 7.4 on page 161,
- 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 \texttt{\gtrPrintEvent} on page 180, \texttt{\gtrPrintDate} on page 175, and \texttt{\gtrPrintPlace} on page 179. The existence of data can be checked by \texttt{\gtrifdatedefined} on page 175 and \texttt{\gtrifplacedefined} on page 179.

\begin{verbatim}
\texttt{/gtr/database/birth={\langle date\rangle}{\langle place\rangle}}
\end{verbatim}

This key holds a birth event with given \langle date\rangle and \langle place\rangle.

\begin{verbatim}
%...
birth = {1744-02-02}{London},
%...
\end{verbatim}

\begin{verbatim}
\texttt{/gtr/database/birth+={\langle date\rangle}{\langle place\rangle}{\langle modifier\rangle}}
\end{verbatim}

This key holds a birth event with given \langle date\rangle, \langle place\rangle, and a \langle modifier\rangle to describe the event further. Feasible values for the \langle modifier\rangle are:

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

\begin{verbatim}
%...
birth+ = {1740-01-04}{London}{out of wedlock},
%...
\end{verbatim}

\begin{verbatim}
\texttt{/gtr/database/birth-={\langle date\rangle}}
\end{verbatim}

This key holds a birth event with given \langle date\rangle.

\begin{verbatim}
%...
birth- = {1744-02-02},
%...
\end{verbatim}

\begin{verbatim}
\texttt{/gtr/database/baptism={\langle date\rangle}{\langle place\rangle}}
\end{verbatim}

This key holds a baptism event with given \langle date\rangle and \langle place\rangle.

\begin{verbatim}
\texttt{/gtr/database/baptism+={\langle date\rangle}{\langle place\rangle}{\langle modifier\rangle}}
\end{verbatim}

Identical to \texttt{/gtr/database/baptism} since there is no valid \langle modifier\rangle.

\begin{verbatim}
\texttt{/gtr/database/baptism-={\langle date\rangle}}
\end{verbatim}

This key holds a baptism event with given \langle date\rangle.
This key holds an engagement event with given ⟨\text{date}⟩ and ⟨\text{place}⟩.

Identical to /gtr/database/engagement since there is no valid ⟨\text{modifier}⟩.

This key holds a marriage event with given ⟨\text{date}⟩, ⟨\text{place}⟩, and a ⟨\text{modifier}⟩ to describe the event further. Feasible values for the ⟨\text{modifier}⟩ are

• empty (normal),
• other.

Identical to /gtr/database/divorce since there is no valid ⟨\text{modifier}⟩.

This key holds a floruit event with given ⟨\text{date}⟩, ⟨\text{place}⟩, and a ⟨\text{modifier}⟩ to describe the event further. Feasible values for the ⟨\text{modifier}⟩ are

• empty (normal),
• killed.

This key holds a death event with given ⟨\text{date}⟩, ⟨\text{place}⟩, and a ⟨\text{modifier}⟩ to describe the event further. Feasible values for the ⟨\text{modifier}⟩ are
/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 \(date\).
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 calendar 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 between Gregorian and Julian calendar should be emphasized.
- **JU**: Julian calendar; use this in situations, where the difference between 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 =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

The following table shows the formatted dates for various specifications.

<table>
<thead>
<tr>
<th>Specification</th>
<th>d/M/yyyy</th>
<th>Month d yyyy</th>
<th>dd/mo/yyy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1875–12–07</td>
<td>7.IX.1875</td>
<td>December 7, 1875</td>
<td>07/Dec/1875</td>
</tr>
<tr>
<td>(JU) 1642–12–25</td>
<td>25.XII.1642</td>
<td>December 25, 1642</td>
<td>25/Dec/1642</td>
</tr>
<tr>
<td>(GR) 1599–08</td>
<td>VIII.1599</td>
<td>August, 1599</td>
<td>15/Aug/1599</td>
</tr>
<tr>
<td>1475</td>
<td>1475</td>
<td>1475</td>
<td>1475</td>
</tr>
<tr>
<td>/1690–03</td>
<td>before III.1690</td>
<td>before March, 1690</td>
<td>before Mar/1690</td>
</tr>
<tr>
<td>1775–07–15/</td>
<td>after 15.VII.1775</td>
<td>after July 15, 1775</td>
<td>after 15/Jul/1775</td>
</tr>
<tr>
<td>1888–05/1889–06–07</td>
<td>V.1888 to 7.VI.1889</td>
<td>May, 1888 to June 7, 1889</td>
<td>May/1888 to 07/Jun/1889</td>
</tr>
</tbody>
</table>

161
\texttt{\textbackslash gtrParseDate\{\textit{name}\}\{\textit{date}\}}

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

\begin{Verbatim}
\texttt{\textbackslash gtrParseDate\{xy\}\{1875-12-07\}}
\textnormal{The parsed date is}
\texttt{\textbackslash gtrDBxy\{cal\}}, \texttt{\textbackslash gtrDBxy\{day\}}, \texttt{\textbackslash gtrDBxy\{month\}}, \texttt{\textbackslash gtrDBxy\{year\}}, \texttt{\textbackslash gtrDBxy\{endcal\}}, \texttt{\textbackslash gtrDBxy\{endday\}}, \texttt{\textbackslash gtrDBxy\{endmonth\}}, \texttt{\textbackslash gtrDBxy\{endyear\}}.
\textnormal{Formatted date: \texttt{\textbackslash gtrset\{date format=d/M/yyyy\}\textbackslash gtrPrintDate\{xy\}}}
\end{Verbatim}

The parsed date is \texttt{\textbackslash gtrDBxy\{cal\} (AD)}, \texttt{\textbackslash gtrDBxy\{day\} (07)}, \texttt{\textbackslash gtrDBxy\{month\} (12)}, \texttt{\textbackslash gtrDBxy\{year\} (1875)}.
\textnormal{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.

Select a predefined `format` for selecting and arranging data values. The standard `format` designs use `/gtrprintlist` to list events. New `format` designs can be added by `/gtrDeclareDatabaseFormat`. The following sections describe how to customize certain parts of the standard `format` designs, e.g. `/gtr/date format` 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 \tccline.
- **full marriage below**: identical to `full`, but engagement, marriage, divorce is put below and separated by a \tccline.
- **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 \tccline.
- **medium marriage below**: identical to `medium`, but marriage (or engagement or divorce) is put below and separated by a \tccline.
- **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 \tccline.
- **short marriage below**: identical to `short`, but marriage (or engagement or divorce) is put below and separated by a \tccline.
- **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`.
Charles Smith
★ ca. 1722
London
※ April 13, 1722
London
✝ October 12, 1764
Copper smith, soldier. Invented the square wheel.

Jane Bowden
♣ March 2, 1742
London
★ 1724 to 1802

Abraham Bowden
★ January 4, 1740
London
† February 23, 1740
London

Elizabeth “Liz” Smith
★ February 2, 1744
London
※ 1780 to 1805
New York
† April 12, 1812
Boston
Had a store in Boston.

Michael Smith
✝ March 1, 1758

Had a store in Boston.
Charles Smith  
★ ca. 1722  
London  
⇒ April 13, 1722  
London  
× October 12, 1764  
Copper smith, soldier. Invented the square wheel.

Jane Bowden  
★ 1724 to 1802  
☐ March 2, 1742  
London

Abraham Bowden  
(★) January 4, 1740  
London  
† February 23, 1740  
London

Elizabeth "Liz" Smith  
★ February 2, 1744  
London  
⇒ 1780 to 1805  
New York  
† April 12, 1812  
Boston  
Had a store in Boston.

Michael Smith  
★† March 1, 1758

Charles Smith  
★ ca. 1722  
London  
⇒ April 13, 1722  
London  
× October 12, 1764  
Copper smith, soldier. Invented the square wheel.

Jane Bowden  
★ 1724 to 1802  
☐ March 2, 1742  
London

Abraham Bowden  
(★) January 4, 1740  
London  
† February 23, 1740  
London

Elizabeth "Liz" Smith  
★ February 2, 1744  
London  
⇒ 1780 to 1805  
New York  
† April 12, 1812  
Boston  
Had a store in Boston.

Michael Smith  
★† March 1, 1758
Charles Smith
★ ca. 1722
London
☒ October 12, 1764
Copper smith, soldier. Invented the square wheel.

Jane Bowden
☒ March 2, 1742
London
★ 1724 to 1802

Abraham Bowden
★★ January 4, 1740
London
† February 23, 1740

Elizabeth “Liz” Smith
★ February 2, 1744
London
† April 12, 1812
Boston
Had a store in Boston.

Michael Smith
★★ March 1, 1758

Abraham Bowden
★★ January 4, 1740
London
† February 23, 1740

Elizabeth “Liz” Smith
★ February 2, 1744
London
† April 12, 1812
Boston
Had a store in Boston.

Michael Smith
★★ March 1, 1758

Charles Smith
★ ca. 1722
London
☒ October 12, 1764
Copper smith, soldier. Invented the square wheel.
Charles Smith
★ ca. 1722
London
X October 12, 1764
Copper smith, soldier. Invented the square wheel.

Jane Bowden
★ 1724 to 1802
London

Abraham Bowden
(★) January 4, 1740
London
† February 23, 1740
London

Elizabeth “Liz” Smith
★ February 2, 1744
London
† April 12, 1812
Boston
Had a store in Boston.

Michael Smith
★† March 1, 1758

Charles Smith
★ ca. 1722
London
X October 12, 1764
Copper smith, soldier. Invented the square wheel.

Jane Bowden
★ 1724 to 1802
London

Abraham Bowden
(★) January 4, 1740
London
† February 23, 1740
London

Elizabeth “Liz” Smith
★ February 2, 1744
London
† April 12, 1812
Boston
Had a store in Boston.

Michael Smith
★† March 1, 1758

167
Charles Smith
★ ca. 1722
London
★ October 12, 1764

Jane Bowden
♀ March 2, 1742
London
♀ 1724 to 1802

Abraham Bowden
(★) January 4, 1740
London
† February 23, 1740
London

Elizabeth “Liz” Smith
★ February 2, 1744
London
† April 12, 1812
Boston

Michael Smith
★★ March 1, 1758

Charles Smith
★ ca. 1722
London
★ October 12, 1764

Jane Bowden
♀ March 2, 1742
London
♀ 1724 to 1802

Abraham Bowden
(★) January 4, 1740
London
† February 23, 1740
London

Elizabeth “Liz” Smith
★ February 2, 1744
London
† April 12, 1812
Boston

Michael Smith
★★ March 1, 1758
\begin{genealogypicture}
\begin{verbatim}
processing=database,database format=name,  
node size=3cm,level size=1cm,  
box={fit basedim=9pt,boxsep=2pt,  
  halign=flush center,valign=center,\gtrDBsex} 
\end{verbatim}  
input{example.database.graph}
\end{genealogypicture}

\begin{genealogypicture}
\begin{verbatim}
processing=database,database format=symbol,  
node size=1cm,level size=1cm,  
box={fit basedim=16pt,boxsep=2pt,  
  halign=flush center,valign=center,\gtrDBsex} 
\end{verbatim}  
input{example.database.graph}
\end{genealogypicture}

\begin{genealogypicture}
\begin{verbatim}
processing=database,database format=empty,  
node size=1cm,level size=1cm,  
box={fit basedim=16pt,boxsep=2pt,  
  halign=flush center,valign=center,\gtrDBsex} 
\end{verbatim}  
input{example.database.graph}
\end{genealogypicture}
\gtrDeclareDatabaseFormat{(\format)\{\option\}\{\content\}}

Declares a new \format{} to be used as value for /gtr/database format\footnote{P. 163}. The \option{} is used after the data is read and before the box is set. The \content{} 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}{{\gtrPrintEvent{birth}}}{}% 
% \gtrifdatedefined{baptism}{{\gtrPrintEvent{baptism}}}{}% 
% \gtrifdatedefined{death}{{\gtrPrintEvent{death}}}{}% 
% \gtrifdatedefined{burial}{{\gtrPrintEvent{burial}}}{}% 
% \end{gtreventlist}% 
% \begin{gtrinfolist} 
% \gtrifprofessiondefined{{\gtrPrintProfession}}{}% 
% \gtrifcommentdefined{{\gtrPrintComment}}{}% 
% \gtrifdatedefined{marriage}{{\tcbline{\begin{gtreventlist}}}{}% 
% \gtrPrintEvent{marriage}{{\end{gtreventlist}}}{}% 
% \end{gtrinfolist}% 
% \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=\parskip, \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}, 
% } 
% \end{genealogypicture} 
% \gtrset{\box={title=\gtrDBkekule}}% 
% \end{genealogypicture} 
% \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=\parskip, \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}, 
% } 
% \end{genealogypicture} 
}
\begin{genealogypicture}

2
Charles Smith
★ 1722
London
Χ October 12, 1764
Copper smith, soldier.

3
Jane Bowden
★ 1724
London
† July 7, 1802
New York
Ω March 2, 1742
London

1
Abraham Bowden
(★) January 4, 1740
London
† February 23, 1740
London

Elizabeth “Liz” Smith
★ February 2, 1744
London
† April 12, 1812
Boston
Had a store in Boston.

