

# The `groupthm` package

Maximilian Keßler

Released 2022-01-12

A central thing in L<sup>A</sup>T<sub>E</sub>X is the usage of “theorems”. With “theorems” we actually mean “environments” that typically have a title, some style applied to their contents and are numbered throughout the document, often later referenced by number and / or name.

Mechanisms for generating such environments are packages like `amsthm`, `ntheorem`, `thmtools`.

While the mechanism in `thmtools` are pretty versatile and suffice for almost all needs, it is pretty time-consuming to largely change the behavior of environments, or have small variants of these.

This package aims at both providing a versatile mechanism,  $\langle theorem\ group \rangle$ s, to structure theorems into groups that can subsequently easily altered, as well as a mechanism for easily generating  $\langle theorem\ families \rangle$ .

As the author is of the opinion that of the mentioned theorem controlling packages `thmtools` provides the most versatile interface, the `groupthm` will be working on top of `thmtools` and use this as a backend for declaring the  $\langle theorem \rangle$ s themselves.

Thus, any styles supported by `thmtools` will be supported by `groupthm` as well, by passing them to `thmtools`.

In this documentation, we first give a more detailed overview of the concepts provided, and then describe the usage in detail.

# Contents

<b>1</b>	<b>Concepts</b>	<b>3</b>
1.1	Theorem groups . . . . .	3
1.2	Grouped theorems . . . . .	3
1.3	Theorem families . . . . .	3
1.4	General notions . . . . .	4
<b>2</b>	<b>Theorem groups</b>	<b>5</b>
2.1	Defining theorem groups . . . . .	5
2.2	Controlling theorem group precedence . . . . .	6
2.3	Inheritance of theorem groups . . . . .	6
2.4	Appending to theorem groups . . . . .	7
2.5	Default theorem groups . . . . .	7
<b>3</b>	<b>Grouped Theorems</b>	<b>8</b>
3.1	Defining grouped theorems . . . . .	8
3.2	Defining families of grouped theorems . . . . .	9
<b>4</b>	<b>L<sup>A</sup>T<sub>E</sub>X3 interface</b>	<b>10</b>
4.1	Theorem groups . . . . .	11
4.2	Grouped theorems . . . . .	11
4.3	Theorem families . . . . .	12
<b>5</b>	<b>groupthm implementation</b>	<b>12</b>
5.1	Dependencies . . . . .	12
5.2	Messages . . . . .	13
5.3	Allocation and initialization . . . . .	14
5.4	Key interface . . . . .	16
5.5	Theorem groups . . . . .	18
5.6	Iterating over powersets . . . . .	28
5.7	Grouped Theorems . . . . .	29
5.8	Theorem families . . . . .	32
5.9	Theorem family options . . . . .	34
5.10	Caching . . . . .	37
5.11	Benchmarking . . . . .	40
	<b>Index</b>	<b>41</b>

# 1 Concepts

## 1.1 Theorem groups

A *⟨theorem group⟩* is some named group holding some properties for the *⟨theorem⟩*s that are contained in this group. Each *⟨theorem⟩* can, when declared, be part of arbitrarily many *⟨theorem group⟩*s, and will be subject to the styles these groups defined.

This enables to group similar *⟨theorem⟩*s and alter them at a late stage of document development in a unique manner, by only having to change the definition of the *⟨theorem group⟩*, and not all *⟨theorem⟩*s separately.

The properties. Such a *⟨theorem group⟩* can hold are as follows

**prefix** A prefix (any *⟨token list⟩*) that will be inserted before the theorem name of each member of this *⟨theorem group⟩*.

**suffix** A suffix (any *⟨token list⟩*) that will be inserted before the theorem name for each member of this *⟨theorem group⟩*. This could be e.g. some “★” appended to the name to indicate variants of environments.

**mapname** A *⟨function⟩* (some macro that takes exactly one argument) that is applied to the **name**.

**thmtools** A *⟨clist⟩* of key-value pairs that are passed to the underlying thmtools backend of the *⟨theorem⟩*. This allows e.g. to set the **topskip** of a certain class of *⟨theorem⟩*s.

The most versatile key here is certainly the **thmtools** key, providing the most customization to an end user (like you).

As mentioned, each *⟨theorem⟩* can be member of arbitrary many *⟨theorem group⟩*s, and will posses their corresponding properties.

To adjust finer controlling of these *⟨theorem group⟩*s, *⟨theorem group⟩*s can inherit from each other, and *⟨theorem group⟩*s are subject to a hierarchy that controls precedence in case of conflicting properties of different *⟨theorem group⟩*s a *⟨theorem⟩* may be part of.

This hierarchy can of course be controlled by the user.

## 1.2 Grouped theorems

A *⟨grouped theorem⟩* is just a theorem that is a member of a given set of groups (possibly empty). It behaves just a regular theorem, except that by changing the definition of its theorem groups, we can alter its behavior.

It is the core concept of the **groupthm** package. For brevity, we will often talk about “theorems”, although in fact we mean “grouped theorems”.

## 1.3 Theorem families

Often, one needs some small *⟨theorem variant⟩*s of some *⟨theorem⟩*, the most typical example being **starred** version of *⟨theorem⟩*s that are not numbered in contrast to their counterparts.

```
\begin{theorem}
  This theorem is numbered.
\end{theorem}
```

```

\begin{theorem*}
  This theorem is not numbered.
  Probably because we do not want a reference to it.
\end{theorem*}

```

`groupthm` extends this idea and provides a versatile mechanism to define a  $\langle theorem family \rangle$ , which is based on some  $\langle theorem name \rangle$  and parses additional arguments / syntax to control the  $\langle theorem groups \rangle$  that this environment is a part of.

So, in addition the name of a  $\langle theorem \rangle$ , the corresponding environment will accept some options and toggle the membership of certain  $\langle theorem groups \rangle$ , thus further customizing its appearance.

This can lead e.g. to usages like the following:

```

\begin{theorem}*
  This theorem has a visual * at its name.
\end{theorem}

```

Providing this consists of two parts: declaring the  $\langle theorem family \rangle$  by listing the groups that can be toggled by this  $\langle theorem family \rangle$ , and declaring the actual option parsing of the  $\langle theorem family \rangle$ , which then controls the membership in these groups (and of course prior to this the definition of the desired  $\langle theorem group \rangle$ s).

## 1.4 General notions

In many cases, there are a number of variants of some command, call it for example `\Foo`. Then the documentation will look like

---

<code>\NewFoo</code>	Defines some <code>foo</code> ...
<code>\RenewFoo</code>	
<code>\ProvideFoo</code>	
<code>\DeclareFoo</code>	

---

and will not mention anything about the variants. This follows some general naming convention that also `xparse` uses, and is the following:

`\NewFoo` Defines `foo` if not defined already. This emits an error in case it has been defined yet.

`\RenewFoo` Redefines `foo` if defined already. This emits an error in case it has *not* been defined yet.


`\ProvideFoo` Defines `foo` if it is not defined already. This does not emit an error if `foo` is already defined (and has no effect in this case).

`\DeclareFoo` Defines `foo` in disregard of any existing definitions. Any old definition will be overwritten (if present).

The documentation margin will list all variants that are available, they follow their respective conventions.

**TeXhackers note:** The `thmtools` package, unfortunately, dose not follow this convention, as its `\declaretheorem` command actually behaves like a `\newtheorem`. The reason for this is that `amsthm` already defines `\newtheorem`.

Thus, actually, calls to `\NewGroupedTheorem` will have an underlying `\declaretheorem`, but you do not have to bother with this.

 Paragraphs like this that begin with a dangerous bend sign give more details that could be used, but whose use is not recommended, as it does not apply to the usual naming conventions. Use at your own responsibility.

## 2 Theorem groups

### 2.1 Defining theorem groups

<code>\NewTheoremGroup</code>	<code>\NewTheoremGroup[⟨key=value list⟩]{⟨theorem group⟩}</code>
<code>\RenewTheoremGroup</code>	This introduces a new <code>⟨theorem group⟩</code> with the given name. The <code>⟨key=value list⟩</code> available are the same as introduced in <a href="#">subsection 1.1</a> :
<code>\ProvideTheoremGroup</code>	
<code>\DeclareTheoremGroup</code>	
	<code>prefix = ⟨token list⟩</code> . Insert the <code>⟨token list⟩</code> in front of the theorem name.
	<code>suffix = ⟨token list⟩</code> . Insert the <code>⟨token list⟩</code> after the theorem name.
	<code>mapname = ⟨function⟩</code> . Apply this <code>⟨function⟩</code> to the theorem name.
	<code>thmtools = {⟨clist⟩}</code> . Pass these options to <code>thmtools</code> .

