# The `genealogytree` package

Manual for version 0.10 (2015/01/12)

Thomas F. Sturm[1]

**Abstract**

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

The current version is an *alpha* version of the package. It consists of a genealogy tree parser and a debugger for the parser. This can be used to check a manually or automatically generated tree source code to be well-formed. Also, the debugger gives a formal structured view of the given data. Note that the targeted visual diagram is not implemented yet.

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

# Contents

# Chapter 1

# Graph Grammar

## 1.1 Graph

The root of a parsable graph is one of the following:

- a `parent` (for ancestor graphs), see Section 1.2,
- a `child` (for descendant graphs), see Section 1.3 on page 7,
- a `sandclock` (for mixed ancestor/descendant graphs), see Section 1.5 on page 9.

## 1.2 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**

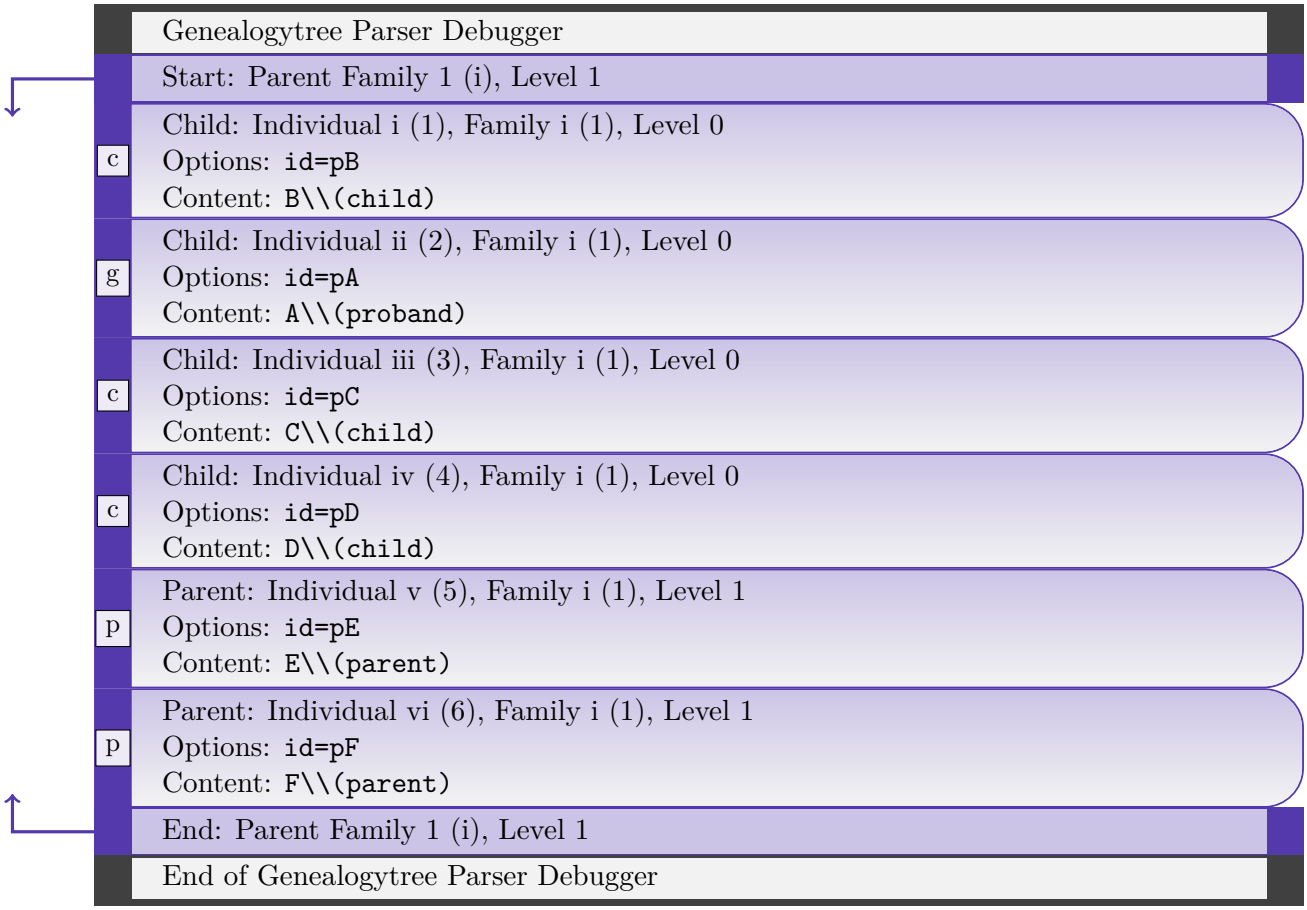
| | |
|---|---|
| `parent[`⟨*subtree options*⟩`]{` | |
| `  g[`⟨*node options*⟩`]{`⟨*node content*⟩`}` | *mandatory; exactly once* |
| `  c[`⟨*node options*⟩`]{`⟨*node content*⟩`}` | *optional; zero or many times* |
| `  p[`⟨*node options*⟩`]{`⟨*node content*⟩`}` | *optional; zero or many times* |
|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