Michael Smith
★† March 1, 1758

\end{genealogypicture}

\begin{verbatim}
\gtrset{database/.cd, 
  name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}, 
  birth = {1744-02-02}{London}, 
  death = {1812-04-12}{Boston}, 
  comment = {Had a store in Boston}, 
}
%...
\gtrPrintDatabase

Elizabeth “Liz” Smith
★ February 2, 1744 in London
† April 12, 1812 in Boston
Had a store in Boston.
\end{verbatim}
7.6 Formatting Names

\texttt{\textbackslash gtrPrintName}

Used to insert the formatted name. The output format of the name is controlled by \texttt{\textbackslash gtr/name} and other following options.

\begin{verbatim}
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}}
\%
\gtrPrintName

Elizabeth “Liz” Smith
\end{verbatim}

\texttt{\textbackslash pref\{(given name)\}}

Marks a preferred \texttt{(given name)}. May be redefined directly or using \texttt{/gtr/pref code}.

\begin{verbatim}
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}}
\gtrset{pref code={\textcolor{blue}{\bfseries #1}}}
\%
\gtrPrintName

Elizabeth “Liz” Smith
\end{verbatim}

\texttt{\textbackslash surn\{(surname)\}}

Marks a \texttt{(surname)}. May be redefined directly or using \texttt{/gtr/surn code}.

\begin{verbatim}
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}}
\gtrset{surn code={\textcolor{blue}{\bfseries #1}}}
\%
\gtrPrintName

Elizabeth “Liz” Smith
\end{verbatim}

\texttt{\textbackslash nick\{(nickname)\}}

Marks a \texttt{(nickname)}. May be redefined directly or using \texttt{/gtr/nick code}.

\begin{verbatim}
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}}
\gtrset{nick code={\textcolor{blue}{\bfseries #1}}}
\%
\gtrPrintName

Elizabeth Liz Smith
\end{verbatim}

\texttt{\textbackslash gtr/name=full|short}

Controls, if \texttt{\gtrPrintName} should preferably use the \texttt{full} version (\texttt{/gtr/database/name} \textsuperscript{P.155}) or the \texttt{short} version (\texttt{/gtr/database/shortname} \textsuperscript{P.155}) of a name. If the preferred version is not available, the other version is used.

173
\texttt{/gtr/name font={⟨code⟩}} \hspace{1em} (no default)
Sets the font (and/or color) for \texttt{\gtrPrintName} \textsuperscript{P.173}.

\begin{verbatim}
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}}}
\gtrset{name font={\fontfamily{ptm}\selectfont\color{green!50!black}}}
%...
\gtrPrintName
\end{verbatim}

\textit{Elizabeth “Liz” Smith}

\texttt{/gtr/empty name text={⟨text⟩}} \hspace{1em} (no default, initially ??)
Sets the text to be print by \texttt{\gtrPrintName} \textsuperscript{P.173}, if \texttt{/gtr/database/name} \textsuperscript{P.155} and \texttt{/gtr/database/shortname} \textsuperscript{P.155} were not set.

\begin{verbatim}
\gtrPrintName
\gtrset{empty name text={N.N.}}
\gtrPrintName
\end{verbatim}

??
N.N.

\texttt{/gtr/name code={⟨code⟩}} \hspace{1em} (no default)
Defines \texttt{⟨code⟩} to be executed by \texttt{\gtrPrintName} \textsuperscript{P.173}. Use this, if \texttt{/gtr/name} \textsuperscript{P.173} and \texttt{/gtr/name font} are not flexible enough.

\begin{verbatim}
\gtrset{database/.cd,name={\pref{Elizabeth} \nick{Liz} \surn{Smith}},female}
\gtrset{name code={\gtrPrintSex-\gtrDBname}}
%...
\gtrPrintName
\end{verbatim}

♀ \textit{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. 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)

This option controls how day, month, and year of a date are formatted when using \gtrPrintDate. This setting is not /gtr/language 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
- 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

175
The birth was February 9, 1354.

\gtrset{database/.cd, birth={1354-02-09}{Rome}}
\gtrset{date format=month d yyyy}
%... 
The birth was $\text{\textbackslash gtrPrintDate}{\text{birth}}$. 