For uniqueness of the given options, the `⟨clist⟩` given to the `thmtools` key has to be surrounded by a pair of braces.

**T<sub>E</sub>Xhackers note:** The `mapname` is expected to be a function of `\fun:n`. The function call is subject to an `x`-type expansion prior to being passed further to `thmtools`.

## 2.2 Controlling theorem group precedence

---

<code>\DeclareTheoremGroupRule</code>	<code>\DeclareTheoremGroupRule[⟨keyname⟩] {⟨theorem group<sub>1</sub>⟩}{⟨relation⟩}{⟨theorem group<sub>2</sub>⟩}</code>
---------------------------------------	---

---

This declares some relation between the two theorem groups, controlling their order of application in case a theorem is member of both groups.

The `⟨keyname⟩` can be one of `prefix`, `suffix`, `mapname`, `thmtools`. If present, it declares the corresponding relation only for this subkey. This can lead to `⟨theorem group⟩` overwriting `⟨theorem group2⟩` when given contradictory `thmtools` options, but the `prefix` of `⟨theorem group⟩` being applied after the one of `⟨theorem group2⟩`. When the `⟨keyname⟩` is not given, this applies to all keywords.

**TeXhackers note:** The `⟨keyname⟩` is just passed to the corresponding argument of the `lthooks` package. If the option argument is not present, `??` is used, this has the described effect.

The behavior of the relations is based on the `\DeclareHookRule` command from the `xparse` package, and all respective keys are in fact available, but typically not needed, so the reader of this manual is referred to the `lthooks` packages documentation for a list of the full keys. For us, the following list will suffice:

**higher or after or >** `⟨theorem group1⟩` takes precedence over `⟨theorem group2⟩`. Its `prefix` is applied after the one of `⟨theorem group2⟩`.

**lower or before or <** `⟨theorem group2⟩` takes precedence over `⟨theorem group1⟩`. Its `prefix` is applied after the one of `⟨theorem group1⟩`.

**TeXhackers note:** The `⟨relation⟩` is first stripped, then checked if it matches either **higher** or **lower** and in this case replaced by the corresponding `lthooks` variant of the relation. The rest is passed as is to `lthooks` and thus subject to the usual normalization process of `lthooks`.

## 2.3 Inheritance of theorem groups

---

<code>\AddTheoremGroupParent</code>	<code>\AddTheoremGroupParent{⟨theorem group<sub>1</sub>⟩}{⟨theorem group<sub>2</sub>⟩}</code>
-------------------------------------	---

---

Declares `⟨theorem group1⟩` to “inherit” all properties of `⟨theorem group2⟩`. In other words, `⟨theorem group2⟩` is a parent of `⟨theorem group1⟩` in a usual inheritance graph.

The definitions of the groups themselves are unchanged, but each new theorem defined with `⟨theorem group1⟩` will also have the properties of `⟨theorem group2⟩`.

Inheritance is transitive, when defining a new theorem, we just flatten out the inheritance graph and apply all properties.

Inheritance is subject to the usual theorem group hierarchies as discussed in [subsection 2.2](#). This can even yield situations, where `⟨theorem group1⟩` inherits from `⟨theorem group2⟩`, but `⟨theorem group2⟩` overwrites `⟨theorem group1⟩`.

## 2.4 Appending to theorem groups

---

---

`\AppendToTheoremGroup` `\AppendToTheoremGroup[⟨key=value list⟩]{⟨theorem group⟩}`

Adds the properties given as `⟨key=value list⟩` to the theorem group. The syntax for the `⟨key=value list⟩` is the same as in `\NewTheoremGroup`.

## 2.5 Default theorem groups

There are a number of theorem groups that `groupthm` will initially declare and that have certain special treatment in some places.

---

**all** Every declared grouped theorem is a member of this group.  
Initially, this group has no effect (i.e. an empty property list). It can be redefined by the user to alter the behavior of all grouped theorems in a unified way.  
It is the lowest theorem group in the hierarchy by default.

---

**starred** This is group that shall represent the standard variant of theorems that are called with a “\*” in the environment name. Theorems of this group are not numbered.  
The user should not add theorems to this group by hand, as this is handled in a unified way by default. See the documentation for `\NewGroupedTheorem`, `\NewGroupedTheoremFamily` and `\NewGroupedTheoremFamilyOptions` how this group is treated.  
It is the highest theorem group in the hierarchy by default, except for `unnumbered`, with which it has no relation.

---

**unnumbered** Theorems in this group are not numbered. Versions of all commands exist that add theorems to this group.  
It is the highest theorem group in the hierarchy by default, except for `starred`, with which it has no relation.  
The reason for the two groups `starred` and `unnumbered` to both exist is that the `starred` group is *meant* to be applied to theorems that were called with a “\*” in their name (thus the name), whereas the “unnumbered” group *means* that the environment is ‘just unnumbered’.  
This has two reasons: First, this enables more fine-tuning of the behavior of the theorems in post-processing of a document. Second, more importantly, this distinguishes semantically between the environments `theorem` and `theorem*`, even if `theorem` is in the `unnumbered` group.

So assuming that `theorem` is member of the `unnumbered` group, both calls


```
\begin{theorem}
  This is not numbered.
\end{theorem}
\begin{theorem*}
  This is not numbered.
\end{theorem*}
```

are defined and will produce the same result by default, but we could still change the definition of the `starred` group later to do anything we want.

**T<sub>E</sub>Xhackers note:** The mentioned hierarchies are kept intact for newly defined theorem groups, i.e. for each new such group, two theorem group rules are created.

## 3 Grouped Theorems

### 3.1 Defining grouped theorems

<code>\NewGroupedTheorem</code>	<code>\NewGroupedTheorem[⟨key=value list⟩]{⟨grouped theorem⟩}</code>
<code>\ProvideGroupedTheorem</code>	This defines <code>⟨grouped theorem⟩</code> and <code>⟨grouped theorem*⟩</code> as new theorem environments. Its properties can be set by the following keys:
	<p><code>name</code> = <code>⟨displayed name⟩</code>. If given, this is the displayed name of the environment in the document. If not present, the <code>⟨grouped theorem⟩</code> is also used as the <code>⟨displayed name⟩</code> in capitalized form.</p> <p><code>group</code> = <code>{⟨clist⟩}</code></p> <p>Makes this theorem a member of the listed groups. It will inherit all respective properties of these groups.</p> <p>If groups are present more than one time, this has no (additional) effect.</p> <p><code>thmtools</code> = <code>{⟨clist⟩}</code></p> <p>Passes these option to the <code>thmtools</code> environment that is declared internally.</p> <p>The <code>⟨grouped theorem*⟩</code> behaves the same as the <code>⟨grouped theorem⟩</code>, but additionally will be a member of the <code>starred</code> theorem group, see <a href="#">subsection 2.5</a>.</p> <p> If you don't wish the <code>⟨grouped theorem*⟩</code> variant to be generated, you can pass the additional option <code>starred version = false</code>. If you are not sure about this, you are probably fine without this option.</p>

<code>\NewGroupedTheorem*</code>	<code>\NewGroupedTheorem*[⟨key=value list⟩]{⟨grouped theorem⟩}</code>
<code>\ProvideGroupedTheorem*</code>	Behaves the same as <code>\NewGroupedTheorem</code> , but also adds the theorem(s) to the default <code>unnumbered</code> group, thus resulting in the environment not being numbered.
	This is thus equivalent to using <code>\NewGroupedTheorem</code> and adding the <code>unnumbered</code> group.

## 3.2 Defining families of grouped theorems

---

<code>\NewGroupedTheoremFamily</code>	<code>\NewGroupedTheoremFamily[⟨key=value list⟩]{⟨theorem family⟩}</code>
<code>\ProvideGroupedTheoremFamily</code>	

---

Defines a family of grouped theorems. The `⟨key=value list⟩` accept the same arguments as the `\NewGroupedTheorem` macro. However, for each *subset* of the given groups, a grouped theorem is defined.