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

**File «examples/parent_subgraph.graph»**

```
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)}%
}
```

```
\gtrparserdebuginput{examples/parent_subgraph.graph}
```



## 1.3 Subgraph 'child'

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

**Syntax for a 'child' subgraph**

```
child[⟨subtree 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[⟨subtree options⟩]{⟨subtree content⟩}      optional; zero or many times
  union[⟨subtree options⟩]{⟨subtree content⟩}      optional; zero or many times
  input{⟨file name⟩}                              optional; zero or many times
}
```

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

**File «examples/child_subgraph.graph»**

```
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)}%
}
```

```
\gtrparserdebuginput{examples/child_subgraph.graph}
```

Genealogytree Parser Debugger

Start: Child Family 1 (i), Level 0

g
Parent: Individual i (1), Family i (1), Level 0
Options: `id=pA`
Content: `A\\(proband)`

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

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

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

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

End: Child Family 1 (i), Level 0

End of Genealogytree Parser Debugger

## 1.4 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[⟨subtree options⟩]{
  c[⟨node options⟩]{⟨node content⟩}          optional; zero or many times
  p[⟨node options⟩]{⟨node content⟩}          optional; zero or many times
  child[⟨subtree options⟩]{⟨subtree content⟩} optional; zero or many times
  input{⟨file name⟩}                         optional; zero or many times
}
```

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

**File «examples/union_subgraph.graph»**

