The gtrlib.largetrees package*

Richard Grewe
r-g+tex@posteo.net

November 2, 2018

1 Introduction

The main goal of the gtrlib.largetrees package is to offer additional database fields and formats for the genealogytree package, particularly for typesetting large trees. This package is the side product of typesetting an actual genealogy tree with almost 400 persons on 16 levels and on a DIN-A0 poster. The package provides the following extensions:

database fields:

- spouse, spousebirth, spousedeath
- children

formats:

- Person+Marriage+Children+Profession

node processors:

- sparse node processor

In the following, each of the extensions is described and illustrated individually.

To use the package, use

```
\usepackage{genealogytree}
\gtruselibrary{largetrees}
```

or

```
\usepackage{gtrlib.largetrees}
```

Note that the following approach does *not* work:

- `\usepackage[largetrees]{genealogytree}`

*This document corresponds to gtrlib.largetrees v1.2b, dated 2018/11/02. The package is available online at https://github.com/Ri-Ga/gtrlib.largetrees.

2 Database Fields

2.1 Spouse

Sometimes, basic information about the spouse of a person is nice to have in a genealogy tree but a separate node in the tree would occupy too much space. For instance, if you have a person in the main line of ancestors, then you might also want to show a node for the sister of that person. However, a separate node for her spouse might be too much. In such a case, the database fields `spouse`, `spousebirth`, and `spousedeath` can come handy.

The `spouse` field can be used for the specification of the name of the spouse. The `spousebirth` and `spousedeath` fields can be used for specifying the birth and death events of the spouse. For these events, the values are expected to be of the same format as other events, such as the `birth` and `death` fields. In particular this means that date and location can be specified. The following example shows the usage of the fields at the example of Gauss' first wife:

```
\begin{genealogypicture}[processing=database,
  database format=person marriage children profession]
sandclock{ child{
  g[id=GauxCarl1777]{male,
    name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
    birth={1777-04-30}{Braunschweig (Niedersachsen)},
    death={1855-02-23}{G\"ottingen (Niedersachsen)},
    marriage={1805-10-09}{Braunschweig (Niedersachsen)},
    spouse={\pref{Johanna} Elisabeth Rosina \surn{Osthoff}},
    spousebirth={1780-05-08}{Braunschweig (Niedersachsen)},
    spousedeath={1809-10-11}{G\"ottingen (Niedersachsen)},
  }}}
\end{genealogypicture}
```

Johann *Carl Friedrich* Gauss
★ April 30, 1777 in Braunschweig (Niedersachsen)
† February 23, 1855 in Göttingen (Niedersachsen)

⚭ October 9, 1805 in Braunschweig (Niedersachsen) with *Johanna* Elisabeth Rosina Osthoff (★ May 8, 1780, † October 11, 1809)

Note that currently only information about a single spouse can be specified through the fields provided by this package.

2.2 Children

When one wants to have information about children in a genealogy tree but cannot spend the space for separate nodes, then one can use the `children`, `children-`, and `children+` keys provided by this package. The following describes the individual keys and provides an illustration, again using data from Gauss' first marriage [Stu16, Section 2.3.5].

- Use `children+={some infos}` for providing arbitrarily formatted information about children.

```
\begin{genealogypicture}[processing=database,
  database format=person marriage children profession]
sandclock{ child{
  g[id=GauxCarl1777]{male,
    name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
    birth={1777-04-30}{Braunschweig (Niedersachsen)},
    death={1855-02-23}{G\"ottingen (Niedersachsen)},
    children+={Carl (\gtrsymBorn\,1806),
      Wilhelmina (\gtrsymBorn\,1808),
      Ludwig (\gtrsymBorn\,1809)},
  }}}
\end{genealogypicture}
```

Johann *Carl Friedrich* Gauss
★ April 30, 1777 in Braunschweig (Niedersachsen)
† February 23, 1855 in Göttingen (Niedersachsen)
children: Carl (★1806), Wilhelmina (★1808), Ludwig (★1809)

- Use `children={D}{S}` to specify that the person has `D` daughters and `S` sons.

```
\begin{genealogypicture}[processing=database,
  database format=person marriage children profession]
sandclock{ child{
  g[id=GauxCarl1777]{male,
    name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
    birth={1777-04-30}{Braunschweig (Niedersachsen)},
    death={1855-02-23}{G\"ottingen (Niedersachsen)},
    children={1}{2},
  }}}
\end{genealogypicture}
```