These grouped theorems are not meant to be accessed directly (but could), so we omit their actual (internal) names here. To call these, some `GroupedTheoremFamilyOptions` have to specified, see `\NewGroupedTheoremFamilyOptions`.

Also, to the given groups, the **starred** group is added automatically.



If you do not wish the **starred** versions, you can set the key `starred version = false`.

---

<code>\NewGroupedTheoremFamily*</code>
<code>\ProvideGroupedTheoremFamily*</code>

---

Behaves the same as `\NewGroupedTheoremFamily`, but also adds each variant to the default **unnumbered** group, thus resulting in the environments not being numbered.

This is *almost* equivalent to calling `\NewGroupedTheoremFamily` with the **unnumbered** group being present, as it does not generate the variants where the **unnumbered** group is not present.

---

<code>\AddTheoremToGroup</code>	<code>\AddTheoremToGroup{⟨theorem group⟩}</code>
---------------------------------	--

---

Means that the current invocation of a theorem family should call the theorem variant with the given group.

Can only be used in the body of `\NewGroupedTheoremFamilyOptions` or similarly.

---

<code>\NewGroupedTheoremFamilyOptions</code>	<code>\NewGroupedTheoremFamilyOptions{⟨theorem family⟩}{⟨argument</code>
<code>\RenewGroupedTheoremFamilyOptions</code>	<code>specification⟩}</code>
<code>\ProvideGroupedTheoremFamilyOptions</code>	<code>{⟨selection body⟩}</code>
<code>\DeclareGroupedTheoremFamilyOptions</code>	

---

Defines two new environment with options, given by  $\langle theorem\ family \rangle$  and  $\langle theorem\ family^* \rangle$ . The  $\langle argument\ specification \rangle$  can be any valid `xparse` argument specification.

The  $\langle selection\ body \rangle$  is there to process the options of the  $\langle argument\ specification \rangle$  and select which variant of the  $\langle theorem\ family \rangle$  to enter. The arguments are available as usual with `xparse` by `#1`, `#2`, ...

The body may also call any number of `\AddTheoremToGroup` calls, which the enables the corresponding groups to be toggled.

When the environment is called within the document, the options are parsed as with `xparse` and the  $\langle selection\ body \rangle$  is executed. Immediately after, the theorem variant of  $\langle theorem\ family \rangle$  with the specified groups by `\AddTheoremToGroup` is called.

At the end of the environment, the  $\langle selection\ body \rangle$  is executed again and the called theorem variant is ended again.

The possible theorem variants that the newly declared environment will call *have to be generated subsequently* by a call to the `\NewGroupedTheoremFamily` function.



As always, if you do not wish the  $\langle theorem\ family^* \rangle$  version to be generated, you can pass `starred version = false` as an additional key.

---

<code>\NewGroupedTheoremFamilyOptions*</code>	<code>\NewGroupedTheoremFamilyOptions*{⟨theorem family⟩}{⟨argument</code>
<code>\RenewGroupedTheoremFamilyOptions*</code>	<code>specification⟩}</code>
<code>\ProvideGroupedTheoremFamilyOptions*</code>	<code>{⟨selection body⟩}</code>
<code>\DeclareGroupedTheoremFamilyOptions*</code>	

---

Does the same as `\NewGroupedTheoremFamilyOptions`, but calls the variants with the additional `unnumbered` group.

The possible theorem variants have to be generated with the `\NewGroupedTheoremFamily*` command before.

## 4 L<sup>A</sup>T<sub>E</sub>X3 interface

There is also an underlying L<sup>A</sup>T<sub>E</sub>X3 interface provided by the package (and in fact, all prior documented macros are just wrappers around this internal programming interface.

When building on top of this package, you can also use this interface, which is possibly easier to use in some cases.

Most of the time, however, the document level commands will provide a better interface that just accepts more options than the underlying L<sup>A</sup>T<sub>E</sub>X3 interface does. Feel free to just use these directly.

In general, for functions that use key-value syntax, there are typically three (public) versions of the command, namely

- A L<sup>A</sup>T<sub>E</sub>X3 command that requires all key-values as mandatory arguments, so this does not use the key-value interface. Use this if you already know with which keys you deal and know their corresponding values.
- A L<sup>A</sup>T<sub>E</sub>X3 command having the first argument accepting the keys as a comma-separated list. Use this if you want to profit of the key-value syntax.

- A L<sup>A</sup>T<sub>E</sub>X2e document command. These were documented before, and these just wrap the second type of command.

## 4.1 Theorem groups

<u>\groupthm_new_group:nn</u>	<u>\groupthm_new_group:nn{&lt;key=value list&gt;}{&lt;theorem group&gt;}</u>
<u>\groupthm_renew_group:nn</u>	
<u>\groupthm_provide_group:nn</u>	L <sup>A</sup> T <sub>E</sub> X3 versions of \NewTheoremGroup, \RenewTheoremGroup, \ProvideTheoremGroup
<u>\groupthm_declare_group:nn</u>	and \DeclareTheoremGroup

<u>\groupthm_new_group:nnnnn</u>	<u>\groupthm_new_group:nnnnn{&lt;theorem group&gt;}{&lt;prefix&gt;}</u>
<u>\groupthm_new_group:nVVVV</u>	<u>{&lt;suffix&gt;}{&lt;mapname clist&gt;}{&lt;thmtools clist&gt;}</u>
<u>\groupthm_renew_group:nnnnn</u>	
<u>\groupthm_renew_group:nVVVV</u>	
<u>\groupthm_provide_group:nnnnn</u>	
<u>\groupthm_provide_group:nVVVV</u>	
<u>\groupthm_declare_group:nnnnn</u>	
<u>\groupthm_declare_group:nVVVV</u>	

Non-keyval versions of \groupthm\_new\_group:nn, \groupthm\_renew\_group:nn, \groupthm\_provide\_group:nn and \groupthm\_declare\_group:nn

These take the individual values of the keyval keys directly, in the order indicated by the syntax specification.

<u>\groupthm_declare_group_rule:nnnn</u>	<u>\groupthm_declare_group_rule:nnnn{&lt;keyname&gt;}{&lt;theorem group<sub>1</sub>&gt;}</u> <u>{&lt;relation&gt;}{&lt;theorem group<sub>2</sub>&gt;}</u>
--	--

L<sup>A</sup>T<sub>E</sub>X3 version of \DeclareTheoremGroupRule

<u>\groupthm_add_parent:nn</u>	<u>\groupthm_add_parent:nn{&lt;theorem group<sub>1</sub>&gt;}{&lt;theorem group<sub>2</sub>&gt;}</u>
--------------------------------	--

L<sup>A</sup>T<sub>E</sub>X3 version of \AddTheoremGroupParent.

<u>\groupthm_append_to_group:nn</u>	<u>\groupthm_append_to_group:nn{&lt;key=value list&gt;}{&lt;theorem group&gt;}</u>
-------------------------------------	--

L<sup>A</sup>T<sub>E</sub>X3 version of \AppendToTheoremGroup.

## 4.2 Grouped theorems

<u>\groupthm_new_theorem:nnnn</u>	<u>\groupthm_new_theorem:nnnn{&lt;grouped theorem&gt;}{&lt;groups&gt;}</u>
<u>\groupthm_provide_theorem:nnnn</u>	<u>{&lt;name&gt;}{&lt;thmtools keys&gt;}</u>

<u>\groupthm_new_theorem:nnn</u>	<u>\groupthm_new_theorem:nnn{&lt;key=value list&gt;}{&lt;theorem group&gt;}{&lt;bool&gt;}</u>
<u>\groupthm_provide_theorem:nnn</u>	

L<sup>A</sup>T<sub>E</sub>X3 version of \NewGroupedTheorem. The given <bool> indicates the presence of the “\*”, i.e. if it is true, the new theorem will be additionally added to the **unnumbered** group.

### 4.3 Theorem families

---

<code>\groupthm_new_family:nnn</code>	<code>\groupthm_new_family:nnn{&lt;key=value list&gt;}{&lt;theorem family&gt;}{&lt;bool&gt;}</code>
<code>\groupthm_provide_family:nnn</code>	

---

L<sup>A</sup>T<sub>E</sub>X3 version of `\NewGroupedTheoremFamily`. The given `<bool>` indicates the presence of the “\*”, i.e. if it is true, the new theorem will be additionally added to the unnumbered group.