```
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)}%
  }
}
```

```
\gtrparserdebuginput{examples/union_subgraph.graph}
```

Genealogytree Parser Debugger

Start: Child Family 1 (i), Level 0

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

g
Parent: Individual ii (2), Family i (1), Level 0
Options: `id=pA`
Content: `A\\(proband)`

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

Start: Union Family 2 (ii), Level 0

p
Parent: Individual iv (4), Family ii (2), Level 0
Options: `id=pD`
Content: `D\\(parent)`

c
Child: Individual v (5), Family ii (2), Level -1
Options: `id=pE`
Content: `E\\(child)`

End: Union Family 2 (ii), Level 0

End: Child Family 1 (i), Level 0

End of Genealogytree Parser Debugger

## 1.5 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.

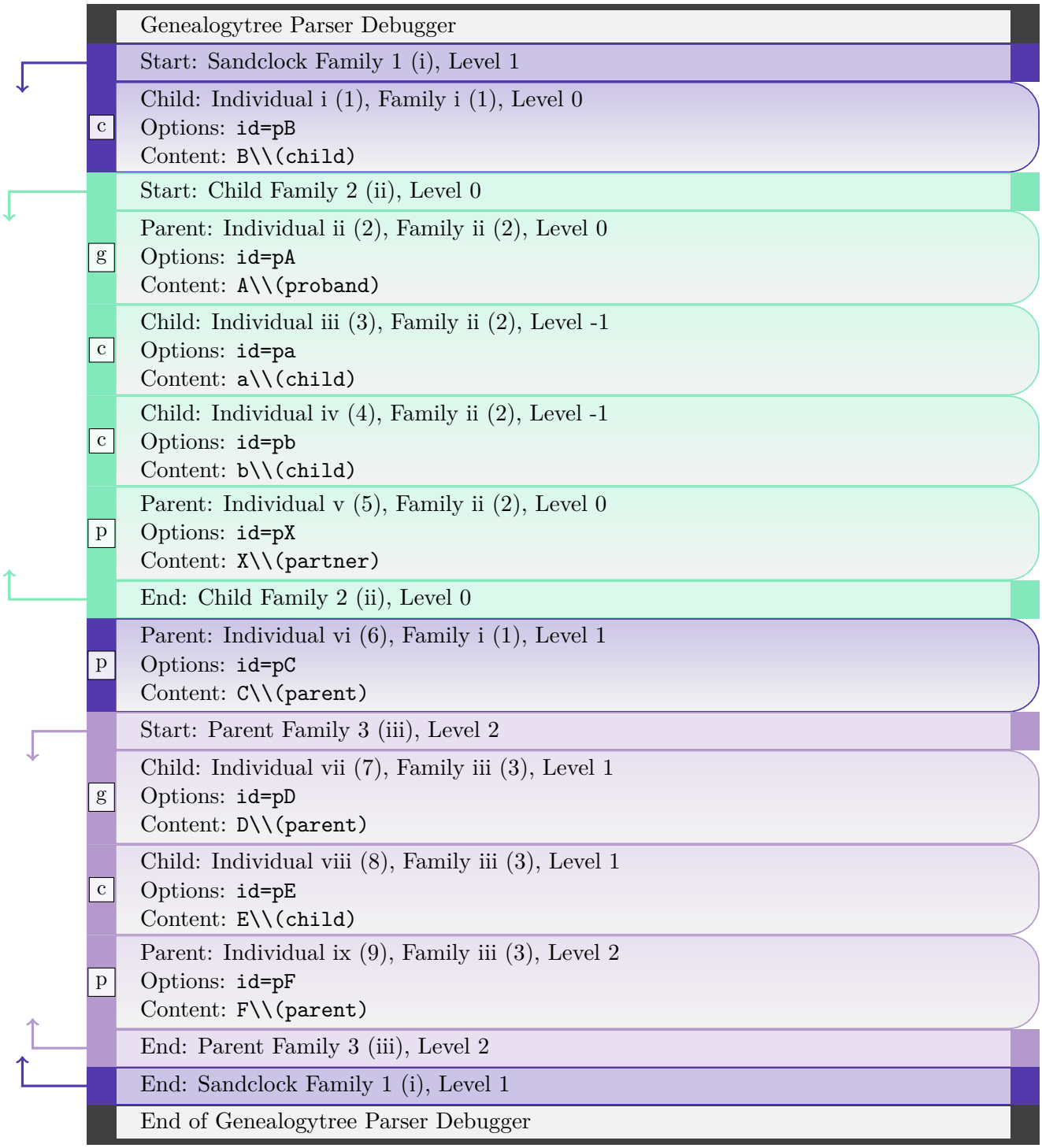
**Syntax for a 'sandclock' subgraph**

```
sandclock[⟨subtree options⟩]{
  c[⟨node options⟩]{⟨node content⟩}          optional; zero or many times
  p[⟨node options⟩]{⟨node content⟩}          optional; zero or many times
  child[⟨subtree options⟩]{⟨subtree content⟩}  mandatory; one or many times
  parent[⟨subtree options⟩]{⟨subtree content⟩}  optional; zero or many times
  input{⟨file name⟩}                         optional; zero or many times
}
'c', 'p', 'child', 'parent', 'input' may appear in arbitrary order.
```

**File «examples/sandclock_subgraph.graph»**

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