The birth was February 9, 1354.
/gtr/date code={⟨code⟩} (no default)
Defines ⟨code⟩ to be executed by \gtrPrintDate\textsuperscript{P.175}. Use this, if /gtr/date format \textsuperscript{P.175} is not flexible enough.

```
\gtrset{database/.cd,birth={1354-02-09}{Rome}}
\gtrset{date code={%\ifcsdef{#1month}{}{%\ifcsdef{#1day}{\csuse{#1day}}{}{%\csuse{#1year}\}}%\}}
%
\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 \textsuperscript{P.233} 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 {})  
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 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 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 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.

/gtr/date range separator={⟨text⟩}  (style, default --, initially unset)  
Sets the same separator text for /gtr/date range full, /gtr/date range before, /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 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\{\langle name \rangle\}

Used to insert a formatted event referred by \langle name \rangle. This \langle name \rangle 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. 181, \gtrPrintEventPrefix, \gtrPrintDate P. 175, and \gtrPrintPlace P. 179.

\gtrset\{database/.cd,birth={1354-02-09}{Rome}\}
\gtrPrintEvent\{birth\}

\gtrifeventdefined\{\langle name \rangle\}\{\langle true \rangle\}\{\langle false \rangle\}

Expands to \langle true \rangle, if an event with the given \langle name \rangle is defined, and to \langle false \rangle otherwise. This is an alias for \gtrifdatedefined P. 175.

\gtrset\{database/.cd,birth={1354-02-09}{Rome}\}
\gtrifeventdefined\{birth\}\{\gtrPrintEvent\{birth\}\}\{\}

\gtrPrintEventPrefix\{\langle name \rangle\}

Used to insert an event prefix like a symbol. The prefix depends upon the \langle name \rangle 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: \gtrsymBorn (no default, initially \gtrsymBorn P. 227)

Prefix \langle text \rangle for a normal birth.

Birth: \gtrPrintEventPrefix\{birth\}

Birth: (b)

\gtr/event prefix/birth=(text) (no default, initially \gtrsymBaptized P. 227)

Prefix \langle text \rangle for a baptism.

\gtr/event prefix/birth/out of wedlock=(text) (no default, initially \gtrsymBaptized P. 227)

Prefix \langle text \rangle for a birth out of wedlock.

\gtr/event prefix/birth/stillborn=(text) (no default, initially \gtrsymBaptized P. 227)

Prefix \langle text \rangle for a stillborn child.

\gtr/event prefix/birth/died=(text) (no default, initially \gtrsymBaptized P. 227)

Prefix \langle text \rangle for a birth if a child who died on birthday.

\gtr/event prefix/baptism=(text) (no default, initially \gtrsymBaptized P. 227)

Prefix \langle text \rangle for a baptism.

\gtr/event prefix/engagement=(text) (no default, initially \gtrsymEngaged P. 227)

Prefix \langle text \rangle for an engagement.

\gtr/event prefix/marriage=(text) (no default, initially \gtrsymMarried P. 227)

Prefix \langle text \rangle for a normal marriage.
Prefix \textit{(text)} for another partnership.

Prefix \textit{(text)} for a divorce.

Prefix \textit{(text)} for a floruit event.

Prefix \textit{(text)} for a normal death.

Prefix \textit{(text)} for a death in war.

Prefix \textit{(text)} for a normal burial.

Prefix \textit{(text)} for a cremation.

This option controls how events are formatted when using \texttt{\textbackslash gtrPrintEvent\text{"P.180"}}. Feasible \textit{(format)} values are:

- \textbf{prefix date place} \quad \textit{February 9, 1354 in Rome}
- \textbf{prefix date} \quad \textit{February 9, 1354}
- \textbf{date} \quad \textit{February 9, 1354}

The four text pieces are placed inside \texttt{\textbackslash gtrPrintEvent\text{"P.180"}} as follows: \texttt{\textbackslash gtrPrintEvent\{#1\} \{#1\} \{#1\} \{#1\}}

This setting is \textit{not} \texttt{\textbackslash gtr/language\text{"P.233"}} dependent.

\texttt{\textbackslash gtrset\{database/.cd,birth={1354-02-09}{Rome}\}}
\texttt{\%...}
\texttt{\textbackslash gtrPrintEvent\{birth\}}

\[\star: \text{February 9, 1354 in Rome}\]

Defines \textit{(code)} to be executed by \texttt{\textbackslash gtrPrintEvent\text{"P.180"}}. Use this, if \texttt{\textbackslash gtr/event format} and \texttt{\textbackslash gtr/event text} are not flexible enough.

\texttt{\textbackslash gtrset\{database/.cd,birth={1354-02-09}{Rome}\}}
\texttt{\%}
\texttt{\textbackslash gtrPrintEventPrefix\{#1\} \{#1\} \{#1\} \{#1\}}
\texttt{\textbackslash gtrifplacedefined\{#1\} \{#1\} \{#1\} \{#1\} \{#1\} \{#1\} \{#1\}}
\texttt{\textbackslash gtrPrintDate\{#1\}}
\texttt{\%...}
\texttt{\textbackslash gtrPrintEvent\{birth\}}

\textit{\star} (in Rome) February 9, 1354
7.10 Formatting Lists of Events

\begin{gtrprintlist}\{⟨first⟩\}⟨⟨middle⟩⟩⟨⟨last⟩⟩{⟨empty⟩}
\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\}\{\unskip,\}⟨\unskip.\}⟨\unskip\}
\gtrlistseparator One
\gtrlistseparator Two
\gtrlistseparator Three
\gtrlistseparator Four
\end{gtrprintlist}

• One
• Two
• Three
• Four

\begin{gtreventlist}
\end{gtreventlist}

This is a \gtrprintlist environment with parameters specified by \gtr/list separators\textsuperscript{P. 183}. This environment is used internally by most \gtr/database format\textsuperscript{P. 163} settings to print event lists.

\begin{gtreventlist}
\end{gtreventlist}

\begin{gtrset}\{list separators=\}
\{\unskip\}\{\unskip,\}\{\unskip.\}⟨\unskip\}
\end{gtrset}

\begin{gtreventlist}\{\unskip\}\{\unskip,\}⟨\unskip.\}⟨\unskip\}
\gtrlistseparator One
\gtrlistseparator Two
\gtrlistseparator Three
\gtrlistseparator Four
\end{gtreventlist}

One, Two, Three, Four.
Defines \texttt{gtreventlist} as a \texttt{gtrprintlist} with the given parameters. This is used to list events.

\begin{genealogypicture}
  \begin{itemize}
  \item \texttt{list separators=} \langle \texttt{first}\rangle\{\langle \texttt{middle}\rangle\{\langle \texttt{last}\rangle\{\langle \texttt{empty}\rangle\}

\begin{itemize}
  \item \texttt{no default, initially \{\langle \par\rangle\{\langle \par\rangle\{\langle \par\rangle\{\langle \par\rangle\}}

\end{itemize}
\end{itemize}
\end{genealogypicture}

\begin{genealogypicture}
  \begin{itemize}
  \item \texttt{list separators hang=} \langle \texttt{length}\rangle

\begin{itemize}
  \item \texttt{style, default \texttt{tcbfitdim}}

\end{itemize}
\end{itemize}
\end{genealogypicture}
7.11 Formatting Comments

\texttt{\textbackslash gtrPrintComment}

Used to insert the formatted comment. May be redefined directly or using \texttt{/gtr/comment code}.

\begin{verbatim}
\gtrset{database/.cd,comment={Had a store in Boston}}
\%
\gtrPrintComment

Had a store in Boston
\end{verbatim}

\texttt{\textbackslash gtrifcommentdefined{(true)\{false\}}}

Expands to \texttt{(true)}, if a comment is defined, and to \texttt{(false)} otherwise.

\begin{verbatim}
\gtrset{database/.cd,comment={Had a store in Boston}}
\%
\gtrifcommentdefined{\gtrPrintComment}\{

Had a store in Boston
\end{verbatim}

\texttt{\textbackslash gtr/comment code=\{\texttt{code}\}}

(no default, initially \texttt{\{\emph{\textbackslash itshape\textbackslash gtrDBcomment}\}})

Redefines \texttt{\gtrPrintComment} using \texttt{(code)}.

\begin{verbatim}
\gtrset{database/.cd,comment={Had a store in Boston}}
\%
\gtrset{comment code=\{\textbackslash gtrDBcomment\}}
\%
\gtrPrintComment

(Had a store in Boston)
\end{verbatim}
7.12 Formatting Professions

\texttt{\textbackslash gtrPrintProfession}

Used to insert the formatted profession. May be redefined directly or using \texttt{/gtr/profession code}.

\texttt{\gtrset\{database/.cd,profession={Copper smith, soldier}\}}
\texttt{\%...}
\texttt{\gtrPrintProfession}

\textit{Copper smith, soldier}

\texttt{\textbackslash gtrifprofessiondefined\{(true)\}{(false)}

Expands to \textit{(true)}, if a profession is defined, and to \textit{(false)} otherwise.

\texttt{\gtrset\{database/.cd,profession={Copper smith, soldier}\}}
\texttt{\%...}
\texttt{\gtrifprofessiondefined\{\gtrPrintProfession\}\{}\texttt{\}}

\textit{Copper smith, soldier}

\texttt{\textbackslash gtr/profession code=\{(code)\} (no default, initially \texttt{\{itshape\textbackslash gtrDBprofession\}})

Redefines \texttt{\gtrPrintProfession} using \textit{(code)}.

\texttt{\gtrset\{database/.cd,profession={Copper smith, soldier}\}}
\texttt{\gtrset\{profession code=\{\gtrDBprofession\}\}}
\texttt{\%...}
\texttt{\gtrPrintProfession}

\textit{(Copper smith, soldier)}
7.13 Formatting Lists of Information

\begin{gtrinfolist}
\begin{environment content}
This is a gtrinfolist \textsuperscript{P.182} environment with parameters specified by /gtr/info separators. This environment is used internally by most /gtr/database format \textsuperscript{P.163} settings to print information lists consisting of /gtr/database/profession \textsuperscript{P.156} and /gtr/database/comment \textsuperscript{P.155} entries.
\end{environment content}
\end{gtrinfolist}

/gtr/info separators={\langle first\rangle}{\langle middle\rangle}{\langle last\rangle}{\langle empty\rangle} \textsuperscript{(no default, initially \par \{ . \} \{ . \} \{ . \})}

Defines gtrinfolist as a gtrprintlist \textsuperscript{P.182} with the given parameters. This is used to list informations.

\begin{genealogypicture}
\begin{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 }
\end{processing}
\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

\iffemale{
⟨true⟩}{⟨false⟩}

Expands to ⟨true⟩, if \gtrDBsex holds female, and to ⟨false⟩ otherwise.

\ifmale{
⟨true⟩}{⟨false⟩}

Expands to ⟨true⟩, if \gtrDBsex holds male, and to ⟨false⟩ otherwise.

Note that the content of the data key /gtr/database/sex \textsuperscript{155} is accessible by \gtrDBsex. Since /tcb/female \textsuperscript{99}, /tcb/male \textsuperscript{99}, /tcb/neuter \textsuperscript{99}, and /gtr/female \textsuperscript{99}, /gtr/male \textsuperscript{99}, /gtr/neuter \textsuperscript{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 163.
7.15 Formatting Images

\texttt{\textbackslash{}gtrifimagedefined\{(true)\}\{(false)\}}

Expands to \texttt{(true)}, if an image is defined, and to \texttt{(false)} otherwise.

\begin{verbatim}
\gtrset{database/.cd,image=Carl_Friedrich_Gauss.jpg}
\%...
\gtrifimagedefined{\includegraphics[width=3cm]{\gtrDBimage}}{no image}
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=0.3\textwidth]{Carl_Friedrich_Gauss.jpg}
\caption{Carl Friedrich Gauss}
\end{figure}

\texttt{\textbackslash{}gtrincludeDBimage\{\textit{options}\}}

Includes a database image with the given \textit{(options)}. These options are added to any options given by /\texttt{gtr/database/imageopt} \textsuperscript{P.156}.

\begin{verbatim}
\gtrset{database/.cd,image=Carl_Friedrich_Gauss.jpg}
\%...
\gtrifimagedefined{\gtrincludeDBimage\{width=3cm\}}{no image}
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=0.3\textwidth]{Carl_Friedrich_Gauss.jpg}
\caption{Carl Friedrich Gauss}
\end{figure}

\texttt{/tcb/if image defined={\{(true)\}\{(false)\}}}

Sets \texttt{(true)} \texttt{tcolorbox options}, if an image is defined, and sets \texttt{(false)} \texttt{tcolorbox options} otherwise. This key is intended to be used inside /\texttt{gtr/box} \textsuperscript{P.96} constructs.

\begin{verbatim}
\gtrset{
  options for node=mynode{
    box={if image defined={watermark graphics=\gtrDBimage}()}
  }
}
\%...
\end{verbatim}
/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{\getDBimage}

Picture file: picturedir/mytest.jpg

\verb^N^ 2018-04-16 /tikz/fill zoom DBimage (no value, initially unset)
Fills the current path with the current database image. The image is zoomed such that it fits inside the path area. Typically, some parts of the path area will stay unfilled.

\gtrset{database/.cd, image=Carl_Friedrich_Gauss.jpg, viewport=30pt 50pt 150pt 180pt}
\%...
\texttt{\getfimagedefined{\tikz\path[draw=red,fill zoom DBimage] (0,0) rectangle (2,3); \tikz\path[draw=red,fill zoom DBimage] (0,0) rectangle (3,3); \tikz\path[draw=red,fill zoom DBimage] (0,0) rectangle (3,2); }}
\{no image\}

\verb^N^ 2018-04-16 /tikz/fill overzoom DBimage (no value, initially unset)
Fills the current path with the current database image. The image is zoomed such that the path area fills the image.

\gtrset{database/.cd, image=Carl_Friedrich_Gauss.jpg, viewport=30pt 50pt 150pt 180pt}
\%...
\texttt{\getfimagedefined{\tikz\path[draw=red,fill overzoom DBimage] (0,0) rectangle (2,3); \tikz\path[draw=red,fill overzoom DBimage] (0,0) rectangle (3,3); \tikz\path[draw=red,fill overzoom DBimage] (0,0) rectangle (3,2); }}
\{no image\}
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 followed by a /gtr/edge/foreground. After all edges are drawn, the nodes are drawn (possibly over the edges).
8.1 Edge Settings

/\texttt{gtr/edges=\{\langle edge options\rangle\}}

(style, no default, initially \texttt{perpendicular})

Defines the \langle edge options\rangle for drawing the edges between the nodes of a family. Normally, an edge is drawn with a \texttt{gtr/edge/background}\footnote{P.202} graph and a \texttt{gtr/edge/foreground}\footnote{P.201} graph to allow visual separation of superposed edges. This setting may be given globally, as option of \texttt{genealogytree}\footnote{P.55} or locally wrapped by \texttt{gtr/family}\footnote{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\texttt{\gtrsymBorn\,2008}}
 c[id=Bert2010,female]{Berta\texttt{\gtrsymBorn\,2010}}
 c[id=Char2014,male]{Charles\texttt{\gtrsymBorn\,2014}}
 parent[id=Smith,family={edges=\{foreground=\texttt{blue!50,line width=2mm}\}}]{
 g[id=John1980,male]{John Smith\texttt{\gtrsymBorn\,1980}}
 p[id=GpSm1949,male]{Grandpa Smith\texttt{\gtrsymBorn\,1949}}
 p[id=GmSm1952,female]{Grandma Smith\texttt{\gtrsymBorn\,1952}}
 }!
 p[id=Jane1982,female]{Jane Doe\texttt{\gtrsymBorn\,1982}}
}
\end{tikzpicture}
This is a shortcut for embedding /gtr/edges \[P. 192\] into /gtr/family \[P. 103\].
This is a shortcut for embedding \texttt{/gtr/edges} \textsuperscript{P.192} into \texttt{/gtr/options for family} \textsuperscript{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}

This is a shortcut for embedding \texttt{/gtr/edges} \textsuperscript{P.192} into \texttt{/gtr/options for subtree} \textsuperscript{P.106}.

This is a shortcut for embedding \texttt{/gtr/edges} \textsuperscript{P.192} into \texttt{/gtr/options for subtree} \textsuperscript{P.105}.
\gtedgeset\{options\}
Sets \{options\} for the /\texttt{gtr/edge} key subtree. Mainly, this macro is intended to easily set up styles for edges.

\begin{tikzpicture}
\gtedgeset\{myedges/.style={rounded=6pt,
foreground={blue!50!black},background={blue!20!white}}
\%...
genealogytree[template=signpost,edges=myedges]
{input{example.option.graph}}
\end{tikzpicture}

\texttt{/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.

\begin{tikzpicture}
\gtedgeset{myedges/.style={rounded=6pt,
foreground={blue!50!black},background={blue!20!white}}}
%. ... Doe
\usetikzlibrary{backgrounds}
\tikzset{genealogytree edges scope/.style={on background layer}}
\end{tikzpicture}

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

\texttt{/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}

\texttt{/gtr/edge/rounded=⟨length⟩} (default 6pt)

The edges are drawn in a perpendicular but rounded style. The \texttt{⟨length⟩} describes the size of the rounding.

\begin{tikzpicture}
genealogytree [template=signpost, edges={rounded=6pt}]
{input{example.option.graph}}
\end{tikzpicture}
The edges are drawn in a swinging style. The \textit{length} describes the control parameter of the underlying curved path.

\begin{tikzpicture}
genealogytree[template=signpost, edges={swing=12pt}]
\input{example.option.graph}
\end{tikzpicture}
The edges are drawn meshed between parents and children. The \textit{options} are \LaTeX{} \texttt{tikz} \texttt{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}
The edges are drawn in a custom style. This option takes four mandatory parameters \(\text{\textlangle down\textrangle}, \text{\textlangle up\textrangle}, \text{\textlangle left\textrangle}, \text{\textlangle right\textrangle}\), each of which is a macro. The \(\text{\textlangle down\textrangle}\) macro is used to draw edges for /\textgtr\text/timelflow.\textsuperscript{P.78} = \text{\textlangle down\textrangle}, etc.

Every macro has to take four mandatory parameters:

1. An etoolbox \(\text{\textlangle listmacro\textrangle}\) which contains the list of anchor positions for the parents.
2. An etoolbox \(\text{\textlangle listmacro\textrangle}\) which contains the list of anchor positions for the children.
3. A TikZ node name which denotes the family core (center).
4. A TikZ style which should be applied to draw the edges.
This is a special \gtr/edge/custom 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

\texttt{/gtr/edge/foreground=} \{\textit{tikz options}\} (style, no default)

Defines the foreground \{\textit{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}

\texttt{/gtr/edge/no foreground} (style, no value)

Removes the \texttt{/gtr/edge/foreground} edges.

\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/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}

/home/gtr/edge/xshift=(length) (no default, in initially 0pt)
Shifts the edge core position horizontally by \langle length\rangle.

\begin{tikzpicture}
\genealogytree[template=signpost, edges={swing,xshift=5mm}]
{input{example.option.graph}}
\end{tikzpicture}
\begin{tikzpicture}
genealogytree[template=signpost,edges={swing,yshift=-3mm}]
{input{example.option.graph}}
\end{tikzpicture}

\begin{itemize}
\item Grandpa Smith ★ 1949
\item Grandma Smith ★ 1952
\item Grandpa Doe ★ 1955
\item Grandma Doe ★ 1956
\item John Smith ★ 1980
\item Jane Doe ★ 1982
\item Uncle Harry ★ 1987
\item Arthur ★ 2008
\item Berta ★ 2010
\item Charles ★ 2014
\end{itemize}
8.4 Edge Labels

\texttt{/gtr/label=\{\langle text \rangle\}} \quad \text{(style, no default)}

Adds a label \langle text \rangle to the current family. This is realized by a \texttt{TikZ} node with \texttt{/gtr/label options}. The current family is determined by a surrounding \texttt{/gtr/family} \footnote{P.103} or \texttt{/gtr/options for family} \footnote{P.102}.

\begin{verbatim}
\begin{genealogypicture}[template=signpost,
  label options={fill=white,node font=\footnotesize},
  options for family={\texttt{SmithDoe}}{\texttt{\langle gtr\symMarried\rangle-2006}} ]
\end{genealogypicture}
\end{verbatim}

\begin{genealogypicture}
  \begin{scope}
    \node[fill=red!50] (g1) at (0,0) {Grandpa Smith \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1949}}};
    \node[fill=red!50] (g2) at (2,0) {Grandma Smith \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1952}}};
    \node[fill=red!50] (g3) at (4,0) {Grandpa Doe \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1955}}};
    \node[fill=red!50] (g4) at (6,0) {Grandma Doe \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1956}}};
    \node[fill=blue!50] (j1) at (0,-2) {John Smith \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1980}}};
    \node[fill=blue!50] (j2) at (2,-2) {Jane Doe \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1982}}};
    \node[fill=blue!50] (j3) at (4,-2) {Uncle Harry \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1987}}};
    \node[fill=blue!50] (a1) at (0,-4) {Arthur \textcolor{gray}{\footnotesize \textbullet\textsuperscript{2008}}};
    \node[fill=blue!50] (b1) at (2,-4) {Berta \textcolor{gray}{\footnotesize \textbullet\textsuperscript{2010}}};
    \node[fill=blue!50] (c1) at (4,-4) {Charles \textcolor{gray}{\footnotesize \textbullet\textsuperscript{2014}}};
  \end{scope}
\end{genealogypicture}

\begin{verbatim}
\begin{genealogypicture}[template=signpost,
  label options={fill=green!20,node font=\footnotesize},
  label={\texttt{\langle gtr\symMarried\rangle}} ]
\end{genealogypicture}
\end{verbatim}

\begin{genealogypicture}
  \begin{scope}
    \node[fill=red!50] (g1) at (0,0) {Grandpa Smith \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1949}}};
    \node[fill=red!50] (g2) at (2,0) {Grandma Smith \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1952}}};
    \node[fill=red!50] (g3) at (4,0) {Grandpa Doe \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1955}}};
    \node[fill=red!50] (g4) at (6,0) {Grandma Doe \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1956}}};
    \node[fill=blue!50] (j1) at (0,-2) {John Smith \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1980}}};
    \node[fill=blue!50] (j2) at (2,-2) {Jane Doe \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1982}}};
    \node[fill=blue!50] (j3) at (4,-2) {Uncle Harry \textcolor{gray}{\footnotesize \textbullet\textsuperscript{1987}}};
    \node[fill=blue!50] (a1) at (0,-4) {Arthur \textcolor{gray}{\footnotesize \textbullet\textsuperscript{2008}}};
    \node[fill=blue!50] (b1) at (2,-4) {Berta \textcolor{gray}{\footnotesize \textbullet\textsuperscript{2010}}};
    \node[fill=blue!50] (c1) at (4,-4) {Charles \textcolor{gray}{\footnotesize \textbullet\textsuperscript{2014}}};
  \end{scope}
\end{genealogypicture}

\texttt{/gtr/label options=\{\langle options \rangle\}} \quad \text{(style, no default)}

Sets \texttt{TikZ} node \langle options \rangle to be used for \texttt{/gtr/label}. See \texttt{/gtr/label} for an example.
\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}
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 "marriage". Note that setting /gtr/database format 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.
```
Charles Smith
★ ca. 1722
London
Χ October 12, 1764
Copper smith, soldier.