---

<code>\groupthm_new_family:nnnnn</code>	<code>\groupthm_new_family:nnnnn{&lt;theorem family&gt;}{&lt;groups<sub>1</sub>&gt;}{&lt;name&gt;}</code>
<code>\groupthm_new_family:nVVVV</code>	<code>{&lt;thmtools clist&gt;}{&lt;groups<sub>2</sub>&gt;}</code>
<code>\groupthm_provide_family:nnnnn</code>	
<code>\groupthm_provide_family:nVVVV</code>	

---

Non-keyval version of `\groupthm_new_family:nnn`. The `<groups2>` will be added to each generated variant, i.e. we generate a variant for the union of (the powerset of `<groups1>`) and `<groups2>`.

---

<code>\groupthm_add_theorem_to_group:n</code>	<code>\groupthm_add_theorem_to_group:n{&lt;theorem group&gt;}</code>
---	--

---

L<sup>A</sup>T<sub>E</sub>X3 version of `\AddTheoremToGroup`

---

<code>\groupthm_new_family_options:nnnn</code>	<code>\groupthm_new_family_options:nnnn{&lt;theorem family&gt;}{&lt;arg spec&gt;}</code>
<code>\groupthm_renew_family_options:nnnn</code>	<code>{&lt;selection body&gt;}{&lt;groups&gt;}</code>
<code>\groupthm_provide_family_options:nnnn</code>	
<code>\groupthm_declare_family_options:nnnn</code>	

---

L<sup>A</sup>T<sub>E</sub>X3 version of `\NewGroupedTheoremFamilyOptions`. The `<groups>` is a comma separated list of groups that will always be added to the variants called.

So, `\NewGroupedTheoremFamilyOptions*` will e.g. add `unnumbered` to this list.

## 5 groupthm implementation

```

1 <*package>
2 <@@=groupthm>

```

### 5.1 Dependencies

First, we import other packages on which we rely on, and set up some private wrappers around these.

```

3 \RequirePackage{amsthm}
4 \RequirePackage{thmtools}
5 <*benchmark>
6 \RequirePackage{l3benchmark}
7 </benchmark>
8 \RequirePackage{l3keys2e}

```

<code>\_groupthm_thmtools_declare_theorem:nn</code>	<code>\_groupthm_thmtools_declare_theorem:nn &lt;theorem name&gt;{&lt;thmtools keyval args&gt;}</code>
<code>\_groupthm_thmtools_declare_theorem:Vn</code>	This is just a private wrapper around <code>\declaretheorem</code> of the <code>thmtools</code> package. We additionally cache
9	<code>\cs_new:Npn \_groupthm_thmtools_declare_theorem:nn #1 #2</code>

```

10 {
11   \tl_log:n { Declaring ~ thmtools ~ theorem ~ #2 }
12   \declaretheorem [ #1 ] { #2 }
13   \__groupthm_cache:n
14   {
15     \csname __groupthm_thmtools_declare_theorem:nn \endcsname { #1 } { #2 }
16   }
17 }
18 \cs_generate_variant:Nn \__groupthm_thmtools_declare_theorem:nn { V n }

```

(End of definition for `\__groupthm_thmtools_declare_theorem:nn`.)

It also comes in handy to have a stronger version of the hook role setting mechanism:

`\__groupthm_hook_gset_rule_foreach:nNnn`

`\__groupthm_hook_gset_rule_foreach:nNnn{<hook>}{<clist name>}{<relation>}{<label>}`

This is a wrapper around the `\hook_gset_rule:nnnn` macro that takes a clist name of labels, and executes the corresponding command for each such label.

```

19 \cs_new:Npn \__groupthm_hook_gset_rule_foreach:nNnn #1 #2 #3 #4
20 {
21   \cs_set:Npn \__groupthm_map_aux:n ##1
22   {
23     \hook_gset_rule:nnnn { #1 } { ##1 } { #3 } { #4 }
24   }
25   \clist_map_function:NN #2 \__groupthm_map_aux:n
26 }

```

(End of definition for `\__groupthm_hook_gset_rule_foreach:nNnn`.)

## 5.2 Messages

These are messages that we might emit.

When an unknown group is used somewhere:

```

\msg_error:nnn{ groupthm }{ unknown group }{<groupname>}
27 \msg_new:nnn { groupthm } { unknown ~ group }
28 {
29   Unknown ~ group ~ '#1' ~ supplied ~ \msg_line_context:
30 }

```

When an unknown key has been used:

```

\msg_error:nnn{ groupthm }{ unknown key }{<key>}
31 \msg_new:nnn { groupthm } { unknown ~ key }
32 {
33   Unknown ~ key ~ '#1' ~ supplied ~ \msg_line_context:
34 }

```

Some data structure is already defined or not defined yet.

```

\msg_error:nnnn{ groupthm }{ wrong definition }
  {<type>}{<name>}{<already || not>}
35 \msg_new:nnn { groupthm } { wrong ~ definition }
36 {
37   Bad ~ definition ~ of ~ #1 ~ '#2' ~ \msg_line_context:, ~ #1 ~ is ~ #3 ~ defined.
38 }

```

When the special `\AddTheoremToGroup` macro is issued outside a theorem family options body.

```

\msg_error:nn { groupthm }{ misuse add theorem to group }

39 \msg_new:nnn { groupthm } { misuse ~ add ~ theorem ~ to ~ group }
40 {
41   Bad ~ usage ~ of ~ 'AddTheoremToGroup' ~ macro ~ outside ~ theorem ~
42   family ~ options ~ \msg_line_context:
43 }

```

When a theorem family is invoked, but has not been generated yet.

```

\msg_error:nn { groupthm }{ undefined theorem variant }

44 \msg_new:nnnn { groupthm } { undefined ~ theorem ~ variant }
45 {
46   Bad ~ call ~ of ~ theorem ~ variant ~ of ~ '#1' ~ \msg_line_context:
47 }
48 {
49   You ~ wanted ~ to ~ call ~ the ~ variant ~ with ~ group(s) ~
50   '#2' ~ of ~ theorem ~ family ~ '#1', ~ but ~ it ~ has ~ not ~ been ~
51   generated ~ yet. ~
52   Probably ~ you ~ forgot ~ this. ~
53   \msg_see_documentation_text:n { groupthm }
54 }

```

### 5.3 Allocation and initialization

We use hooks at several places. However, these are not intended for outer use, and we thus mark them with a preceding `__`.

```

55 \hook_new:n { __groupthm/prefix }
56 \hook_new:n { __groupthm/suffix }
57 \hook_new:n { __groupthm/mapname }
58 \hook_new:n { __groupthm/thmtools }
59 \hook_new:n { __groupthm/groupsort }

```

`\hook_gset_rule:nnVn`      `\hook_gset_rule:nnVn{<hook>}{<label1>}{<relation>}{<label2>}`  
Just a variant of the usual `\hook_gset_rule:nnnn` macro that we use.

```

60 \cs_generate_variant:Nn \hook_gset_rule:nnnn { n n V n }

```

*(End of definition for `\hook_gset_rule:nnVn`. This function is documented on page ??.)*

These variables will be set by the key-value interface provided by `l3keys` and are used in various places in the package.

```

61 \bool_new:N \l__groupthm_key_starred_version_bool
62 \tl_new:N \l__groupthm_key_prefix_tl
63 \tl_new:N \l__groupthm_key_name_tl
64 \tl_new:N \l__groupthm_key_suffix_tl
65 \clist_new:N \l__groupthm_key_group_clist
66 \clist_new:N \l__groupthm_key_mapname_clist
67 \clist_new:N \l__groupthm_key_thmtools_clist