```
\gtrparserdebuginput{examples/sandclock_subgraph.graph}
```



## 1.6 Data 'input'

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

**Syntax for data 'input'**

`input{`⟨*file name*⟩`}`

## 1.7 Node 'g'

The `g` (genealogical) 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*⟩`}`

## 1.8 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*⟩`}`

## 1.9 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*⟩`}`

# Chapter 2

# Debugging

## 2.1 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 1. It is not checked, if all mandatory graph elements are present or if too many elements are given.

Also, the debugger gives a formal structured view of the given data which is useful to search for input errors if the graphical representation fails.

`\gtrparserdebug[`⟨*options*⟩`]{`⟨*graph content*⟩`}`

Parses the given ⟨*graph content*⟩. If the content is well-formed, a structured list of the given data is produced. The families are automatically colored in the list. Any ⟨*options*⟩ are checked by setting them and they are logged in the produced list.

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

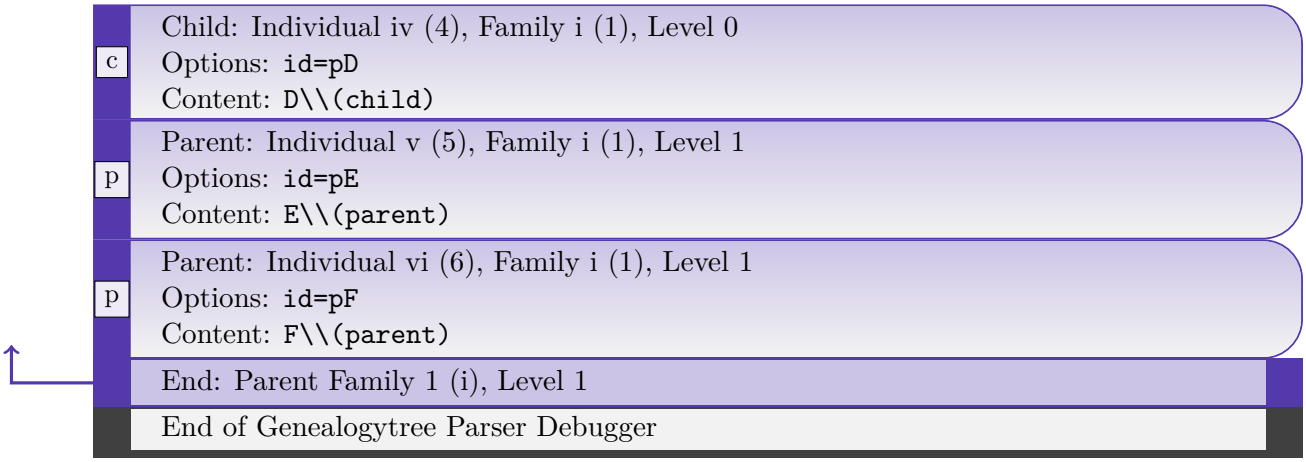
Genealogytree Parser Debugger

Start: Parent Family 1 (i), Level 1

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

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

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


c
Child: Individual iv (4), Family i (1), Level 0
Options: id=pD
Content: D\\(child)
p
Parent: Individual v (5), Family i (1), Level 1
Options: id=pE
Content: E\\(parent)
p
Parent: Individual vi (6), Family i (1), Level 1
Options: id=pF
Content: F\\(parent)
End: Parent Family 1 (i), Level 1
End of Genealogytree Parser Debugger

`\gtrparserdebuginput[⟨options⟩]{⟨file name⟩}`

Loads the file denoted by ⟨*file name*⟩ and parses its content. If the content is well-formed, a structured list of the given data is produced. The families are automatically colored in the list. Any ⟨*options*⟩ are checked by setting them and they are logged in the produced list.

**File «examples/smithdoe.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}
  }
}
```

```
\gtrparserdebuginput{examples/smithdoe.graph}
```

Genealogytree Parser Debugger

Start: Parent Family 1 (i), Level 1
Options: `id=SmithDoe`

g — Child: Individual i (1), Family i (1), Level 0
Options: `id=Arth2008,male`
Content: `Arthur\\\gtrsymborn ~2008`

c — Child: Individual ii (2), Family i (1), Level 0
Options: `id=Bert2010,female`
Content: `Berta\\\gtrsymborn ~2010`

c — Child: Individual iii (3), Family i (1), Level 0
Options: `id=Char2014,male`
Content: `Charles\\\gtrsymborn ~2014`