Johann *Carl Friedrich* Gauss
★ April 30, 1777 in Braunschweig (Niedersachsen)
† February 23, 1855 in Göttingen (Niedersachsen)
children: 1♀ 2♂

- Use `children-=N` for providing only the overall number of children of the person.

```
\gtrset{language=german}
\begin{genealogypicture}[processing=database,
  database format=person marriage children profession]
sandclock{ child{
  g[id=GauxCarl1777]{male,
    name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
    birth={1777-04-30}{Braunschweig (Niedersachsen)},
    death={1855-02-23}{G\"ottingen (Niedersachsen)},
    children-={3},
  }}}
\end{genealogypicture}
```

Johann *Carl Friedrich* Gauss
★ April 30, 1777 in Braunschweig (Niedersachsen)
† February 23, 1855 in Göttingen (Niedersachsen)
Kinder: 3

Note that we show here also that the text "children" changes with the selected language.

3 Database Formats

This package extends genealogytree's list of formats for displaying database entries [Stu16, Section 7.3.5].

3.1 Person+Marriage+Children+Profession

This format is close to the format "medium marriage below", which is defined by the genealogytree package. The format can be selected by the `database format` key, as shown in the previous examples. The format displays

1. name, birth, and death of the person
2. a comment, if provided
3. the marriage event and the spouse
4. the children
5. the profession

The following long example shows a full example, based on combined data from the previous examples.

```
\begin{genealogypicture}[processing=database,
  database format=person marriage children profession]
sandclock{ child{
  g[id=GauxCarl1777]{male,
    name={Johann \pref{Carl Friedrich} \surn{Gau\ss{}}},
    birth={1777-04-30}{Braunschweig (Niedersachsen)},
    death={1855-02-23}{G\"ottingen (Niedersachsen)},
    marriage={1805-10-09}{Braunschweig (Niedersachsen)},
    spouse={\pref{Johanna} Elisabeth Rosina \surn{Osthoff}},
    spousebirth={1780-05-08}{Braunschweig (Niedersachsen)},
    spousedeath={1809-10-11}{G\"ottingen (Niedersachsen)},
    children={1}{2},
    profession={Mathematiker, Astronom, Geod\"at und Physiker},
  }}}
\end{genealogypicture}
```

4 Node Processors

4.1 Sparse node processor

The sparse node processor serves the purpose of saving space by rotating nodes for which only few database fields are set. This follows the consideration that horizontally displayed nodes have the advantage of being easily readable, but require a rather wide display for a nice appearance. In a large genealogy tree with many nodes on a level, space can be saved by rotating those nodes for which only few database fields are set. The sparse node processor allows one to enable rotation dependent on how much information is available for a node, rather than merely on the level of the node or whether the node is a leaf or inner node. The sparse node processor inherits from the "fit" processor defined by the genealogytree package [Stu16, Section 6.2.1].

The sparse node processor can be configured by several options. Some of these options can be set globally, some also for individual levels. The following options are defined:

`sparse display`**:**
enables the node processor.

`sparse node size from=`⟨*min*⟩ `to` ⟨*max*⟩**:**
sets minimum and maximum width of sparse nodes.

`sparse max fields=`⟨*N*⟩**:**
defines the maximum number ⟨*N*⟩ of database fields of a person with which

the node is still considered sparse and, therefore, displayed specially (rotated).