```

*(End of definition for `\l__groupthm_key_starred_version_bool` and others.)*

$\backslash l\_groupthm\_starred\_version\_bool$ $\backslash l\_groupthm\_prefix\_tl$ $\backslash l\_groupthm\_name\_tl$ $\backslash l\_groupthm\_suffix\_tl$ $\backslash l\_groupthm\_mapname\_clist$ $\backslash l\_groupthm\_thmtools\_clist$ $\backslash l\_groupthm\_group\_clist$	<p>General local variables. Will typically be used to extract the variables set by the <code>l3keys</code> interface, but also in just a local variable sense.</p> <pre> 68 \tl_new:N \l_groupthm_prefix_tl 69 \tl_new:N \l_groupthm_name_tl 70 \tl_new:N \l_groupthm_suffix_tl 71 \clist_new:N \l_groupthm_mapname_clist 72 \clist_new:N \l_groupthm_thmtools_clist 73 \clist_new:N \l_groupthm_group_clist </pre> <p>(End of definition for <code>\l_groupthm_starred_version_bool</code> and others.)</p>
$\backslash g\_groupthm\_defined\_groups\_clist$	<p>This variable will hold a global list of declared theorem groups</p> <pre> 74 \clist_new:N \g_groupthm_defined_groups_clist </pre> <p>(End of definition for <code>\g_groupthm_defined_groups_clist</code>.)</p>
$groupthm\_in\_family\_options\_environment\_bool$	<p>This variable indicates whether we are in the special environment used to parse a set of groups out of options given to a theorem family.</p> <p>This bool toggles the availability of the special <code>\AddTheoremToGroup</code> macro.</p> <pre> 75 \bool_new:N \l_groupthm_in_family_options_environment_bool </pre> <p>(End of definition for <code>\l_groupthm_in_family_options_environment_bool</code>.)</p>
$\backslash g\_groupthm\_append\_groups\_int$	<p>This counts the number of times we appended to a group.</p> <pre> 76 \int_new:N \g_groupthm_append_groups_int </pre> <p>(End of definition for <code>\g_groupthm_append_groups_int</code>.)</p>
$\backslash g\_groupthm\_cache\_bool$	<p>This stores whether we cache definitions in the aux file.</p> <pre> 77 \bool_new:N \g_groupthm_cache_bool </pre> <p>(End of definition for <code>\g_groupthm_cache_bool</code>.)</p>
$\backslash g\_groupthm\_cache\_version\_aux\_int$ $\backslash g\_groupthm\_cache\_version\_document\_int$	<p>The version of the cached theorems in the aux file and the version the package has been loaded with.</p> <pre> 78 \int_new:N \g_groupthm_cache_version_aux_int 79 \int_new:N \g_groupthm_cache_version_document_int 80 \int_set:Nn \g_groupthm_cache_version_aux_int { -1 } </pre> <p>(End of definition for <code>\g_groupthm_cache_version_aux_int</code> and <code>\g_groupthm_cache_version_document_int</code>.)</p>
$\backslash g\_groupthm\_lazy\_document\_tl$ $\backslash g\_groupthm\_lazy\_auxfile\_tl$	<p>During the preamble, if in caching mode, we will collect definitions for theorems in these token lists. At the beginning of the document, we select the proper one.</p> <pre> 81 \tl_new:N \g_groupthm_lazy_document_tl 82 \tl_new:N \g_groupthm_lazy_auxfile_tl </pre> <p>(End of definition for <code>\g_groupthm_lazy_document_tl</code> and <code>\g_groupthm_lazy_auxfile_tl</code>.)</p>
$\backslash g\_groupthm\_dump\_auxfile\_clist$ $\backslash g\_groupthm\_dump\_cache\_clist$	<p>With these, we collect stuff written to the aux file at the end of the document</p> <p>The <code>\g_groupthm_dump_cache_clist</code> will be treated specially: We will dump such that in a re-run of <math>\text{\LaTeX}</math>, the contents will be available for execution in the <code>\g_groupthm_lazy_auxfile_tl</code></p> <pre> 83 \clist_new:N \g_groupthm_dump_auxfile_clist 84 \clist_new:N \g_groupthm_dump_cache_clist </pre> <p>(End of definition for <code>\g_groupthm_dump_auxfile_clist</code> and <code>\g_groupthm_dump_cache_clist</code>.)</p>

## 5.4 Key interface

As mentioned, all keys will set their corresponding local variables (containing “\_key\_” in their name) and store the user input in these.

Additionally, we group these keys by use cases, and provide defaults that in most cases will not require further handling.

```

85 \keys_define:nn { groupthm }
86 {
87   cache .bool_set:N = \g__groupthm_cache_bool,
88   cache .default:n = { true },
89   __cache_version__ .int_set:N = \g__groupthm_cache_version_document_int,
90   cache version .meta:nn = { groupthm } { cache = true, __cache_version__ = #1 },
91   cache version .default:n = { 0 },
92 }
93 \keys_define:nn { groupthm / theorem ~ group }
94 {
95   prefix .tl_set:N = \l__groupthm_key_prefix_tl,
96   prefix .default:n = \c_empty_tl,
97   suffix .tl_set:N = \l__groupthm_key_suffix_tl,
98   suffix .default:n = \c_empty_tl,
99   suffix .groups:n = { theoremgroup },
100  map ~ name .clist_set:N = \l__groupthm_mapname_clist,
101  map ~ name .default:n = {},
102  map ~ name .groups:n = { theoremgroup },
103  thmtools .clist_set:N = \l__groupthm_key_thmtools_clist,
104  thmtools .default:n = {},
105  unknown .code:n =
106    \msg_error:nnx { groupthm } { unknown ~ key } { \str_use:N \l_keys_key_str }
107 }
108 \keys_define:nn { groupthm / grouped ~ theorem }
109 {
110   name .tl_set:N = \l__groupthm_key_name_tl,
111   name .default:n = \c_novalue_tl,
112   group .clist_set:N = \l__groupthm_key_group_clist,
113   group .default:n = {},
114   thmtools .clist_set:N = \l__groupthm_key_thmtools_clist,
115   thmtools .default:n = {},
116   starred ~ version .bool_set:N = \l__groupthm_key_starred_version_bool,
117   starred ~ version .default:n = { true },
118   unknown .code:n =
119     \msg_error:nnx { groupthm } { unknown ~ key } { \str_use:N \l_keys_key_str }
120 }
121 \keys_define:nn { groupthm / theorem ~ family }
122 {
123   name .tl_set:N = \l__groupthm_key_name_tl,
124   name .default:n = \c_novalue_tl,
125   group .clist_set:N = \l__groupthm_key_group_clist,
126   group .default:n = {},
127   thmtools .clist_set:N = \l__groupthm_key_thmtools_clist,
128   thmtools .default:n = {},
129   starred ~ version .bool_set:N = \l__groupthm_key_starred_version_bool,
130   starred ~ version .default:n = { true },
131   unknown .code:n =
132     \msg_error:nnx { groupthm } { unknown ~ key } { \str_use:N \l_keys_key_str }

```

```

133 }
134 \keys_define:nn { groupthm / theorem ~ family ~ options }
135 {
136   starred ~ version      .bool_set:N      = \l__groupthm_key_starred_version_bool,
137   starred ~ version      .default:n       = { true },
138   unknown                .code:n          =
139     \msg_error:nnx { groupthm } { unknown ~ key } { \str_use:N \l_keys_key_str }
140 }

```

Process package options:

```

141 \ProcessKeysOptions { groupthm }

```

The only key whose default requires such handling is the “name” key, which will be set to a capitalized version of the environment name when not specified.

```

\__groupthm_set_normalized_keys:nnn    \__groupthm_set_normalized_keys:nnn{<key=value list>}{<key group>}{<fallback
name>}

```

Sets the packages keys and normalizes the retrieved values, that is, clears old set keys, stores all keys in local variables, and replaces the `\l__groupthm_name_tl` with the capitalized version of the *<fallback name>*.

```

142 \cs_new:Npn \__groupthm_set_normalized_keys:nnn #1 #2 #3
143 {
144   \keys_set:nn { groupthm / theorem ~ group }
145   { prefix, suffix, thmtools, map ~ name }
146   \keys_set:nn { groupthm / grouped ~ theorem }
147   { name, group, thmtools, starred ~ version }
148   \keys_set:nn { groupthm / theorem ~ family }
149   { name, group, thmtools, starred ~ version }
150   \keys_set:nn { groupthm / #2 } { #1 }

```

Normalize given name

```

151 \tl_if_eq:NnTF \l__groupthm_key_name_tl { \c_novalue_tl }
152 {
153   \tl_set:Nx \l__groupthm_name_tl
154   {
155     \text_titlecase_first:n {#3}
156   }
157 }
158 {
159   \tl_set_eq:NN \l__groupthm_name_tl \l__groupthm_key_name_tl
160 }

```

Copy set keys into local variables

```

161 \bool_set_eq:NN \l__groupthm_starred_version_bool \l__groupthm_key_starred_version_bool
162 \tl_set_eq:NN \l__groupthm_prefix_tl \l__groupthm_key_prefix_tl
163 \tl_set_eq:NN \l__groupthm_suffix_tl \l__groupthm_key_suffix_tl
164 \clist_set_eq:NN \l__groupthm_group_clist \l__groupthm_key_group_clist
165 \clist_set_eq:NN \l__groupthm_mapname_clist \l__groupthm_key_mapname_clist
166 \clist_set_eq:NN \l__groupthm_thmtools_clist \l__groupthm_key_thmtools_clist
167 }

```

(End of definition for `\__groupthm_set_normalized_keys:nnn`.)