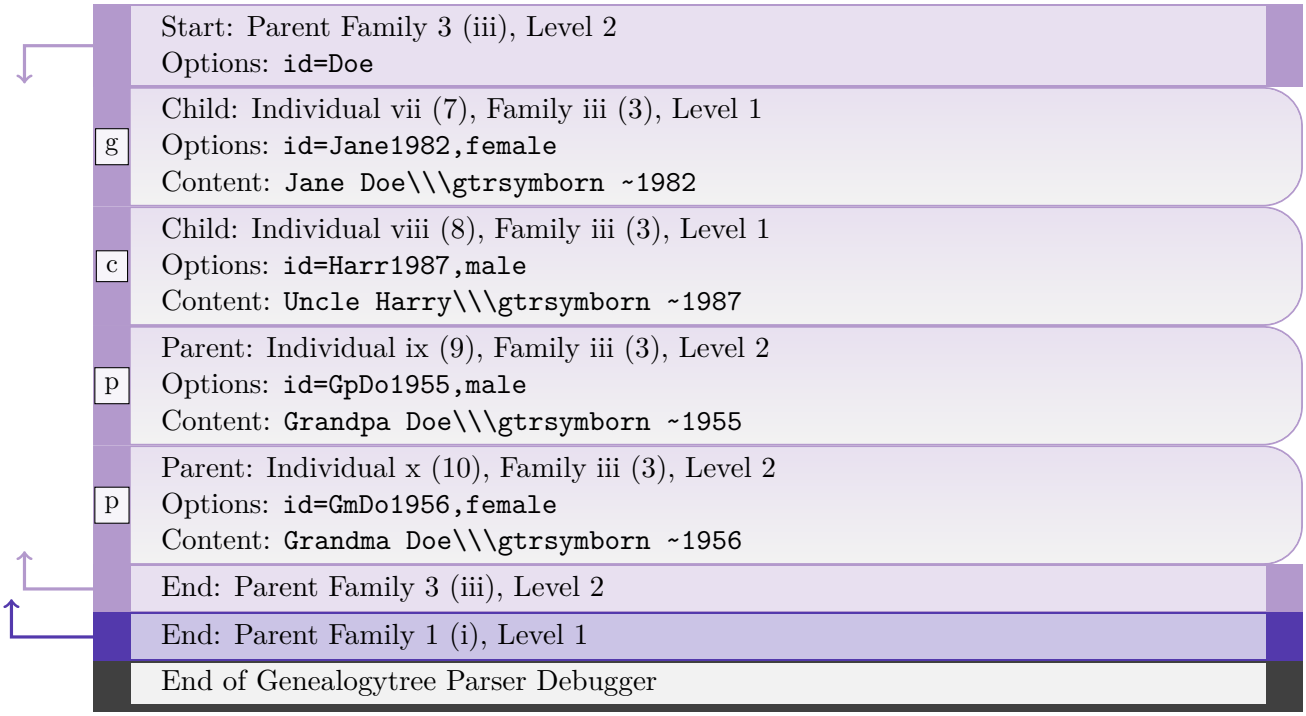
Start: Parent Family 2 (ii), Level 2
Options: `id=Smith`

g — Child: Individual iv (4), Family ii (2), Level 1
Options: `id=John1980,male`
Content: `John Smith\\\gtrsymborn ~1980`

p — Parent: Individual v (5), Family ii (2), Level 2
Options: `id=GpSm1949,male`
Content: `Grandpa Smith\\\gtrsymborn ~1949`

p — Parent: Individual vi (6), Family ii (2), Level 2
Options: `id=GmSm1952,female`
Content: `Grandma Smith\\\gtrsymborn ~1952`

End: Parent Family 2 (ii), Level 2


Start: Parent Family 3 (iii), Level 2
Options: id=Doe
g
Child: Individual vii (7), Family iii (3), Level 1
Options: id=Jane1982,female
Content: Jane Doe\\\gtrsymborn ~1982
c
Child: Individual viii (8), Family iii (3), Level 1
Options: id=Harr1987,male
Content: Uncle Harry\\\gtrsymborn ~1987
p
Parent: Individual ix (9), Family iii (3), Level 2
Options: id=GpDo1955,male
Content: Grandpa Doe\\\gtrsymborn ~1955
p
Parent: Individual x (10), Family iii (3), Level 2
Options: id=GmDo1956,female
Content: Grandma Doe\\\gtrsymborn ~1956
End: Parent Family 3 (iii), Level 2
End: Parent Family 1 (i), Level 1
End of Genealogytree Parser Debugger

# Index