March 2, 1742 (London)

Jane Bowden
★ ca. 1724
† July 7, 1802
New York

Abraham Bowden
(★) January 4, 1740
London
† February 23, 1740
London

Elizabeth “Liz” Smith
★ February 2, 1744
London
† April 12, 1812
Boston
Had a store in Boston.

Michael Smith
★† March 1, 1758
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 \textit{(child)} to a \textit{(family)} of the current graph. The auto-layout algorithm is not aware of this addition.
/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 \texttt{gtr/add child} \textsuperscript{P.209} and \texttt{gtr/add parent} \textsuperscript{P.210} is to draw the interconnecting node twice (the first one could be drawn as \texttt{gtr/phantom} \textsuperscript{P.124}). The second instance is drawn over the first instance using \texttt{gtr/set position} \textsuperscript{P.109}. Both instances need to have different \texttt{gtr/id} \textsuperscript{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 \texttt{genealogytree} \textsuperscript{P.55} separately. The second tree gets an \texttt{gtr/id prefix} \textsuperscript{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\[\texttr{\textgtrsymBorn},2012\]}
  p[id=Harr1987,male,box={colback=red!30!white}]{Uncle Harry\[\texttr{\textgtrsymBorn},1987\]}
  parent[id=Jones]{
    g[id=Mary1988,female,distance=1.4cm]{Aunt Mary\[\texttr{\textgtrsymBorn},1988\]}
    p[id=JimJ1944,male]{Jim Jones\[\texttr{\textgtrsymBorn},1944\]}
    p[id=Jenn1949,female]{Jenny Jones\[\texttr{\textgtrsymBorn},1949\]}
  }
}
\end{tikzpicture}
\texttt{/gtr/remove \textit{child}=(\textit{child}) from (\textit{family}) (style, no default)}

Removes a node as \textit{child} from a \textit{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}]
\end{tikzpicture}

\texttt{/gtr/remove \textit{parent}=(\textit{parent}) from (\textit{family}) (style, no default)}

Removes a node as \textit{parent} from a \textit{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 ]
\end{tikzpicture}
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 on the given \textit{value}, the node is disconnected as parent or child or both.

Feasible values are
- \textbf{child}: disconnect the node as child of a family.
- \textbf{parent}: disconnect the node as parent of a family. Note that a \texttt{g} node is only removed from its primary family, but not from connected \texttt{union} families.
- \textbf{both}: disconnect the node as child and as parent.

\texttt{/gtr/remove child} \footnote{P. 212} and \texttt{/gtr/remove parent} \footnote{P. 212} allow more precise control, but \texttt{/gtr/disconnect} needs no \texttt{/gtr/id} \footnote{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

\texttt{/gtr/extra edges=\{\langle parents \rangle\}\{\langle children \rangle\}\{\langle edge options \rangle\}  \hspace{1cm} \text{(style, no default)}}

Appends an extra set of edges to the current family. The edges are drawn between the given \texttt{\langle parents \rangle} list and the given \texttt{\langle children \rangle} list using the \texttt{\langle edge options \rangle}. Note that parents and children are defined by their \texttt{/gtr/id=\{P,90\}} values. They do not necessarily have to be \textit{real} members of the current family. The current family is given by a surrounding \texttt{/gtr/family=\{P,103\}} or \texttt{/gtr/options for family=\{P,102\}}.

\begin{tikzpicture}
\genealogytree
\[ \text{options for family=\{SmithDoe\}\{extra edges=\{Jane1982,Arth2008,Char2014\}\{foreground=\{red,line width=2pt,-Latex\},background=\{yellow,line width=5pt\}\}\},} \]
\{input\{example.option.graph\}\}
\end{tikzpicture}

\texttt{/gtr/family extra edges=\{\langle parents \rangle\}\{\langle childs \rangle\}\{\langle edge options \rangle\}  \hspace{1cm} \text{(style, no default)}}

This is a shortcut for embedding \texttt{/gtr/extra edges} into \texttt{/gtr/family=\{P,103\}}.
This is a shortcut for embedding `/gtr/extra edges for family` into `/gtr/options for family`.

```
\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}
```

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⟩}`.
/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 \textit{parents} list and the given \textit{children} list using the \textit{edge options}. This is identical to /gtr/extra edges \textsuperscript{P.214}, but the drawing lies under the normal edges.

\begin{tikzpicture}
\genealogytree
\nosetfamilyoptions{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 \textsuperscript{P.103}.
This is a shortcut for embedding /gtr/extra edges prepend for family\(^{\text{P.216}}\) into /gtr/options for family\(^{\text{P.102}}\).

\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)\}
/tikz/genealogytree extra edges scope (style, initially empty)

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. 195.

\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.
8.8 Edge Shifting

Typically, the edge core position of a family is located on half the way between the parents and the children. But, this is true only, if no \texttt{union} constructs are used, where a man or woman has two or more relationships with or without children. For e.g. two relationships, the edge core positions are located on one-third and on two-thirds of the way between the parents and the children. This edge shift is computed automatically.

\begin{genealogypicture}[template=signpost]
  child{
    p{Husband1}
    g{Wife} c{Child1} c{Child2}
    union{ p{Husband2} c{Child3} c{Child4} c{Child5} c{Child6} }
  }
\end{genealogypicture}

Depending on the situation, the edges may cross or not.

\begin{genealogypicture}[template=signpost]
  child{
    p{Husband1}
    g{Wife} c{Child1} c{Child2} c{Child3} c{Child4}
    union{ p{Husband2} c{Child5} }
  }
\end{genealogypicture}

The automatic shift does not prevent edge crossing, but ensures to a certain degree that one edge does not hide another edge.
Nodes could be shifted inside their level (here horizontally) to influence the edge drawing, see /gtr/pivot $^\text{P.95}$, /gtr/pivot shift $^\text{P.104}$, /gtr/distance $^\text{P.94}$, etc.

\begin{genealogypicture}[template=signpost]
  \child{p{Husband1}
    g{Wife} c{Child1} c{Child2} c{Child3} c[pivot]{Child4}
    union{ p{Husband2} c{Child5} }
  }
\end{genealogypicture}

In the following, options for manual edge shifting to avoid edge crossings are described.

/gtr/edges shift=$(length)$

(style, no default)

Shifts the edge core position by \textit{(length)} towards the parents. In contrast to /gtr/edge/xshift $^\text{P.203}$ and /gtr/edge/yshift $^\text{P.204}$, the /gtr/timeflow $^\text{P.78}$ is automatically respected. Also note that this is a /gtr/family edges $^\text{P.193}$ option, i.e. it can be applied directly inside the option list of a family.

\begin{genealogypicture}[template=signpost]
  \child[edges shift=-3mm]{p{Husband1}
    g{Wife} c{Child1} c{Child2} c{Child3} c{Child4}
    union[edges shift=3mm]{ p{Husband2} c{Child5} }
  }
\end{genealogypicture}
Shifts the edge core position by fraction \(<\text{number}/\text{parts}\) of the /gtr/level distance\(^{\text{P.81}}\) towards the parents. This is a style which uses /gtr/edges shift\(^{\text{P.220}}\).

\begin{genealogypicture}[template=signpost]
  child[edges down by=2 of 4]{
    p{Husband1}
    g{Wife} c{Child1} c{Child2} c{Child3} c{Child4}
  }
  union[edges up by=1 of 4]{ p{Husband2} }
  union[edges up by=1 of 4]{ p{Husband3} }
\end{genealogypicture}

Shifts the edge core position by fraction \(<\text{number}/\text{parts}\) of the /gtr/level distance\(^{\text{P.81}}\) towards the children. This is a style which uses /gtr/edges shift\(^{\text{P.220}}\).

\begin{genealogypicture}[template=signpost]
  child[edges down]{
    p{Husband1}
    g{Wife} c{Child1} c{Child2} c{Child3} c{Child4}
  }
  union[edges up]{ p{Husband2} c{Child5} }
\end{genealogypicture}

Shifts the edge core position by one third of the /gtr/level distance\(^{\text{P.81}}\) towards the parents. This is a special case of /gtr/edges up by usable for two relationships.

\begin{genealogypicture}[template=signpost]
  child[edges down]{
    p{Husband1}
    g{Wife} c{Child1} c{Child2} c{Child3} c{Child4}
  }
  union[edges up]{ p{Husband2} c{Child5} }
\end{genealogypicture}

Shifts the edge core position by one third of the /gtr/level distance\(^{\text{P.81}}\) towards the children. This is a special case of /gtr/edges up by usable for two relationships.

\begin{genealogypicture}[template=signpost]
  child[edges down]{
    p{Husband1}
    g{Wife} c{Child1} c{Child2} c{Child3} c{Child4}
  }
  union[edges up]{ p{Husband2} c{Child5} }
\end{genealogypicture}
The following options can only be applied globally for a graph or the whole document. They cannot be applied locally to influence just one family.

/gtr/reset edge level shift  
This sets the default algorithm for the automatic level shift. The first relationship is closest to the parents while the last relationship is closest to the children.

\begin{genealogypicture}[template=signpost,reset edge level shift]  
child{  
  p{Husband1}  
  g{Wife} c{Child1} c{Child2} c{Child3} c{Child4}  
  union{ p{Husband2} c{Child5} }  
}  
\end{genealogypicture}

/gtr/switch edge level shift  
This switches the algorithm for the automatic level shift. The first relationship is closest to the children while the last relationship is closest to the parents.

\begin{genealogypicture}[template=signpost,switch edge level shift]  
child{  
  p{Husband1}  
  g{Wife} c{Child1} c{Child2} c{Child3} c{Child4}  
  union{ p{Husband2} c{Child5} }  
}  
\end{genealogypicture}
Sets all edge core positions to be on half the way between the parents and the children. Note that edge overlapping has to be prevented *manually* for all families. Use this option only, if you are really sure about that.

\begin{genealogypicture}[template=signpost,nullify edge level shift]
  child{
    p{Husband1}
    g{Wife} c{Child1} c{Child2} c{Child3} c{pivot}{Child4}
    union{    p{Husband2} c{Child5}  }
  }
\end{genealogypicture}
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 \texttt{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 \texttt{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 \textit{after} the package is loaded and \textit{before} \texttt{\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 \texttt{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.

\begin{verbatim}
\gtrSymBorn,14.XI.1475
{\
  \gtrSymbolsSetCreate{red}\
  \gtrSymMarried,22.II.1502,\
  \gtrSymDied,8.X.1553,\
}
\gtrSymBuried,10.X.1553
\end{verbatim}

\gtrSymBorn, 14.XI.1475 \gtrSymMarried, 22.II.1502, \gtrSymDied, 8.X.1553, \gtrSymBuried, 10.X.1553

\gtrSymbolsSetCreateSelected{(color)}\{\langle list\rangle\}

Recreates all symbols from the given comma separated (list) for the current \TeX group with the given (color). The named color \texttt{gtrsymbol} will also be set to (color). The (list) contains the base names of the selected symbols, e.g. \texttt{Born} for \texttt{\gtrSymBorn}. 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.

\begin{verbatim}
\gtrSymbolsSetCreateSelected{blue}{Male}\
\gtrSymbolsSetCreateSelected{red}{Female}\
\gtrSymbolsSetCreateSelected{yellow!50!black}{Born,Died}\
\end{verbatim}

\gtrSymBorn, \texttt{gtrsymMale}, \gtrSymFemale, \gtrSymNeuter, \gtrSymDied.

\gtrSymBorn, \texttt{♂}, \texttt{♀}, \texttt{♂}, \texttt{♀}.

\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 \texttt{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.

\begin{verbatim}
\gtrSymBorn,14.XI.1475
{\
  \gtrSymbolsSetDraw{red}\
  \gtrSymMarried,22.II.1502,\
  \gtrSymbolsSetDraw{blue}\
  \gtrSymDied,8.X.1553,\
}
\gtrSymBuried,10.X.1553
\end{verbatim}

\gtrSymBorn, 14.XI.1475 \gtrSymMarried, 22.II.1502, \gtrSymDied, 8.X.1553, \gtrSymBuried, 10.X.1553

226
### 9.2 List of Symbols

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{\textbf{\texttt{\textbackslash{gtrsymBorn}}}} \star</td>
<td>Birth / born (Unicode U+2A)</td>
<td>Johann Maier \star 14.XI.1475</td>
</tr>
<tr>
<td>\text{\textbf{\texttt{\textbackslash{gtrsymBornoutofwedlock}}}} (\star)</td>
<td>Born out of wedlock / illegitimate.</td>
<td>Johann Maier (\star) 14.XI.1475</td>
</tr>
<tr>
<td>\text{\textbf{\texttt{\textbackslash{gtrsymStillborn}}}} ††</td>
<td>Stillborn</td>
<td>\textit{Anonymus} Maier †† 14.XI.1475</td>
</tr>
<tr>
<td>\text{\textbf{\texttt{\textbackslash{gtrsymDiedonbirthday}}}} ⋆†</td>
<td>Died on the birthday</td>
<td>Johann Maier ⋆† 14.XI.1475</td>
</tr>
<tr>
<td>\text{\textbf{\texttt{\textbackslash{gtrsymBaptized}}}} ≈</td>
<td>Baptism / baptized (Unicode U+2248)</td>
<td>Johann Maier ≈ 14.XI.1475</td>
</tr>
<tr>
<td>Event</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>\texttt{\LaTeX{}Engaged}</td>
<td>Engagement / engaged (Unicode U+26AC).</td>
<td>14.XI.1475</td>
</tr>
<tr>
<td>\texttt{\LaTeX{}Married}</td>
<td>Marriage / married (Unicode U+26AD).</td>
<td>14.XI.1475</td>
</tr>
<tr>
<td>\texttt{\LaTeX{}Divorced}</td>
<td>Divorce / divorced (Unicode U+26AE).</td>
<td>14.XI.1475</td>
</tr>
<tr>
<td>\texttt{\LaTeX{}Partnership}</td>
<td>Partnership / unmarried (Unicode U+26AF).</td>
<td>14.XI.1475</td>
</tr>
<tr>
<td>\texttt{\LaTeX{}Died}</td>
<td>Death / died (Unicode U+2020, U+271D).</td>
<td>14.XI.1475</td>
</tr>
<tr>
<td>\texttt{\LaTeX{}Killed}</td>
<td>Killed in action / fallen (Unicode U+2694).</td>
<td>14.XI.1475</td>
</tr>
</tbody>
</table>
**Buried**

Burial / buried (Unicode U+26B0).