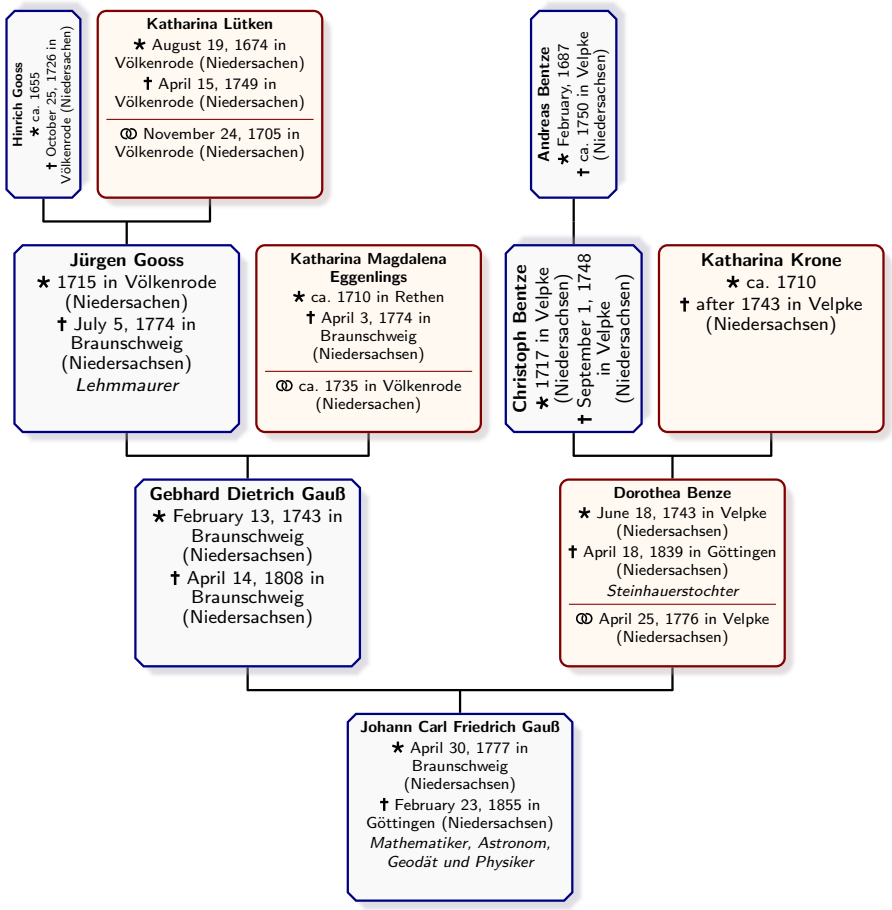
`sparse inner from level=`$\langle N \rangle$`:`
: specifies that inner nodes shall be displayed specially only if they are above or on level $\langle N \rangle$.

`sparse leaf from level=`$\langle N \rangle$`:`
: specifies that leaf nodes shall be displayed specially only if they are above or on level $\langle N \rangle$.

`sparse inner never:`
: specifies that inner nodes shall never be displayed specially.

The following example shows the use and the effect of the options in an example, again based on the tree for Gauss. Concretely, the example demonstrates the following settings:

- a node is considered sparse if at most 2 fields are defined: Hinrich Gooss (2 fields) is rotated, Katharina Ltken (3 fields) is not rotated;
- inner nodes get special display earliest from level 2: Christoph Bentze (2 fields and on level 2) is rotated, Gebhard Dietrich Gauß (sparse due to only 2 fields, but on level 1) is not rotated);
- leaf nodes get special display earliest from level 3: Katharina Krone (2 fields but on level 2) is not rotated, Andreas Bentze (2 fields and on level 3) is rotated;
- sparse nodes on levels 3 and 2 have different width settings.




```
\usepackage[templates]{genealogytree}
\begin{genealogypicture}[template=database pole,
    database format=person marriage children profession,
    level size=2.5cm, node size=3cm,
    level 3/.style={sparse node size from=0.4cm to 1.2cm},
    sparse node size from=1cm to 2cm,
    sparse max fields=2,
    sparse inner from level=2,
    sparse leaf from level=3,
    sparse display,
    options for node={GauxCarl1777}{pivot},
  ]
  input{example.gauss.graph}
\end{genealogypicture}
```

Note that sparse node display currently is only possible for the database formats defined in Section 3. For enabling it in combination with other database formats, the `\gtrltDeclareFieldCount{`⟨*db-format*⟩`}{`⟨*macro*⟩`}` has to be used (see the use of this macro in the implementation of the database formats provided by this package).

References

[Stu16] Thomas F. Sturm. *genealogytree – Manual for version 1.01*. July 29, 2016. URL: http://mirrors.ctan.org/macros/latex/contrib/genealogytree/genealogytree.pdf (visited on 12/23/2016).

A Implementation

A.1 Wrapper Package

The package has the following dependencies:

```
\RequirePackage{genealogytree}[2017/01/29 version 1.10]
\gtruselibrary{largetrees}
```

The remaining code provides the largetrees library for the genealogytree package.

A.2 Dependencies

The library has the following dependencies:

```
\RequirePackage{etoolbox}
```

A.3 Database Fields

A.3.1 Spouse

This adds three new fields for providing information about the spouse of a person, within the same node. The fields are for specifying the name of the spouse as well as the birth date and the day of death.