## 5.5 Theorem groups

For technical reasons, we need to some arbitrary, but unique total ordering on the set of all defined theorem groups.

Since unfortunately (by now) there is no standard mechanism for sorting strings in L<sup>A</sup>T<sub>E</sub>X3 directly, we use an ugly hack to achieve what we want:

We will use an internal hook, and apply hook rules to each pair of internal groups such that the resulting relation is a total order. Whenever we want to sort a list of groups now, we do the following: First, for each group element in the list, insert into the hook the function “put this group back into the list”, using the group itself as a label. Then we clear the list, and finally execute the hook.

Essentially, we thus split up the hook in the single groups, let the L<sup>A</sup>T<sub>E</sub>X3 hook mechanism take care of the sorting, and restore the sorted single pieces into our list. Of course, this is very inefficient, but for now it seems to be the simplest solution, without having to implement an own string sorting function.

Once there is such a proper mechanism, the author will likely update this to proper string sorting.

```

\__groupthm_add_to_group_ordering:n      \__groupthm_add_to_group_ordering:n{<theorem group>}
Sets hook relations for this group and all already defined theorem groups.

168 \cs_new:Npn \__groupthm_add_to_group_ordering:n #1
169 {
170   \__groupthm_hook_gset_rule_foreach:nNnn
171   { \__groupthm/groupsort }
172   \g__groupthm_defined_groups_clist
173   { before }
174   { #1 }
175 }

(End of definition for \__groupthm_add_to_group_ordering:n.)

\__groupthm_remove_from_group_ordering:n \__groupthm_remove_from_group_ordering:n{<theorem group>}
Removes all relations of this theorem group with the currently defined theorem
groups.

176 \cs_new:Npn \__groupthm_remove_from_group_ordering:n #1
177 {
178   \__groupthm_hook_gset_rule_foreach:nNnn
179   { \__groupthm/groupsort }
180   \g__groupthm_defined_groups_clist
181   { unrelated }
182   { #1 }
183 }

(End of definition for \__groupthm_remove_from_group_ordering:n.)

\__groupthm_add_to_sort_hook:n          \__groupthm_add_to_sort_hook:n{<theorem group>}
Adds the theorem group into the sort hook to be restored later from it. This al-
ready uses the assumption, that we want to use the \l__groupthm_group_clist variable
(which we do).

184 \cs_new:Npn \__groupthm_add_to_sort_hook:n #1
185 {
186   \hook_gput_code:nnn { \__groupthm/groupsort }

```

```

187 { #1 }
188 {
189   \clist_put_left:Nn \l__groupthm_group_clist { #1 }
190 }
191 }

```

(End of definition for \\_\_groupthm\_add\_to\_sort\_hook:n.)

\\_\_groupthm\_sort\_group\_names: As explained briefly before, we first insert all theorems into the hook, clear the list, and then use the hook again.

This then sorts the \l\_\_groupthm\_group\_clist variable, which is also assumed to hold only defined theorem group names.

```

192 \cs_new:Npn \__groupthm_sort_group_names:
193 {
194   \hook_gremove_code:nn { __groupthm/groupsort }{*}
195   \clist_map_function:NN \l__groupthm_group_clist \__groupthm_add_to_sort_hook:n
196   \clist_clear:N \l__groupthm_group_clist
197   \hook_use:n { __groupthm/groupsort }
198 }

```

(End of definition for \\_\_groupthm\_sort\_group\_names:.)

\\_\_groupthm\_define\_group:nnnnn \\_\_groupthm\_define\_group:nnnnn{<theorem group>}{<prefix t1>}{<suffix t1>}{<mapname clist>}{<thmtools clist>}

This creates a new theorem group out of the given parameters. We store all given contents in our (private) hooks, using the group name as the key so that we can later retrieve the components of each group separately.

This is an internal function and assumes that the group is currently not defined, and also removed from all hooks.

```

199 \cs_new:Npn \__groupthm_define_group:nnnnn #1#2#3#4#5
200 {

```

\\_\_groupthm\_use\_group\_\_\meta{theorem\_group}: This is the internal macro that will be called when retrieving contents of a group. We define this here to store the properties of the group.

```

201 \cs_new:cpn { __groupthm_use_group__#1: }
202 {
203   \hook_gput_code:nnn { __groupthm/prefix } { #1 }
204   {
205     \tl_put_left:Nx \l__groupthm_prefix_tl { #2 }
206   }
207   \hook_gput_code:nnn { __groupthm/suffix } { #1 }
208   {
209     \tl_put_right:Nx \l__groupthm_suffix_tl { #3 }
210   }
211   \hook_gput_code:nnn { __groupthm/mapname } { #1 }
212   {
213     \clist_put_right:Nn \l__groupthm_mapname_clist { #4 }
214   }
215   \hook_gput_code:nnn { __groupthm/thmtools } { #1 }
216   {
217     \clist_put_right:Nn \l__groupthm_thmtools_clist { #5 }
218   }
219 }

```

This ensures the ordering hacks explained before.

```
220 \__groupthm_add_to_group_ordering:n { #1 }
```

This variable will accumulate the parents of this group.

```
\parents_group__\meta{theorem_group}__clist 221 \clist_new:c { g__groupthm_parents_group__#1__clist }
```

This ensures default priorities between groups.

```
222 \hook_gset_rule:nnnn { ?? } { all } { before } { #1 }
223 \hook_gset_rule:nnnn { ?? } { unnumbered } { after } { #1 }
224 \hook_gset_rule:nnnn { ?? } { starred } { after } { #1 }
```

Add defined group to corresponding list

```
225 \clist_gput_left:Nn \g__groupthm_defined_groups_clist { #1 }
226 }
```

(End of definition for `\__groupthm_define_group:nnnnn`, `\__groupthm_use_group__\meta{theorem_group}:`, and `\g__groupthm_parents_group__\meta{theorem_group}__clist`.)

```
\__groupthm_undefine_group:n 227 \__groupthm_undefine_group:n{theorem_group}
```

Undeclares / undefines the given theorem group. This means removing its hook code in the `prefix`, `suffix`, `mapname` and `thmtools` hooks, and removing all relations with other theorem groups globally as well as for each hook individually.