<table>
<thead>
<tr>
<th>Johann Maier</th>
<th>Buried, 14.XI.1475</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johann Maier</td>
<td>14.XI.1475</td>
</tr>
</tbody>
</table>

**Funeral urn**

Funeral urn / cremated (Unicode U+26B1).

<table>
<thead>
<tr>
<th>Johann Maier</th>
<th>Funeral urn, 14.XI.1475</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johann Maier</td>
<td>14.XI.1475</td>
</tr>
</tbody>
</table>

**Floruit**

Floruit / flourished.

<table>
<thead>
<tr>
<th>Johann Maier</th>
<th>Floruit, 1475--1503</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johann Maier</td>
<td>1475–1503</td>
</tr>
</tbody>
</table>

**Female**

Female (Unicode U+2640).

<table>
<thead>
<tr>
<th>Maria Maier</th>
<th>Female</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maria Maier</td>
<td></td>
</tr>
</tbody>
</table>

**Male**

Male (Unicode U+2642).

<table>
<thead>
<tr>
<th>Johann Maier</th>
<th>Male</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johann Maier</td>
<td></td>
</tr>
</tbody>
</table>

**Neuter**

Neuter / Unknown sex (Unicode U+26B2).

<table>
<thead>
<tr>
<th>Anonymus Maier</th>
<th>Neuter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymus Maier</td>
<td></td>
</tr>
</tbody>
</table>
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\textsuperscript{P.231} 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.

\begin{verbatim}
\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.
\end{verbatim}

\gtrSymbolsRecordReset
Identical to \gtrSymbolsRecordReset. This option is useful for application inside \genealogytree\textsuperscript{P.55} or \genealogypicture\textsuperscript{P.57}. See \gtrSymbolsLegend\textsuperscript{P.231} for an example.

\gtrSymbolsFullLegend[(language)]
Prints a full unabridged legend to symbols according to \gtr/language\textsuperscript{P.233} or optionally according to \texttt{language}.

\begin{verbatim}
\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.
\end{verbatim}
\gtrSymbolsLegend[(language)]
Prints a legend to symbols according to /gtr/language \textsuperscript{P.233} or optionally according to \\langle language \rangle. The legend only contains these symbols which were actually used inside the document so far or since the last call to \gtrSymbolsRecordReset \textsuperscript{P.230} or /gtr/symbols record reset \textsuperscript{P.239}.

\begin{genealogy}{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\par \gtrsymKilled-12.X.1764}
p{Jane Bowden \par \gtrsymBorn-1724\par \gtrsymDied-7.VII.1802 \\\\\\\\tcbline \gtrsymMarried-2.III.1742\par}
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{Michael Smith \par \gtrsymDiedonbirthday-1.III.1761}
}\end{genealogy}

9.3.2 Description Texts and Language Settings
The following options of the key family /gtr/sym\textsubscript{lang}/ are language dependent and can be set globally by /gtr/language \textsuperscript{P.233}.

\begin{center}
\begin{tabular}{|c|c|}
\hline
english & \\hline
\end{tabular}
\end{center}

\% ... 
\gtrSymbolsFullLegend

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textsuperscript{\star}=born, \(\heartsuit\)=born out of wedlock, \(\bigstar\)=stillborn, \(\bigstar\)=died on the birthday, \(\approx\)=baptized, \(\odot\)=engaged, \(\otimes\)=married, \(\bigodot\)=divorced, \(\bigodot\=unmarried partnership, \(\mathfrak{s}\)=floruit, \(\dagger\)=died, \(\bigstar\)=killed in action, \(\varnothing\)=female, \(\sigma\)=male, \(\varnothing\)=neuter. 
\end{tabular}
\end{center}
\gtrset{language=german}
\%
\gtrSymbolsFullLegend

\gtr/symlang/Born={(text)} (no default, initially born)
Legend \textit{(text)} used for \textit{★}.

\gtr/symlang/Bornoutofwedlock={(text)} (no default, initially born out of wedlock)
Legend \textit{(text)} used for \textit{★★}.

\gtr/symlang/Stillborn={(text)} (no default, initially stillborn)
Legend \textit{(text)} used for \textit{★†}.

\gtr/symlang/Diedonbirthday={(text)} (no default, initially died on the birthday)
Legend \textit{(text)} used for \textit{★†}.

\gtr/symlang/Baptized={(text)} (no default, initially baptized)
Legend \textit{(text)} used for \textit{≈}.

\gtr/symlang/Engaged={(text)} (no default, initially engaged)
Legend \textit{(text)} used for \textit{○}.

\gtr/symlang/Married={(text)} (no default, initially married)
Legend \textit{(text)} used for \textit{○○}.

\gtr/symlang/Divorced={(text)} (no default, initially divorced)
Legend \textit{(text)} used for \textit{○○○}.

\gtr/symlang/Partnership={(text)} (no default, initially unmarried partnership)
Legend \textit{(text)} used for \textit{○○○}.

\gtr/symlang/Floruit={(text)} (no default, initially floruit)
Legend \textit{(text)} used for \textit{♀}.

\gtr/symlang/Died={(text)} (no default, initially died)
Legend \textit{(text)} used for \textit{†}.

\gtr/symlang/Killed={(text)} (no default, initially killed in action)
Legend \textit{(text)} used for \textit{♀}.  

\gtr/symlang/Buried={(text)} (no default, initially buried)
Legend \textit{(text)} used for \textit{♀}.

\gtr/symlang/Funeralurn={(text)} (no default, initially cremated)
Legend \textit{(text)} used for \textit{♀}.

\gtr/symlang/Female={(text)} (no default, initially female)
Legend \textit{(text)} used for \textit{♀}.

\gtr/symlang/Male={(text)} (no default, initially male)
Legend \textit{(text)} used for \textit{♂}.

\gtr/symlang/Neuter={(text)} (no default, initially neutral)
Legend \textit{(text)} used for \textit{♀}.  

232
The document *genealogytree-languages.pdf* displays the effects of language specific settings.

### 10.1 Preamble Settings

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 \texttt{\gtrloadlanguage} \texttt{\textcolor{red}{P. 234}}.
- If this option is not used at all, the \texttt{english} language is set.

Feasible values for \texttt{(language)} are:

- \texttt{danish} (Translation provided by Mikkel Eide Eriksen)
- \texttt{dutch} (Translation provided by Dirk Bosmans)
- \texttt{english}
- \texttt{french} (Translation provided by Denis Bitouzé)
- \texttt{german} with variants:  
  - \texttt{german-german}
  - \texttt{german-austrian}
- \texttt{italian} (Translation provided by Andrea Vaccari)
- \texttt{spanish} (Translation provided by Francisco G. Pérez Sánchez)
- \texttt{swedish} (Translation provided by Per Starbäck)

\begin{verbatim}
\documentclass{...}
%
\gtrset{language=german-austrian}
%
\begin{document}
%
\end{document}
\end{verbatim}

The current language name is stored inside \texttt{\gtrlanguagename}.

The current language is ‘\texttt{\gtrlanguagename}’.

The current language is ‘english’.
The \texttt{/gtr/language} option sets various keys for description texts. These texts can be customized selectively, if needed.

\begin{verbatim}
\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}
\end{verbatim}

\begin{itemize}
\item 1775, \textsuperscript{†} 1832. \quad (★=geboren, †=gestorben.)
\item 1775, \textsuperscript{†} 1832. \quad (★=geb., †=gestorben.)
\item 1775, \textsuperscript{†} 1832. \quad (★=born, †=died.)
\end{itemize}

\texttt{\gtrloadlanguage{list of languages}}

Loads a comma separated \texttt{⟨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 \texttt{/gtr/language} \textsuperscript{P. 233} inside the document. For a list of feasible language names, see \texttt{/gtr/language} \textsuperscript{P. 233}.

\begin{verbatim}
\documentclass{...}
%...
\gtrloadlanguage{english, german}
%...
\begin{document}
%...
\end{document}
\end{verbatim}

\section*{10.2 Document Settings}

Switching between languages inside the document is done by setting \texttt{/gtr/language} \textsuperscript{P. 233}. Note that every language to be used has to be loaded inside the preamble by \texttt{\gtrloadlanguage}.
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 \emph{(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 \emph{(options)} are checked by setting them and they are logged in the produced list.
\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 295.
### 11.2 Processor Debugging

\gtrprocessordebug\{(options)\}\{(graph content)\}

Processes the given \textit{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 \textit{options} are set for processing.

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

**Genealogytree Processor Debugger**

<table>
<thead>
<tr>
<th>Family 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: par</td>
</tr>
<tr>
<td>id: (none)</td>
</tr>
<tr>
<td>fam: (none)</td>
</tr>
<tr>
<td>offset: 0.0pt</td>
</tr>
<tr>
<td>pos: 7.11319pt</td>
</tr>
<tr>
<td>cwest@anchor: 1</td>
</tr>
<tr>
<td>ceast@anchor: 4</td>
</tr>
<tr>
<td>g: 2</td>
</tr>
<tr>
<td>par: 5, 6</td>
</tr>
<tr>
<td>chi: 1, 2, 3, 4</td>
</tr>
<tr>
<td>patpar: 5, 6</td>
</tr>
<tr>
<td>patchi: 1, 2, 3, 4</td>
</tr>
<tr>
<td>union: (none)</td>
</tr>
<tr>
<td>ps: 0pt</td>
</tr>
<tr>
<td>x: 0.0pt</td>
</tr>
<tr>
<td>y: (none)</td>
</tr>
<tr>
<td>frac: 0.5</td>
</tr>
<tr>
<td>opt@family: (none)</td>
</tr>
<tr>
<td>opt@subtree: (none)</td>
</tr>
</tbody>
</table>

**Parents of Family 1**
<table>
<thead>
<tr>
<th>Person 5</th>
<th>Person 6</th>
<th>Childs of Family 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>id: pE</td>
<td>id: pF</td>
<td>id: pB</td>
</tr>
<tr>
<td>fam: 1</td>
<td>fam: 1</td>
<td>fam: 1</td>
</tr>
<tr>
<td>chief: (none)</td>
<td>chief: (none)</td>
<td>chief: 1</td>
</tr>
<tr>
<td>parof: 1</td>
<td>parof: 1</td>
<td>parof: (none)</td>
</tr>
<tr>
<td>x: 0.0pt</td>
<td>x: 76.82242pt</td>
<td>x: -72.5545pt</td>
</tr>
<tr>
<td>y: 113.811pt</td>
<td>y: 113.811pt</td>
<td>y: 0.0pt</td>
</tr>
<tr>
<td>dim: 71.13188pt</td>
<td>dim: 71.13188pt</td>
<td>dim: 71.13188pt</td>
</tr>
<tr>
<td>cwest@val: 0.0pt</td>
<td>cwest@val: 76.82242pt</td>
<td>cwest@val: -72.5545pt</td>
</tr>
<tr>
<td>cwest@next: (none)</td>
<td>cwest@next: (none)</td>
<td>cwest@next: 5</td>
</tr>
<tr>
<td>cwest@thread: (none)</td>
<td>cwest@thread: (none)</td>
<td>cwest@thread: (none)</td>
</tr>
<tr>
<td>cwest@thread: (none)</td>
<td>cwest@thread: (none)</td>
<td>cwest@thread: (none)</td>
</tr>
<tr>
<td>ceast@val: 71.13188pt</td>
<td>ceast@val: 147.9543pt</td>
<td>ceast@val: -1.42262pt</td>
</tr>
<tr>
<td>ceast@next: (none)</td>
<td>ceast@next: (none)</td>
<td>ceast@next: 6</td>
</tr>
<tr>
<td>ceast@thread: (none)</td>
<td>ceast@thread: (none)</td>
<td>ceast@thread: (none)</td>
</tr>
<tr>
<td>ceast@thread: (none)</td>
<td>ceast@thread: (none)</td>
<td>ceast@thread: (none)</td>
</tr>
</tbody>
</table>
Person 2

id: pA
fam: 1
chief: 1
parof: ⟨none⟩
x: 1.42264pt
y: 0.0pt
dim: 71.13188pt
cwest@val: 1.42264pt
cwest@next: ⟨none⟩
cwest@thread: ⟨none⟩
cwest@tgap: ⟨none⟩
ceast@val: 72.55452pt
ceast@next: 6
ceast@thread: ⟨none⟩
ceast@tgap: ⟨none⟩

Person 3

id: pC
fam: 1
chief: 1
parof: ⟨none⟩
x: 75.39978pt
y: 0.0pt
dim: 71.13188pt
cwest@val: 75.39978pt
cwest@next: ⟨none⟩
cwest@thread: ⟨none⟩
cwest@tgap: ⟨none⟩
ceast@val: 146.53166pt
ceast@next: 6
ceast@thread: ⟨none⟩
ceast@tgap: ⟨none⟩

Person 4

id: pD
fam: 1
chief: 1
parof: ⟨none⟩
x: 149.37692pt
y: 0.0pt
dim: 71.13188pt
cwest@val: 149.37692pt
cwest@next: ⟨none⟩
cwest@thread: ⟨none⟩
cwest@tgap: ⟨none⟩
ceast@val: 220.5088pt
ceast@next: 6
ceast@thread: ⟨none⟩
ceast@tgap: ⟨none⟩

End of Genealogytree Processor Debugger
\texttt{\textbackslash{}gtrprocessordebuginput\{(options)\}\{\textbackslash{}file name\}}

Loads the file denoted by \textbackslash{}textit{\{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 \textbackslash{}textit{\{options\}} are set for processing.