`\gtrDBspouse` The `\gtrDBspouse` macro holds what is provided for the `spouse` key of a database entry (this macro is defined by `\gtr@db@new@store`).

```
\gtr@db@new@store{spouse}
\gtr@db@new@event@store{spousebirth}
\gtr@db@new@event@store{spousedeath}
```

`\gtrifspousedefined` The `\gtrifspousedefined{`⟨*true*⟩`}{`⟨*false*⟩`}` macro expands to ⟨*true*⟩ if the spouse of a database entry is defined, and expands to ⟨*false*⟩ otherwise. The distinction is made solely on the `spouse` field – i.e., `spousebirth` and `spousedeath` are ignored. This macro can be used, for instance, by database formats (see Appendix A.5).

```
\def\gtrifspousedefined#1#2{\ifdefvoid{\gtrDBspouse}{#2}{#1}}
```

`\gtrPrintSpouse` `\gtrPrintSpouseDetails` The `\gtrPrintSpouseDetails` macro prints the name of the spouse as well as the birth and death of the spouse, if provided by the database entry. The `\gtrPrintSpouse` macro only prints the name.

```
\newcommand\gtrPrintSpouse{\gtrDBspouse}
\newcommand\gtrPrintSpouseDetails{%
  \gtrPrintSpouse
  \begin{gtrprintlist}{\unskip\ (}{\unskip,\ }{\unskip)}{\unskip}%
    \gtrifeventdefined{spousebirth}%
      {\gtrlistseparator\gtrPrintEventPrefix{birth}\,%
       \gtrPrintDate{spousebirth}}%
      {}%
    \gtrifeventdefined{spousedeath}%
```

```
      {\gtrlistseparator\gtrPrintEventPrefix{death}\,%
       \gtrPrintDate{spousedeath}}%
      {}%
  \end{gtrprintlist}}
```

A.3.2 Children

This adds a field for providing information about the children of a person, within the same node.

`\gtrDBchildren` `\gtrDBdaughters` `\gtrDBsons` If detailed information about children is provided through the `children+` key, then the `\gtrDBchildren` holds this detailed information. Otherwise, i.e., if information about children is provided through the `children-` or `children` key, then `\gtrDBchildren` holds the number of children of the person. The `\gtrDBdaughters` and `\gtrDBsons` macros store the number of daughters and, respectively, sons of the person.

```
\gtrset{%
  database/children/.code n args={2}{%
    \numdef\gtrDBchildren{#1+#2}%
    \def\gtrDBdaughters{#1}%
    \def\gtrDBsons{#2}},%
  database/children-/.store in=\gtrDBchildren,
  database/children+/.store in=\gtrDBchildren,
}
```

`\gtrifchildrendefined` The `\gtrifchildrendefined{`⟨*true*⟩`}{`⟨*false*⟩`}` macro expands to ⟨*true*⟩ if information about children of a database entry is defined, and expands to ⟨*false*⟩ otherwise. This macro can be used, for instance, by database formats (see Appendix A.5).

```
\def\gtrifchildrendefined#1#2{\ifdefvoid{\gtrDBchildren}{#2}{#1}}
```

`\gtrPrintChildren` The `\gtrPrintChildren` macro prints information about the children of a person.

```
\def\gtrPrintChildren{%
  \gtrTranslatedChildren:
```

If `\gtrDBdaughters` is defined, then information about children must have been provided by the `children` key – that is, numbers of daughters and sons have been provided. Otherwise information about children was provided though the `children+` (arbitrarily formatted information) or `children-` (number of children in total) keys, in which cases this information is printed as is.

```
    \ifdefvoid{\gtrDBdaughters}%
      {\gtrDBchildren}%
      {\gtrDBdaughters\kern0.5pt\gtrsymFemale~%
       \gtrDBsons\kern0.5pt\gtrsymMale}}
```

A.4 Node Processors

A.4.1 Sparse Node Processor

The following defines the keys that can be used for enabling and configuring sparse node display.

```
\gtrset{%
```

The `sparse display` key enables the node processor.

```
  sparse display/.style={node processor=\gtrltSparseNodeProcessor},
```