This macro assumes that the group was defined prior to calling.

```
227 \cs_new:Npn \__groupthm_undefine_group:n #1
228 {
229 \tl_log:n { Undefining ~ theorem ~ group ~ '#1' }
230 \cs_undefine:c { __groupthm_use_group__#1: }
```

Remove properties from hooks

```
231 \hook_gremove_code:nn { __groupthm/prefix } { #1 }
232 \hook_gremove_code:nn { __groupthm/suffix } { #1 }
233 \hook_gremove_code:nn { __groupthm/mapname } { #1 }
234 \hook_gremove_code:nn { __groupthm/thmtools } { #1 }
```

Remove theorem group from list of defined theorems

```
235 \clist_gremove_all:Nn \g__groupthm_defined_groups_clist { #1 }
```

Delete the known parents of this group:

```
236 \cs_undefine:c { g__groupthm_parents_group__#1__clist }
```

Now, unset all relations with all defined theorem groups in the internal hooks.

```
237 \__groupthm_hook_gset_rule_foreach:nNnn
238 { ?? }
239 \g__groupthm_defined_groups_clist
240 { unrelated }
241 { #1 }
242 \__groupthm_hook_gset_rule_foreach:nNnn
243 { __groupthm/prefix }
244 \g__groupthm_defined_groups_clist
245 { unrelated }
246 { #1 }
247 \__groupthm_hook_gset_rule_foreach:nNnn
248 { __groupthm/suffix }
249 \g__groupthm_defined_groups_clist
```

```

250     { unrelated }
251     { #1 }
252 \__groupthm_hook_gset_rule_foreach:nNnn
253 { __groupthm/mapname }
254 \g__groupthm_defined_groups_clist
255 { unrelated }
256 { #1 }
257 \__groupthm_hook_gset_rule_foreach:nNnn
258 { __groupthm/thmtools }
259 \g__groupthm_defined_groups_clist
260 { unrelated }
261 { #1 }

```

Also clear all sorting relations

```

262 \__groupthm_remove_from_group_ordering:n { #1 }
263 }

```

(End of definition for \\_\_groupthm\_undefine\_group:n.)

```

\__groupthm_define:nnnNNNn
\__groupthm_define:nnncNNn

```

```

\__groupthm_define:nnnNNNn{<declarator>}{<type>}{<instance>}{<existence
cs>}{<undefine function>}{<define function>}{<definition args>}

```

A general definition macro that is used to implement the **new**, **renew**, **provide** and **declare** definition variants of  $\langle type \rangle$ s. For the purpose of this package,  $\langle type \rangle$  will be one of “theorem group”, “grouped theorem”, “theorem family” and “theorem family options”, but could technically be anything.

The  $\langle instance \rangle$  is the actual thing that will be defined by this function. The  $\langle declarator \rangle$  is one of **new**, **renew**, **provide** and **declare** and indicates the definition behavior: **new** only defines if  $\langle instance \rangle$  does not yet exist and throws an error otherwise, **renew** only (re)defines if  $\langle instance \rangle$  does exist yet and throws an error otherwise, **provides** defines if  $\langle instance \rangle$  does not exist yet, but does nothing otherwise and **declare** defines  $\langle instance \rangle$  in any case, possibly by overwriting the old definition.

The  $\langle existence\ cs \rangle$  is the one whose existence will be checked to determine whether the  $\langle instance \rangle$  already exists.

The  $\langle undefine\ function \rangle$  will be called in case  $\langle instance \rangle$  has to be undefined. It is assumed to have argument type **n** and will be given the  $\langle instance \rangle$  as an argument in this case.

The  $\langle define\ function \rangle$  will be called with the arguments given as  $\langle definition\ args \rangle$  in case no error occurs and  $\langle instance \rangle$  should be defined.

```

264 \cs_new:Npn \__groupthm_define:nnnNNNn #1 #2 #3 #4 #5 #6 #7
265 {

```

We first check for wrong usage of **new**

```

266 \bool_if:nT
267 {
268   \str_if_eq_p:nn { #1 } { new }
269   &&
270   \cs_if_exist_p:N #4
271 }
272 {
273   \tl_log:n { Wrong ~ 'new' ~ definition ~ of ~ #2 ~ '#3' ~ detected. }
274   \msg_error:nnnnn { groupthm } { wrong ~ definition }
275   { #2 } { #3 } { already }
276 }

```

Then check for wrong usage of `renew`

```

277 \bool_if:nT
278 {
279   \str_if_eq_p:nn { #1 } { renew }
280   &&
281   ! \cs_if_exist_p:N #4
282 }
283 {
284   \tl_log:n { Wrong ~ 'renew' ~ definition ~ of ~ #2 ~ '#3' ~ detected. }
285   \msg_error:nnnnn { groupthm } { wrong ~ definition }
286   { #2 } { #3 } { not }
287 }

```

Now, remove the old definition if necessary

```

288 \bool_if:nT
289 {
290   (
291     \str_if_eq_p:nn { #1 } { declare } ||
292     \str_if_eq_p:nn { #1 } { renew }
293   ) &&
294   \cs_if_exist_p:N #4
295 }
296 {
297   \tl_log:n { Removing ~ definition ~ of ~ #2 ~ '#3'. }
298   \tl_log:n { Declarator ~ was ~ #1. }
299   #5 { #3 }
300 }

```

Finally, define new version if not already defined (this check is necessary for the provide version.)

```

301 \bool_if:nTF
302 {
303   \cs_if_exist_p:N #4
304   &&
305   \str_if_eq_p:nn { #1 } { provide }
306 }
307 {
308   \tl_log:n { Providing ~ #2 ~ '#3' ~ skipped: ~ '#3' ~ already ~ defined. }
309 }
310 {
311   \tl_log:n { Defining ~ #2 ~ '#3'. }
312   #6 #7
313 }
314 }
315 \cs_generate_variant:Nn \__groupthm_define:nnnNNNn { n n n c N N n }

```

(End of definition for `\__groupthm_define:nnnNNNn`.)

```

\__groupthm_define_multiple:nnnNNNn {<declarator list>} {<type>} {<existence cs>}
{<undefine function>}{<define function>}{<function name>}{<definition args>}

```

This is a wrapper around the `\__groupthm_define:nnnNNNn` macro. It is intended to wrap the multiple variants of it into a family of macros indicating the variant in their name, e.g. to define `\groupthm_new_group:nnnnn`, `\groupthm_renew_group:nnnnn`, `\groupthm_provide_group:nnnnn` and `\groupthm_declare_group:nnnnn` in the same

way except for their indicated declaration behavior, to avoid repetition when defining these.

The first five arguments work the same way as in `\__groupthm_define:nnnnNNn`, except that *<declarator list>* is now a comma separated list. In *<existence cs>*, `##1` is used to denote the *<instance>* that has currently been called to define.

The *<function name>* is expected to contain `#1`, for which the current *<declarator>* is inserted. These control sequences will then be defined.

The *<definition args>* denote the arguments passed to the *<define function>* and may contain `##1`, `##2`, etc. for the arguments that the *<function name>* received as arguments on expansion.

```

316 \cs_new:Npn \__groupthm_define_multiple:nnnNNnn #1 #2 #3 #4 #5 #6 #7
317 {
318   \cs_set:Npn \__groupthm_map_aux:n ##1
319   {
320     \cs_new:cn { #6 }
321     {
322       \__groupthm_lazy:n
323       {
324         \__groupthm_define:nnncNNn
325         { ##1 }
326         { #2 }
327         { ####1 }
328         { #3 }
329         #4
330         #5
331         { #7 }
332       }
333     }
334   }
335   \clist_map_function:nN { #1 } \__groupthm_map_aux:n
336 }

```

(End of definition for `\__groupthm_define_multiple:nnnNNnn`.)

With these helper functions, we can now easily generate the `new`, `renew`, `provide` and `declare` variants of the theorem group macro:

```

\groupthm_new_group:nnnnn
\groupthm_new_group:nVVVV 337 \__groupthm_define_multiple:nnnNNnn
\groupthm_renew_group:nnnnn 338 { new, renew, provide, declare }
\groupthm_renew_group:nVVVV 339 { theorem group }
\groupthm_provide_group:nnnnn 340 { __groupthm_use_group__##1: }
\groupthm_provide_group:nVVVV 341 \__groupthm_undefine_group:n
\groupthm_declare_group:nnnnn 342 \__groupthm_define_group:nnnnn
\groupthm_declare_group:nVVVV 343 { groupthm_#1_group:nnnnn }
344 { { ##1 } { ##2 } { ##3 } { ##4 } { ##5 } }

```

Finally, generate some extra variants

```

345 \cs_generate_variant:Nn \groupthm_new_group:nnnnn { n V V V V }
346 \cs_generate_variant:Nn \groupthm_renew_group:nnnnn { n V V V V }
347 \cs_generate_variant:Nn \groupthm_provide_group:nnnnn { n V V V V }
348 \cs_generate_variant:Nn \groupthm_declare_group:nnnnn { n V V V V }

```

(End of definition for `\groupthm_new_group:nnnnn` and others. These functions are documented on page 11.)