The following example uses the graph from Section \textbf{14.1} on page 295.

\begin{center}
\texttt{\textbackslash{}gtrprocessordebuginput\{example.option.graph\}}
\end{center}

\begin{table}[h]
\centering
\begin{tabular}{|l|}
\hline
\textbf{Genealogytree Processor Debugger} \\
\hline
\textbf{Family 1} \\
\hline
\begin{tabular}{|l|}
\hline
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 \\
\hline
\end{tabular}
\hline
\textbf{Parents of Family 1} \\
\hline
\begin{tabular}{|l|}
\hline
\textbf{Person 4} \\
\hline
id: John1980 & identifier (also node alias) \\
fam: 2 & enclosing family \\
chif: 2 & child of family \\
parf: 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 \\
\hline
\end{tabular}
\end{tabular}
\end{table}

John Smith ★ 1980
Person 7

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id:</td>
<td>Jane1982</td>
</tr>
<tr>
<td>fam:</td>
<td>3</td>
</tr>
<tr>
<td>chiof:</td>
<td>3</td>
</tr>
<tr>
<td>parof:</td>
<td>1</td>
</tr>
<tr>
<td>x:</td>
<td>157.91275pt</td>
</tr>
<tr>
<td>y:</td>
<td>113.811pt</td>
</tr>
<tr>
<td>dim:</td>
<td>71.13188pt</td>
</tr>
<tr>
<td>cwest@val:</td>
<td>1.42264pt</td>
</tr>
<tr>
<td>cwest@thread:</td>
<td>(none)</td>
</tr>
<tr>
<td>cwest@tgap:</td>
<td>(none)</td>
</tr>
<tr>
<td>ceast@val:</td>
<td>72.55452pt</td>
</tr>
<tr>
<td>ceast@thread:</td>
<td>(none)</td>
</tr>
<tr>
<td>ceast@tgap:</td>
<td>(none)</td>
</tr>
</tbody>
</table>

Jane Doe ★ 1982

Childs of Family 1

Person 1

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id:</td>
<td>Arth2008</td>
</tr>
<tr>
<td>fam:</td>
<td>1</td>
</tr>
<tr>
<td>chiof:</td>
<td>1</td>
</tr>
<tr>
<td>parof:</td>
<td>(none)</td>
</tr>
<tr>
<td>x:</td>
<td>24.18484pt</td>
</tr>
<tr>
<td>y:</td>
<td>0.0pt</td>
</tr>
<tr>
<td>dim:</td>
<td>71.13188pt</td>
</tr>
<tr>
<td>cwest@val:</td>
<td>24.18484pt</td>
</tr>
<tr>
<td>cwest@thread:</td>
<td>(none)</td>
</tr>
<tr>
<td>cwest@tgap:</td>
<td>(none)</td>
</tr>
<tr>
<td>ceast@val:</td>
<td>95.31673pt</td>
</tr>
<tr>
<td>ceast@thread:</td>
<td>(none)</td>
</tr>
<tr>
<td>ceast@tgap:</td>
<td>(none)</td>
</tr>
</tbody>
</table>

Arthur ★ 2008

Person 2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id:</td>
<td>Bert2010</td>
</tr>
<tr>
<td>fam:</td>
<td>1</td>
</tr>
<tr>
<td>chiof:</td>
<td>1</td>
</tr>
<tr>
<td>parof:</td>
<td>(none)</td>
</tr>
<tr>
<td>x:</td>
<td>98.16199pt</td>
</tr>
<tr>
<td>y:</td>
<td>0.0pt</td>
</tr>
<tr>
<td>dim:</td>
<td>71.13188pt</td>
</tr>
<tr>
<td>cwest@val:</td>
<td>98.16199pt</td>
</tr>
<tr>
<td>cwest@thread:</td>
<td>(none)</td>
</tr>
<tr>
<td>cwest@tgap:</td>
<td>(none)</td>
</tr>
<tr>
<td>ceast@val:</td>
<td>169.29387pt</td>
</tr>
<tr>
<td>ceast@thread:</td>
<td>(none)</td>
</tr>
<tr>
<td>ceast@tgap:</td>
<td>(none)</td>
</tr>
</tbody>
</table>

Berta ★ 2010
Person 3

- **id**: Char2014 (identifier (also node alias))
- **fam**: 1 (enclosing family)
- **chief**: 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)
- **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)

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  
fam: 2  
chief: (none)  
parof: 2  
  
x: 0.0pt  
y: 227.62201pt  
dim: 71.13188pt  
cwest@val: 0.0pt  
cwest@next: (none)  
cwest@thread: (none)  
cwest@tgap: (none)  
ceast@val: 71.13188pt  
ceast@next: (none)  
ceast@thread: (none)  
ceast@tgap: (none)  

Grandpa Smith  
1949

Person 6  
id: GmSm1952  
fam: 2  
chief: (none)  
parof: 2  
  
x: 76.82242pt  
y: 227.62201pt  
dim: 71.13188pt  
cwest@val: 76.82242pt  
cwest@next: (none)  
cwest@thread: (none)  
cwest@tgap: (none)  
ceast@val: 147.9543pt  
ceast@next: (none)  
ceast@thread: (none)  
ceast@tgap: (none)  

Grandma Smith  
1952

Childs of Family 2

Person 4  
id: John1980  
fam: 2  
chief: 2  
parof: 1  
  
x: 38.41121pt  
y: 113.8111pt  
dim: 71.13188pt  
cwest@val: 38.41121pt  
cwest@next: 5  
cwest@thread: (none)  
cwest@tgap: (none)  
ceast@val: 109.54309pt  
ceast@next: 6  
ceast@thread: (none)  
ceast@tgap: (none)  

John Smith  
1980
Family 3

<table>
<thead>
<tr>
<th>Type of family: par</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier: Doe</td>
</tr>
<tr>
<td>Enclosing family: 1</td>
</tr>
<tr>
<td>Offset: 156.49011pt</td>
</tr>
<tr>
<td>Position: 120.9242pt</td>
</tr>
<tr>
<td>West contour starting node: 7</td>
</tr>
<tr>
<td>East contour starting node: 8</td>
</tr>
<tr>
<td>G-node of the family: 7</td>
</tr>
<tr>
<td>Parent nodes: 9, 10</td>
</tr>
<tr>
<td>Child nodes: 7, 8</td>
</tr>
<tr>
<td>Patchwork parent nodes: 9, 10</td>
</tr>
<tr>
<td>Patchwork child nodes: 7, 8</td>
</tr>
<tr>
<td>Further partner families: (none)</td>
</tr>
<tr>
<td>Pivot shift length (parents vs childs): 0pt</td>
</tr>
<tr>
<td>Anchor x: 156.49011pt</td>
</tr>
<tr>
<td>Anchor y: (none)</td>
</tr>
<tr>
<td>Line positioning fraction: 0.5</td>
</tr>
<tr>
<td>Options for the family: (none)</td>
</tr>
<tr>
<td>Options for the subtree: (none)</td>
</tr>
</tbody>
</table>

Parents of Family 3

Person 9

| Identifier (also node alias): GpDo1955 |
| Enclosing family: 3 |
| Child of family: (none) |
| Parent of family: 3 |
| Anchor x: 156.49011pt |
| Anchor y: 227.62201pt |
| Width (or height): 71.13188pt |
| West contour value: 0.0pt |
| West contour successor: (none) |
| West contour thread: (none) |
| West contour thread gap: (none) |
| East contour value: 71.13188pt |
| East contour successor: (none) |
| East contour thread: (none) |
| East contour thread gap: (none) |

Person 10

| Identifier (also node alias): GmDo1956 |
| Enclosing family: 3 |
| Child of family: (none) |
| Parent of family: 3 |
| Anchor x: 233.31253pt |
| Anchor y: 227.62201pt |
| Width (or height): 71.13188pt |
| West contour value: 76.82242pt |
| West contour successor: (none) |
| West contour thread: (none) |
| West contour thread gap: (none) |
| East contour value: 147.9543pt |
| East contour successor: (none) |
| East contour thread: (none) |
| East contour thread gap: (none) |
## Childs of Family 3

<table>
<thead>
<tr>
<th>Person 7</th>
<th></th>
<th>Person 8</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id: Jane1982</td>
<td>identifier (also node alias)</td>
<td>id: Harr1987</td>
<td>identifier (also node alias)</td>
</tr>
<tr>
<td>fam: 3</td>
<td>enclosing family</td>
<td>fam: 3</td>
<td>child of family</td>
</tr>
<tr>
<td>chiof: 3</td>
<td>child of family</td>
<td>chiof: 3</td>
<td>child of family</td>
</tr>
<tr>
<td>parof: 1</td>
<td>parent of family</td>
<td>parof: ⟨none⟩</td>
<td>parent of family</td>
</tr>
<tr>
<td>x: 157.91275pt</td>
<td>x anchor</td>
<td>x: 231.8899pt</td>
<td>x anchor</td>
</tr>
<tr>
<td>y: 113.811pt</td>
<td>y anchor</td>
<td>y: 113.811pt</td>
<td>y anchor</td>
</tr>
<tr>
<td>dim: 71.13188pt</td>
<td>width (or height)</td>
<td>dim: 71.13188pt</td>
<td>width (or height)</td>
</tr>
<tr>
<td>cwest@val: 1.42264pt</td>
<td>west contour value</td>
<td>cwest@val: 75.39978pt</td>
<td>west contour value</td>
</tr>
<tr>
<td>cwest@next: 9</td>
<td>west contour successor</td>
<td>cwest@next: ⟨none⟩</td>
<td>west contour successor</td>
</tr>
<tr>
<td>cwest@thread: ⟨none⟩</td>
<td>west contour thread</td>
<td>cwest@thread: ⟨none⟩</td>
<td>west contour thread</td>
</tr>
<tr>
<td>ceast@val: 72.55452pt</td>
<td>east contour value</td>
<td>ceast@val: 146.53166pt</td>
<td>east contour value</td>
</tr>
<tr>
<td>ceast@next: 10</td>
<td>east contour successor</td>
<td>ceast@next: 10</td>
<td>east contour successor</td>
</tr>
<tr>
<td>ceast@thread: ⟨none⟩</td>
<td>east contour thread</td>
<td>ceast@thread: ⟨none⟩</td>
<td>east contour thread</td>
</tr>
<tr>
<td>ceast@tgap: ⟨none⟩</td>
<td>east contour thread gap</td>
<td>ceast@tgap: ⟨none⟩</td>
<td>east contour thread gap</td>
</tr>
</tbody>
</table>
11.3 Graphical Debugging

\gtrdebugdrawcontour{(options)}{path options}

After a \genealogytree \textsuperscript{P. 55} is drawn inside a \texttt{tikzpicture} environment, the auto-layout contour lines of a family can be displayed with this macro. For \textit{(options)}, the keys /\texttt{gtr/debug/family number} \textsuperscript{P. 248}, /\texttt{gtr/debug/family id} \textsuperscript{P. 248}, /\texttt{gtr/debug/contour} \textsuperscript{P. 249} may be used to specify the family and the contour lines to draw. The \textit{(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 \texttt{union} families are not displayed, since they are melted to their embedding \texttt{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}
\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}

\begin{tikzpicture}
genealogytree[template=formal graph,show family]
{ 
  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}
The two contour lines are always referred to as west and east contour lines independent of the \texttt{/gtr/timeflow} 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} 
      } 
   } 
\end{tikzpicture}

\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} 
      } 
   } 
\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)
  }
}
\end{tikzpicture}
11.4 Show Information

Note that most options in this section only work, if a /gtr/processing 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 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 
Shows the level numbers of every node. This information can be used for setting /gtr/level \textsuperscript{P. 107} and /gtr/level n \textsuperscript{P. 108}.

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

/gtr/show number 
Shows the internal numbers of every node and every family. It is strongly recommended to reference a node by a chosen /gtr/id \textsuperscript{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}
The library is loaded by a package option or inside the preamble by:

\gtruselibrary{templates}

### 12.1 Using Templates

\textbf{/gtr/template=⟨name⟩} \quad \text{(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’

\textbf{template=\texttt{formal graph}}

This style is based on \texttt{/gtr/processing\textsuperscript{P.128}=\texttt{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'

```
\begin{tikzpicture}
genealogytree[template=signpost]{
    child{
        g{Father}
        p{Mother}
        c{Child 1}
        c{Child 2}
        c{Child 3}
    }
}
\end{tikzpicture}
```

This style is based on /gtr/processing\textsuperscript{P.128}=fit. For further examples, see Section 5.2 on page 78 and many more.

12.4 Template 'symbol nodes'

```
\begin{tikzpicture}
genealogytree[template=symbol nodes]{
    child{
        gm pf cf
        child{gm pf cf cm}
        child{gm pf cm c- cm}
    }
}
\end{tikzpicture}
```

This style is based on /gtr/processing\textsuperscript{P.128}=tcbox*. For the content, a single token \texttt{m} selects a male node (also \texttt{male}), a single token \texttt{f} selects a female node (also \texttt{female}), and every other token selects a neuter node. The symbol coloring with \texttt{gtrSymbolsSetCreateSelected}\textsuperscript{P.226} has to be done before entering a \texttt{tikzpicture} environment.