The `sparse node size from` sets minimum and maximum width of sparse nodes.

```
  sparse node size from/.code args={#1 to #2}{%
    \dimdef\gtr@@kv@@gtrlt@sparsenode@minsize{#1}%
    \dimdef\gtr@@kv@@gtrlt@sparsenode@maxsize{#2}},
```

The `sparse max fields` defines the maximum number of fields of a person with which the node is still considered sparse.

```
  sparse max fields/.code={%
    \gdef\gtrlt@sparse@maxfields{#1}},%
```

The following three options specify when sparse nodes are displayed specially. The `sparse inner from level` option specifies that inner nodes shall be displayed specially only if they are above or equal a particular level.

```
  sparse inner from level/.code={\def\gtrlt@sparseinner@level{#1}},
```

The `sparse leaf from level` option specifies that leaf nodes shall be displayed specially only if they are above or equal a particular level.

```
  sparse leaf from level/.code={\def\gtrlt@sparseleaf@level{#1}},
```

The `sparse inner never` option specifies that inner nodes shall never be displayed specially.

```
  sparse inner never/.code={\undef\gtrlt@sparseinner@level},
}
```

The following registers the level-local macros used by the above options.

```
\gtr@proc@register@level@local{\gtrlt@sparsenode@minsize}
\gtr@proc@register@level@local{\gtrlt@sparsenode@maxsize}
```

`\gtrltSparseNodeProcessor` The `\gtrltSparseNodeProcessor` is a node processor that inherits from the "fit" node processor, which it augments by means for displaying sparse nodes rotated.

```
\newcommand\gtrltSparseNodeProcessor{%
  \gtrltIfSparseEnabled{\gtrltIfSparse{%
    \let\gtrNodeMinWidth=\gtrlt@sparsenode@minsize%
    \let\gtrNodeMaxWidth=\gtrlt@sparsenode@maxsize%
    \gtrkeysappto\gtrNodeBoxOptions{%
      rotate=90,gtrNodeDimensionsLandscape,%
    }%
  }{}}{}%
  \csuse{gtr@boxcontent@fit}}
```

\gtrltIfSparseEnabled Sparse display is enabled for leaf parent and leaf nodes and (if sparse display is desired for inner nodes) for inner nodes above the configured level. The `\gtrltIfSparseEnabled{`⟨*true*⟩`}{`⟨*false*⟩`}` macro checks the current node for whether these properties are satisfied and expands to ⟨*true*⟩ if this is the case and ⟨*false*⟩ otherwise.

```
\newcommand\gtrltIfSparseEnabled{%
  \gtrifleaf
    {\ifdef\gtrlt@sparseleaf@level%
```

For leaf nodes, if a mimimum level is specified (via `sparse leaf from level`), then sparse display is enabled if the node's level is greater or equal than the specified minimum:

```
      {\ifnumgreater{1+\gtrnodelevel}{\gtrlt@sparseleaf@level}}%
```

If a mimimum level for leaf nodes is not specified, then sparse display is enabled no matter the node's level:

```
      {\@firstoftwo}}%
    {\ifdef\gtrlt@sparseinner@level%
```

For inner nodes, if a mimimum level is specified (via `sparse inner from level`), then sparse display is enabled if the node's level is greater or equal than the specified minimum:

```
      {\ifnumgreater{1+\gtrnodelevel}{\gtrlt@sparseinner@level}}%
```

If a mimimum level for inner nodes is not specified, then sparse display is disabled no matter the node's level:

```
      {\@secondoftwo}}}
```

\gtrltFieldCount
\gtrltIfSparse The `\gtrltIfSparse{`⟨*true*⟩`}{`⟨*false*⟩`}` macro uses database format specific code to count the number of fields available for a node and expands to ⟨*true*⟩ if this number is less than or equal the maximum number of fields for sparse display (specified by `sparse max fields`). Otherwise, the macro expands to ⟨*false*⟩. `\gtrltFieldCount` is a counter used internally by the macro.

```
\newcount\gtrltFieldCount
\newcommand\gtrltIfSparse{%
  \gtrltFieldCount=0\relax
  \gtrltCountDatabaseFields
```

Decrement the counter for using `\ifnumless` comparison below rather than "less-or-equal" (for which there is no conditional):

```
  \advance\gtrltFieldCount by-1\relax%
  \ifnumless{\the\gtrltFieldCount}{\gtrlt@sparse@maxfields}}
```