\texttt{\gtrSymbolsSetCreateSelected{blue}{Male}}
\texttt{\gtrSymbolsSetCreateSelected{red}{Female}}
\texttt{\gtrSymbolsSetCreateSelected{black}{Neuter}}
12.5 Template 'tiny boxes'

```
\begin{tikzpicture}
genealogytree[template=tiny boxes]{
child{
  g-p-c-
  child{g-p-c-c-}
  child{g-p-c-c-c-}
}
}
\end{tikzpicture}
```

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 285.

12.6 Template 'tiny circles'

```
\begin{tikzpicture}
genealogytree[template=tiny circles]{
child{
  g-c-
  child{g-c-c-}
  child{g-c-c-c-}
}
}
\end{tikzpicture}
```

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 285.
12.7 Template 'directory'

This style is based on /gtr/processing\textsuperscript{P.128}=tcbox* and sets /gtr/timeflow\textsuperscript{P.78}=right. Note that optimal level sizes have to be set manually.
12.8 Template 'database pole'

This style is based on \texttt{/gtr/processing}\footnote{P. 180} = \texttt{database} and sets \texttt{/gtr/database format}\footnote{P. 185} = \texttt{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 296.

\begin{genealogypicture}
\[template=database pole\]
input{example.database.graph}
\end{genealogypicture}

\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.
12.9 Template 'database pole reduced'

This style is identical to database pole (Section 12.8 on page 259), 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.

- \( \text{\LaTeX}/\text{\LaTeX} \) 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 \( \text{\LaTeX} \) 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{verbatim}
\begin{tikzpicture}
    spy on (0.6,-0.1) in node [left] at (7,1);
    \spy using outlines={circle, magnification=10, size=5cm, connect spies}
\end{tikzpicture}
\genealogytreeinput[template=database pole reduced]{example.gauss.graph}
\spy on (0.6,-0.1) in node [left] at (7,1);
\end{verbatim}
12.10 Template 'database poleportrait'

This style is based on /gtr/processing\textsuperscript{\cite{128}} database and sets /gtr/database format\textsuperscript{\cite{163}} full marriage above.

In contrast to database pole (Section 12.8 on page 259), portraits are drawn, if present.

\begin{genealogypicture}
\[template=database poleportrait\]
\textbf{Charles Smith}★ ca. 1722 in London
\textcolor{red}{\textbf{October 12, 1764}} in London
\textcolor{blue}{\textbf{Copper smith, soldier. Invented the square wheel.}}

\textbf{Jane Bowden}★\textcolor{red}{\textbf{March 2, 1742}} in London
\textcolor{blue}{\textbf{1724 to 1802}}

\textbf{Abraham Bowden}★ \text{January 4, 1740 in London
\textcolor{red}{\textbf{February 23, 1740 in London}}

\textbf{Elizabeth “Liz” Smith}★ \text{February 2, 1744 in London
\textcolor{red}{\textbf{1780 to 1805 in New York}}
\textcolor{blue}{\textbf{April 12, 1812 in Boston. Had a store in Boston.}}

\textbf{Michael Smith}★\textcolor{red}{\textbf{March 1, 1758}}

\end{genealogypicture}
The next example uses the graph data from Section 2.3.5 on page 39.
12.11 Template 'database poleportrait reduced'

This style is identical to database poleportrait (Section 12.10 on page 262), but every size property is scaled by factor \( \frac{1}{10} \). See Section 12.9 on page 261 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}
\end{tikzpicture}
12.12 Template 'database portrait'

```
\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.
This style is identical to `database portrait` (Section 12.12 on page 265), but every size property is scaled by factor $\frac{1}{10}$. See Section 12.9 on page 261 for more explanations.

The following example uses the graph data from Section 2.3.5 on page 39.
12.14 Template 'database traditional'

```
\begin{genealogypicture}[template=database traditional, level size=1.3cm ]
input{example.database.graph}
\end{genealogypicture}
```

This style is based on /gtr/processing=database, sets /gtr/database format=short no marriage and /gtr/timeflow=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 usually reasonable, g nodes in families without children 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 296.

```
\begin{genealogypicture}[template=database traditional,
level size=1.3cm,node size=2.2cm,date format=yyyy,list separators={\par}{ }{}{}]
input{example.gauss.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}
```
This style is identical to \textit{database traditional} (Section 12.14 on page 268), but every size property is scaled by factor $\frac{1}{10}$. See Section 12.9 on page 261 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'

This style is based on /gtr/processing\textsuperscript{P.128}=\texttt{database}, sets /gtr/database format\textsuperscript{P.163}=\texttt{medium} and /gtr/timeflow\textsuperscript{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 296.

\begin{genealogypicture}\[template=database sideways\]input{example.database.graph}\end{genealogypicture}

Charles Smith  
ca. 1722 in London  
October 12, 1764  
Copper smith, soldier. Invented the square wheel.

Jane Bowden  
March 2, 1742 in London  
1724 to 1802

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}
12.17 Template 'database sideways reduced'

This style is identical to database sideways (Section 12.16 on page 270), but every size property is scaled by factor $\frac{1}{10}$. See Section 12.9 on page 261 for more explanations.

The following example uses the graph data from Section 2.3.5 on page 39.

```latex
\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'

This style is based on /gtr/processing P.128 = database, sets /gtr/database format P.163 = medium and /gtr/timeflow P.78 = left.

In contrast to database sideways (Section 12.16 on page 270), portraits are drawn, if present.

\begin{genealogypicture}
[template=database sidewaysportrait]
input{example.database.graph}
\end{genealogypicture}

Charles Smith
\[\star\] ca. 1722 in London
\[\dagger\] October 12, 1764
Copper smith, soldier. Invented the square wheel.

Jane Bowden
\[\odot\] March 2, 1742 in London
\[\dagger\] 1724 to 1802

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'

This style is identical to database sidewaysportrait (Section 12.18 on page 273), but every size property is scaled by factor \( \frac{1}{10} \). See Section 12.9 on page 261 for more explanations.

The following example uses the graph data from Section 2.3.5 on page 39.

\begin{tikzpicture}
  \spy on (-0.2,-0.5) in node [left] at (7,1);
  \spy using outlines={circle, magnification=10, size=5cm, connect spies}\genealogytreeinput[template=database sidewaysportrait reduced]{example.gauss.graph}
  \end{tikzpicture}
12.20 Template 'database relationship'

This style is based on /gtr/processing=P.128=database, sets /gtr/database format=P.163=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={\pref{Carl Friedrich} \surn{Gau\ss{}}},
    birth={1777-04-30}{Braunschweig (Niedersachsen)},
    death={1855-02-23}{Gö\ottingen (Niedersachsen)},
    profession={Mathematiker, Astronom, Geodät 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}{Gö\ottingen (Niedersachsen)},
    comment={Weißgerberstochter},
  }
  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}
This style is based on /gtr/processing and sets /gtr/timeflow = 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=\textit{Frederik} \textit{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=\textit{Ernest} \textit{Smith},
      birth={1870-02-02}{London},
      death={1940-02-02}{London},
      marriage={1899-02-02}{London},
      comment={Milkman}}

  parent{
    g{male,name=\textit{Dominik} \textit{Schmidt},
      birth={1840-03-03}{Berlin},
      death={1910-03-03}{London},
      marriage={1869-03-03}{Berlin},
      comment={Baker}}

    p{male,name=\textit{Christian} \textit{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}
```
12.22 Template 'ahnentafel 4'

This style is based on /gtr/processing \footnote{P.128}=database and sets /gtr/timeflow \footnote{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},
marrige={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},
marrige={1899-02-02}{London},comment={Milkman}}  
parent{  
g{male,name=\pref{Dominik} \surn{Schmidt},
birth={1840-03-03}{Berlin},death={1910-03-03}{London},
marrige={1869-03-03}{Berlin},comment={Baker}}  
parent{  
g{male,name=\pref{Christian} \surn{Schmied},
birth={1810-04-04}{Vienna},death={1870-04-04}{Vienna},
marrige={1839-04-04}{Vienna},comment={Blacksmith}}  
p{male,name=\pref{Bartholom"aus} \surn{Schmid},
birth={1780-05-05}{Eger},death={1840-05-05}{Eger},
marrige={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 1970 in New York
Used Cars Salesman.

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 1840 in Eger
Blacksmith.

Ernest Smith
* 2 Feb 1870 in London
‡ 2 Feb 1899 in London
+ 2 Feb 1940 in London
Milkman.

Frederik Smith
* 1 Jan 1900 in New York
‡ 1 Jan 1929 in New York
+ 1 Jan 1970 in New York
Used Cars Salesman.
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.
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 Schmidt
★ 4 Apr 1810 in Vienna
☀ 4 Apr 1839 in Vienna
† 4 Apr 1870 in Vienna
Blacksmith.

Bartholomäus Schmidt
★ 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}
This control sequence inserts four generations of parents with content \texttt{male} and \texttt{female}.

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

This control sequence inserts five generations of parents with content \texttt{male} and \texttt{female}.

This control sequence inserts six generations of parents with content \texttt{male} and \texttt{female}.

This control sequence inserts seven generations of parents with content \texttt{male} and \texttt{female}.

\gtrDrawSymbolicPortrait
Inserts Ti\textit{k}Z code to draw a symbolic portrait. The colors are frame and back color of a \texttt{tcolorbox}. Therefore, the intended application is inside a \texttt{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 multifold 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 288. 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 288 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 288.

\(^1\)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 287 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 287, 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.
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 \texttt{/gtr/timeflow} option appropriately.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{graph_directions}
\caption{Graph growing directions: top-down, bottom-up, right to left, left to right.}
\end{figure}
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.

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.

\(^2\)As described in detail in this document, this algorithm can be adapted with various option settings, e.g. to change the pivot alignment procedure
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 debug library documented in Chapter 11 on page 235, provides the \texttt{gtrprocessordebug} P. 238 command which displays the offset value and the different contour values. Also, \texttt{gtrdebugdrawcontour} P. 247 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.

\begin{center}
\textbf{Sewing Subtrees}
\end{center}

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 op-
tions. If (A5) is really needed for aesthetic reasons, one can simply cheat by adding some
/\texttt{gtr/distance} \footnote{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 288 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.
The following example graph files are used for various examples inside this document.

14.1 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.
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}
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

\[
\text{\texttt{gtrnewstack}}\{\texttt{name}\}\}
\]

Creates a new empty stack \textit{name}.

\[
\text{\texttt{gtrnewstack}}\{\texttt{foo}\} \quad \% \text{new empty stack}
\]

Stack size: \texttt{gtrstacksize}\{foo\}

Stack size: 0

\[
\text{\texttt{gtrstacksize}}\{\texttt{name}\}\}
\]

Returns the current stack size.

\[
\text{\texttt{gtrnewstack}}\{\texttt{foo}\}
\]

Stack size: \texttt{gtrstacksize}\{foo\}

\[
\text{\texttt{gtrstackpush}}\{\texttt{foo}\}\{\texttt{a}\}\par
\]

Stack size: \texttt{gtrstacksize}\{foo\}

Stack size: 0

Stack size: 1

### 15.2 Push to a Stack

\[
\text{\texttt{gtrstackpush}}\{\texttt{name}\}\{\texttt{content}\}\}
\]

Pushes \textit{content} to a stack \textit{name}. The \textit{content} is expanded during pushing.

\[
\texttt{\def\myx{X}}
\]

\[
\texttt{\gtrnewstack}\{\texttt{foo}\}
\]

\[
\texttt{\gtrstackpush}\{\texttt{foo}\}\{\texttt{a}\}\par
\]

\[
\texttt{\gtrstackpop}\{\texttt{foo}\}
\]

\[
\texttt{X}
\]

299
15.3 Pop from a Stack

\gtrstackpop{\langle name \rangle}

Pops content from a stack \langle name \rangle. The last pushed content is popped first.

\begin{verbatim}
\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}
\end{verbatim}

world hello a is This

\gtrstackpopto{\langle name \rangle}\{\langle macro \rangle\}

Pops content from a stack \langle name \rangle into a \langle macro \rangle.

\begin{verbatim}
\gtrnewstack{foo}
\gtrstackpush{foo}{My}
\gtrstackpush{foo}{test}
\gtrstackpopto{foo}{\myA}
\gtrstackpopto{foo}{\myB}
'\myA' and '\myB'.
\end{verbatim}

'test' and 'My'.

15.4 Peek into a Stack

\gtrstackpeek{\langle name \rangle}

Reads from a stack \langle name \rangle without reducing the stack content.

\begin{verbatim}
\gtrnewstack{foo}
\gtrstackpush{foo}{First entry}
'\gtrstackpeek{foo}', '\gtrstackpeek{foo}';
\gtrstackpush{foo}{Second entry}
'\gtrstackpeek{foo}', '\gtrstackpeek{foo}';
\end{verbatim}

'First entry', 'First entry'; 'Second entry', 'Second entry';

\gtrstackpeekto{\langle name \rangle}\{\langle macro \rangle\}

Peeks content from a stack \langle name \rangle into a \langle macro \rangle.

\begin{verbatim}
\gtrnewstack{foo}
\gtrstackpush{foo}{My}
\gtrstackpush{foo}{test}
\gtrstackpeekto{foo}{\myA}
\gtrstackpeekto{foo}{\myB}
'\myA' and '\myB'.
\end{verbatim}

'test' and 'test'.
15.5 Creating Stack Shortcuts

\gtrmakestack\{\langle\text{name}\rangle\}

Creates a new empty stack \langle\text{name}\rangle and creates new macros \langle\text{name}\rangle\text{size}, \langle\text{name}\rangle\text{push}, \langle\text{name}\rangle\text{popto}, \langle\text{name}\rangle\text{pop}, \langle\text{name}\rangle\text{peekto}, and \langle\text{name}\rangle\text{peek}. These macros serve as shortcuts to the corresponding stack macros from above.

\gtrmakestack\{\text{foo}\}
\foopush\{\text{First}\}
\foopush\{\text{Second}\}
\foopush\{\text{Third}\}
The stack contains \text{\foosize} entries. The last one is \text{\foopeek}.

\foopopto\\myA
The stack contains \text{\foosize} entries after \text{\myA} was removed.

The remaining entries are \text{\foopop} and \text{\foopop}.
Now, the stack contains \text{\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\{\text{foo}\}
\foopush\{\text{Mary}\}\foopush\{\text{had}\}\foopush\{\text{a}\}\foopush\{\text{little}\}\foopush\{\text{lamb}\}
\loop
\ifnum\foosize>0
\foopop,
\repeat
lamb, little, a, had, Mary,
16

Version History

v1.32 (2019/04/08)

- Internal database print command documented: \texttt{\gtrPrintDatabase} \textsuperscript{P.172}
- The templates library \texttt{\templates} used some internal color names of \texttt{tcolorbox}. They are adapted now to the new official names.
- \textit{The tcolorbox package} \textsuperscript{[3]} needs to be version 4.20 (2019/03/02) or newer.

v1.31 (2018/04/17)

- New data keys \texttt{/gtr/database/imageopt} \textsuperscript{P.156} and \texttt{/gtr/database/viewport} \textsuperscript{P.156} accompanied by \texttt{\gtrDBimageopt} to pass options to included images.
- New data access macro \texttt{\gtrincludeDBimage} \textsuperscript{P.188} for including a database image.
- New \texttt{TikZ} options \texttt{/tikz/fill overzoom DBimage} \textsuperscript{P.189} and \texttt{/tikz/fill zoom DBimage} \textsuperscript{P.189} to fill a path with a database image.
- \textit{The tcolorbox package} \textsuperscript{[3]} needs to be version 4.13 (2018/03/22) or newer.

v1.30 (2017/12/08)

- Spanish translation provided by Francisco G. Pérez Sánchez – new value \texttt{spanish} for \texttt{/gtr/language} \textsuperscript{P.233}.
- Swedish translation provided by Per Starbäck – new value \texttt{swedish} for \texttt{/gtr/language} \textsuperscript{P.233}.
- New values for \texttt{/gtr/date format} \textsuperscript{P.175}: \texttt{d/m yyyy}, \texttt{yyyy d/m}.
- Warnings for missing language strings added to \texttt{genealogytree-languages.pdf}
- New options for local edge shifting: \texttt{/gtr/edges shift} \textsuperscript{P.220}, \texttt{/gtr/edges up} \textsuperscript{P.221}, \texttt{/gtr/edges down} \textsuperscript{P.221}, \texttt{/gtr/edges up by} \textsuperscript{P.221}, and \texttt{/gtr/edges down by} \textsuperscript{P.221}.
- New options for global edge shifting: \texttt{/gtr/reset edge level shift} \textsuperscript{P.222}, \texttt{/gtr/switch edge level shift} \textsuperscript{P.222}, and \texttt{/gtr/nullify edge level shift} \textsuperscript{P.223}.
v1.21 (2017/09/15)

- Italian translation provided by Andrea Vaccari – new value `italian` for `/gtr/language`\textsuperscript{P.233}.
- \texttt{\textbackslash gtrsymFloruit}\textsuperscript{P.229} symbol rotated by 36 degrees for an enhanced optical differentiation from \texttt{\textbackslash gtrsymBorn}\textsuperscript{P.227}.
- New `/gtr/template`\textsuperscript{P.255} styles, see Chapter 12 on page 255, namely `database pole reduced` (Section 12.9 on page 261), `database poleportrait` (Section 12.10 on page 262), `database poleportrait reduced` (Section 12.11 on page 264), `database portrait reduced` (Section 12.13 on page 267), `database traditional reduced` (Section 12.15 on page 269), `database sideways reduced` (Section 12.17 on page 272), `database sidewaysportrait` (Section 12.18 on page 273), `database sidewaysportrait reduced` (Section 12.19 on page 274).

v1.20 (2017/07/18)

- Dutch translation provided by Dirk Bosmans – new value `dutch` for `/gtr/language`\textsuperscript{P.233}.
- New event «floruit» with symbol \texttt{\textbackslash gtrsymFloruit}\textsuperscript{P.229} proposed by Mikkel Eide Eriksen. Accompanying keys are `/gtr/database/floruit`\textsuperscript{P.159}, `/gtr/database/floruit+`\textsuperscript{P.159}, `/gtr/database/floruit-`\textsuperscript{P.159}, `/gtr/symlang/Floruit`\textsuperscript{P.232}, and `/gtr/event prefix/floruit`\textsuperscript{P.181}. The standard options for `/gtr/database format`\textsuperscript{P.163} were extended for «floruit».

v1.10 (2017/01/29)

- Danish translation provided by Mikkel Eide Eriksen – new value `danish` for `/gtr/language`\textsuperscript{P.233}.
- French translation provided by Denis Bitouzé – new value `french` for `/gtr/language`\textsuperscript{P.233}.
- Separate document `genealogytree-languages` added to give a short survey of language settings.
- New values for `/gtr/date format`\textsuperscript{P.175}: `typical`, `dd mon.yyyy`, `d mon.yyyy`, `dd/mm yyyy`, `yyyyyymmdd`. The initial setting is changed to be `typical`.
- New data key `/gtr/database/profession`\textsuperscript{P.156} accompanied by \texttt{\textbackslash gtrDBprofession}, \texttt{\textbackslash gtrPrintProfession}\textsuperscript{P.185}, \texttt{\textbackslash gtrprofessiondefined}\textsuperscript{P.185}, and `/gtr/profession code`\textsuperscript{P.185}.
- Option settings for `/gtr/database format`\textsuperscript{P.163} updated to process `/gtr/database/comment`\textsuperscript{P.155} and `/gtr/database/profession`\textsuperscript{P.156}.
- New environment `gtrinfolist`\textsuperscript{P.186} and new corresponding option `/gtr/info separators`\textsuperscript{P.186}.
- Implementation for level sensitive variables changed to support extension packages.
- Implementation for `\gtruselibrary`\textsuperscript{P.13} changed to support third party extension packages.
v1.01 (2016/07/29)

- Family options like `/gtr/pivot shift`\(^\text{P.104}\) are now applicable for `/gtr/options for family`\(^\text{P.102}\) and `/gtrsetoptionsforfamily`\(^\text{P.102}\).
- New option `/gtr/tikz`\(^\text{P.101}\).
- New option `/tikz/genealogytree extra edges scope`\(^\text{P.218}\).
- 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`\(^\text{P.255}\) called `database relationship` (Section 12.20 on page 275).

v0.91 beta (2015/06/22)

- `/gtr/phantom`\(^\text{P.122}\) and `/gtr/phantom*`\(^\text{P.124}\) switch off more settings of `tcolorbox`.
- New options `/gtr/id prefix`\(^\text{P.92}\) and `/gtr/id suffix`\(^\text{P.92}\).
- New `/gtr/template`\(^\text{P.255}\) called `database sideways` (Section 12.16 on page 270).
- New tutorials “Externalization” (Section 2.6 on page 49) and “Conversion” (Section 2.7 on page 52).
- Settings for `/gtr/date range full`\(^\text{P.178}\) and `/gtr/date range separator`\(^\text{P.178}\) changed.
- `/gtr/date format`\(^\text{P.175}\) 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.


above value, 110, 111
AD value, 161
add child key, 209
add parent key, 210
adjust node key, 111
adjust position key, 110
after parser key, 109
after tree key, 113
ahntenafel 3 value, 276
ahntenafel 4 value, 278
ahntenafel 5 value, 280
all key, 13
all value, 177
all but AD value, 161, 177
anchoring key, 202
background key, 202
baptism key, 158, 180
baptism+ key, 158
baptism- key, 158
Baptized key, 232
BC value, 161
below value, 110, 111
birth key, 158, 180
birth+ key, 158
birth- key, 158
Born key, 232
Borntoutofwedlock key, 232
both value, 95, 213, 249
box key, 96
box clear key, 96
burial key, 160, 181
burial+ key, 160
burial- key, 160
Buried key, 232
c grammer, 17, 19, 21, 55, 59, 60, 62, 71, 75, 76, 115, 268, 286
c value, 161
cAD value, 161
cBC value, 161
calendar print key, 177
calendar text for key, 177
center value, 202
child value, 95, 213
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, 282
gtr_Blue_2, 282
gtr_Blue_3, 282
gtr_Blue_4, 282
gtr_Blue_5, 282
gtr_Blue_6, 282
gtr_Bright_Red, 282
gtr_Peach_Back, 282
gtr_Peach_Frame, 282
gtr_Yellow_Back, 282
gtr_Yellow_Frame, 282
gtr_symbol, 225, 226
comment key, 155
comment code key, 184
content interpreter key, 145
content interpreter code key, 146
content interpreter content key, 146
content interpreter id and content key, 149
contour key, 249
cremated key, 181
cremated value, 160
custom key, 199
d M yyyy value, 175
d mon yyyy value, 175
d mon.yyyy value, 175, 304
d month yyyy value, 175
d-M-yyyy value, 176
d-m-yyyy value, 175
d-mon-yyyy value, 176
d-month-yyyy value, 176
d.M.yyyy value, 175
d.m.yyyy value, 175
d.mon.yyyy value, 175
d.month yyyy value, 175
d.month yyyy value, 175
d/m yyyy value, 175, 303
d/M/yyyy value, 175
d/m/yyy value, 175
d/mon/yyyy value, 175
d/month/yyyy value, 175
danish value, 233, 304
database value, 35, 128, 151, 259, 262, 265, 268, 270, 273, 275, 276, 278, 280
database content interpreter key, 148
database format key, 163
database pole value, 259, 261, 262
database pole reduced value, 261, 304
database poleportrait value, 262, 264, 304
database poleportrait reduced value, 264, 304
template key, 255
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, 257
tiny circles value, 257
tree offset key, 109
turn key, 98
typical value, 175, 304

union grammar, 26, 27, 55, 62, 65, 67, 104, 213, 219, 247, 253, 286–288, 290

up value, 78
upsidedown value, 98
use family database key, 207
uuid key, 156

Values
above, 110, 111
AD, 161
ahnentafel 3, 276
ahnentafel 4, 278
ahnentafel 5, 280
all, 177
all but AD, 161, 177
BC, 161
below, 110, 111
both, 95, 213, 249
c, 161
catAD, 161
catBC, 161
center, 202
child, 95, 213
cremated, 160
d M yyyy, 175
d mon yyyy, 175
d mon.yyyy, 175, 304
d month yyyy, 175
d–M-yyyy, 176
d–m-yyyy, 175
d–mon–yyyy, 176
d–month–yyyy, 176
d.M.yyy, 175
d.m.yyy, 175
d.m mon yyyy, 175
d.month yyyy, 175
d/m yyyy, 175, 303
d/M/yyyy, 175
d/m/yyyy, 175
d/month/yyyy, 175
d–mon/yyyy, 175
d/month/yyyy, 175
danish, 233, 304
database, 35, 128, 151, 259, 262, 265, 268, 270, 273, 275, 276, 278, 280
database pole, 259, 261, 262
database pole reduced, 261, 304
database poleportrait, 262, 264, 304
database poleportrait reduced, 264, 304
database portrait, 265, 267
database portrait reduced, 267, 304
database relationship, 275, 305
database sideways, 270, 272, 273, 305
database sideways reduced, 272, 304
database sidewaysportrait, 304
database sidewaysportrait reduced, 304
database sidewaysportrait, 273, 274
database sidewaysportrait reduced, 274
database traditional, 268, 269
database traditional reduced, 269, 304
date, 181
dd mm yyyy, 175
dd mon yyyy, 175
dd mon.yyyy, 175, 304
dd–mm–yyy, 175
dd–mon–yyy, 176
dd–mon–yyyy, 175
dd.mm.yyyy, 175
dd/month/yyyy, 175
dd/month/yyy, 175
dd/month/yyyy, 175
dd/month/yyyy, 175
dd/mon MMM yyyy, 175
dd/mon/yyyy, 175
dd/mon.yyyy, 175
dd/mon.yyyy, 175
dd/mon/y, 175
dd/mon/yyy, 175
dd/mm/yyyy, 175
dd/mm/yyyy, 175
dd/mm/yyyy, 175
dd/mm/yyy, 175
dd/mon/yyyy, 175
dd/mon/yy, 175

died, 158
directory, 258
down, 78, 199, 268, 275
dutch, 233, 304
east, 249
empty, 163
english, 231, 233
error, 157
female, 155, 187
fit, 128, 129, 256
formal graph, 255
french, 233, 304
full, 163, 173
full marriage above, 163, 259, 262
full marriage below, 163
full no marriage, 163
german, 232, 233
german-austrian, 233
german-german, 233
GR, 161
ignore, 157
italian, 233, 304
JU, 161
killed, 159
left, 78, 98, 110, 111, 270, 273, 276, 278, 280
m–d–yyyy, 176
m.d.yyyy, 176
m/d/yyyy, 176
male, 155, 187