\gtrltDeclareFieldCount The `\gtrltDeclareFieldCount{`⟨*db-format*⟩`}{`⟨*macro*⟩`}` registers, for the given database format ⟨*db-format*⟩, macro code `{`⟨*macro*⟩`}` for counting the number of displayed database fields. The ⟨*macro*⟩ must ensure that, after expansion, the counter `\gtrltFieldCount` holds the number of fields.

```
\newcommand\gtrltDeclareFieldCount[2]{%
  \gtrset{database format/#1/.append code={%
    \def\gtrltCountDatabaseFields{#2}}}}
```

\gtrltFieldCountByConditionals

The \gtrltFieldCountByConditionals{⟨*cond-list*⟩} macro counts the number of satisfied conditionals in the comma-separated list ⟨*cond-list*⟩ of conditionals. This number is returned in the \gtrltFieldCount counter.

```
\newcommand\gtrltFieldCountByConditionals[1]{%
```

The \do{⟨*conditional*⟩} macro increases \gtrltFieldCount if ⟨*conditional*⟩ is true and otherwise does nothing. For this, the ⟨*conditional*⟩ must be a macro with two argument, expanding to the first argument if the conditional is true, and to the second argument otherwise.

```
  \def\do##1{%
    ##1{\advance\gtrltFieldCount by1\relax}{}}%
  \gtrltFieldCount=0\relax
  \docsvlist{#1}}
```

A.5 Database Formats

A.5.1 Person+Marriage+Children+Profession

The following code defines the format "person marriage children profession".

```
\gtrDeclareDatabaseFormat{person marriage children profession}{}{%
  \gtrPrintName%
  \begin{gtreventlist}%
    \gtr@list@event@birth%
    \gtr@list@event@death%
  \end{gtreventlist}%
  \gtrifcommentdefined{\gtrPrintComment}{}%
  \gtr@ifmarriagedefined{\tcbline
    \begin{gtreventlist}
       \gtr@list@event@marriage
       \gtrifspousedefined{
         \gtrTranslatedWith\ \gtrPrintSpouseDetails}{}%
       \gtrifchildrendefined{, \gtrPrintChildren}{}%
    \end{gtreventlist}}{%
    \gtrifchildrendefined{\tcbline\gtrPrintChildren}{}}%
  \gtrifprofessiondefined{\tcbline\gtrPrintProfession}{}%
}
```

Specify how the number of displayed database fields of a node are counted:

```
\gtrltDeclareFieldCount{person marriage children profession}{%
  \gtrltFieldCountByConditionals{%
    \gtrifeventdefined{birth},
    \gtrifeventdefined{death},
    \gtrifeventdefined{marriage},
    \gtrifcommentdefined,
    \gtrifspousedefined,
    \gtrifchildrendefined,
    \gtrifprofessiondefined,
  }}
```

A.6 Translations

For text to be displayed in nodes (for instance because there is no established symbol for the respective), we provide some additional small translation facility. This feature is complementary to the symbols facility provided by the genealogytree package.

`\gtrlt@texttranslation@add` The `\gtrlt@texttranslation@add{`⟨*identifier*⟩`}` adds the ⟨*identifier*⟩ to the list of understood translatables and provides a macro `\gtrTranslated`⟨*identifier*⟩ through which the translation for the chosen language can be accessed.

```
\newcommand\gtrlt@texttranslation@add[1]{%
  \begingroup\edef\x{\endgroup\noexpand\gtrset{%
    textlang/#1/.store in=\expandonce{%
      \csname gtrTranslated#1\endcsname}}}\x}
```

The following are the identifiers for which there are translations.

```
\gtrlt@texttranslation@add{Children}
\gtrlt@texttranslation@add{With}
```

The following are the actual translations used by this library. We encapsulate the code in an `\AtBeginDocument` such that the `append code` portion is ensured to be run after genealogytree set its language code. This allows the user of the library to use `\gtrset{language=XXX}` in the preamble as well as in the document without any changes to the normal use in genealogytree.

```
\AtBeginDocument{
  \gtrset{language@/english/.append code={%
    \gtrset{textlang/.cd,
      Children=children,
      With=with,
    }
  }}
  \gtrset{language@/german/.append code={%
    \gtrset{textlang/.cd,
      Children=Kinder,
      With=mit,
    }
  }}
  \gtrset{language=\gtrlanguagename}}
```

Change History

Index

I

K

L

N

R

T

U

X