With the `\__groupthm_set_normalized_keys:nnn` macro at hand, it is also easy to provide key-value interfaces for these commands:

```
\__groupthm_wrap_multiple:nnn
    \__groupthm_wrap_multiple:nnn{<declarator list>}{<function name>}{<code>}
    Defines <function name>, which is assumed to contain \declarator by <code> for
    each declarator in <declarator list>.
```

```
349 \cs_new:Npn \__groupthm_wrap_multiple:nnn #1 #2 #3
350 {
351     \cs_set:Npn \__groupthm_map_aux:n ##1
352     {
353         \cs_new:cn { #2 }
354         {
355             #3
356         }
357     }
358     \clist_map_function:nN { #1 } \__groupthm_map_aux:n
359 }
```

(End of definition for `\__groupthm_wrap_multiple:nnn`.)

```
\groupthm_new_group:nn
\groupthm_renew_group:nn
\groupthm_provide_group:nn
\groupthm_declare_group:nn
    \groupthm_new_group:nn{<key=value list>}{<theorem group>}
360 \__groupthm_wrap_multiple:nnn
361 { new, renew, provide, declare }
362 { groupthm_#1_group:nn }
363 {
364     \__groupthm_set_normalized_keys:nnn { ##1 } { theorem ~ group } { ##2 }
365     \use:c { groupthm_#1_group:nVVVV }
366     { ##2 }
367     \l__groupthm_prefix_tl
368     \l__groupthm_suffix_tl
369     \l__groupthm_mapname_clist
370     \l__groupthm_thmtools_clist
371 }
```

Additional variant

```
372 \cs_generate_variant:Nn \groupthm_new_group:nn { n x }
373 % \end{macro}
374 %
375 %
376 % Finally, we provide \LaTeX2e wrappers as document commands for these.
377 %
378 % \begin{macro}{\__groupthm_new_document_command:Nnn, \__groupthm_new_document_command:cnn}
379 %
380 % Private wrappers around \cs{NewDocumentCommand}.
381 %
382 % \begin{macrocode}
383 \cs_new:Npn \__groupthm_new_document_command:Nnn #1 #2 #3
384 {
385     \NewDocumentCommand { #1 } { #2 } { #3 }
386 }
387 \cs_generate_variant:Nn \__groupthm_new_document_command:Nnn { c n n }
```

(End of definition for `\groupthm_new_group:nn` and others. These functions are documented on page 11.)

```
\__groupthm_wrap_multiple_document:nnnn
    \__groupthm_wrap_multiple_document:nnnn{<declarator list>}{<function name>}{<arg
spec>}{<code>}
```

This is very similar to `\__groupthm_wrap_multiple:nnn`, except that it produces document commands. For this reason, `\declarator` and `\Declarator` are available to refer to the lower and upper-case versions of the current declarator.

```
388 \cs_new:Npn \__groupthm_wrap_multiple_document:nnnn #1 #2 #3 #4
389 {
390   \cs_set:Npn \__groupthm_map_aux:n ##1
391   {
392     \cs_set:Nn \__groupthm_Declarator: { \text_titlecase_first:n { ##1 } }
393     \__groupthm_new_document_command:cnn { #2 } { #3 } { #4 }
394   }
395   \clist_map_function:nN { #1 } \__groupthm_map_aux:n
396 }
```

(End of definition for `\__groupthm_wrap_multiple_document:nnnn`.)

`\NewTheoremGroup`  
`\RenewTheoremGroup`  
`\ProvideTheoremGroup`  
`\DeclareTheoremGroup`

These just wrap the `\groupthm_<declarator>_group:nn` macros.

```
397 \__groupthm_wrap_multiple_document:nnnn
398 { new, renew, provide, declare }
399 { \__groupthm_Declarator: TheoremGroup }
400 { 0{ } m }
401 {
402   \__groupthm_lazy:n
403   {
404     \use:c { groupthm_#1 _group:nn } { ##1 } { ##2 }
405   }
406 }
```

(End of definition for `\NewTheoremGroup` and others. These functions are documented on page 5.)

We also provide the interface for declaring the precedence rules for theorem groups.

`\groupthm_declare_group_rule:nnnn`

```
\groupthm_declare_group_rule:nnnn{<keyname>}{<theorem group1>}
{<relation>}{<theorem group2>}
```

We have to normalize the arguments a little bit, namely replacing `higher` and `lower` with `before` and `after` respectively, and prefix the `<keyname>` with `\__groupthm` in case it is not the general hook “??”.

```
407 \cs_new:Npn \groupthm_declare_group_rule:nnnn #1 #2 #3 #4
408 {
409   \__groupthm_lazy:n
410   {
411     \str_set:Nx \l_tmpa_str { \tl_trim_spaces:n { #3 } }
412     \str_if_eq:VnT \l_tmpa_str { higher }
413     {
414       \str_set:Nn \l_tmpa_str { after }
415     }
416     \str_if_eq:VnT \l_tmpa_str { lower }
417     {
418       \str_set:Nn \l_tmpa_str { before }
419     }
420   }
```

```

420     \str_if_eq:nnTF { #1 } { ?? }
421     {
422         \hook_gset_rule:nnVn {??} {#2} \l_tmpa_str {#4}
423     }
424     {
425         \hook_gset_rule:nnVn { __groupthm / #1 } {#2} \l_tmpa_str {#4}
426     }
427 }
428 }
429 \cs_generate_variant:Nn \groupthm_declare_group_rule:nnnn { n n n x }

```

(End of definition for `\groupthm_declare_group_rule:nnnn`. This function is documented on page 11.)

**`\DeclareTheoremGroupRule`**

```

\DeclareTheoremGroupRule[⟨keyname⟩]
{⟨theorem group1⟩}{⟨relation⟩}{⟨theorem group2⟩}
430 \NewDocumentCommand { \DeclareTheoremGroupRule } { O{??} m m m }
431 {
432     \__groupthm_lazy:n
433     {
434         \groupthm_declare_group_rule:nnnn {#1} {#2} {#3} {#4}
435     }
436 }

```

(End of definition for `\DeclareTheoremGroupRule`. This function is documented on page 6.)

**`\groupthm_add_parent:nn`**

```

\groupthm_add_parent:nn{⟨theorem group1⟩}{⟨theorem group2⟩}
Declares ⟨theorem group2⟩ as a parent of ⟨theorem group1⟩
437 \cs_new:Npn \groupthm_add_parent:nn #1 #2
438 {
439     \__groupthm_lazy:n
440     {
441         \__groupthm_ensure_group_exists:n { #1 }
442         \__groupthm_ensure_group_exists:n { #2 }
443         \clist_gput_left:cn { g__groupthm_parents_group__#1__clist } { #2 }
444     }
445 }
446 \cs_generate_variant:Nn \groupthm_add_parent:nn { n x }

```

(End of definition for `\groupthm_add_parent:nn`. This function is documented on page 11.)

**`\AddTheoremGroupParent`**

```

\AddTheoremGroupParent{⟨theorem group1⟩}{⟨theorem group2⟩}
Document command version of \groupthm_add_parent:nn
447 \NewDocumentCommand { \AddTheoremGroupParent } { m m }
448 {
449     \__groupthmlazy:n
450     {
451         \groupthm_add_parent:nn { #1 } { #2 }
452     }
453 }

```

(End of definition for `\AddTheoremGroupParent`. This function is documented on page 6.)

**`\__groupthm_push_tmpa_seq:n`**

```

\__groupthm_push_tmpa_seq:n{⟨balanced text⟩}
454 %

```

(End of definition for `\__groupthm_push_tmpa_seq:n`.)

`\__groupthm_flatten_groups_hierarchy:nN`

`\__groupthm_flatten_groups_hierarchy:nN{<theorem groups>}{<clist>}`

Expects a comma separated list of *<theorem group>*s. The inheritance relation is flattened, and the set of obtained theorem groups is stored in *<clist>*

```

455 \cs_new:Npn \__groupthm_push_tmpa_seq:n #1
456 {
457   \seq_push:Nn \l_tmpa_seq { #1 }
458 }
459 \cs_new:Npn \__groupthm_flatten_groups_hierarchy:nN #1 #2
460 {
461   \clist_clear:N #2
462   \seq_set_from_clist:Nn \l_tmpa_seq { #1 }
463   \bool_until_do:nn
464   {
465     \seq_if_empty_p:N \l_tmpa_seq
466   }
467   {
468     \seq_pop:NN \l_tmpa_seq \l_tmpa_tl
469     \__groupthm_ensure_group_exists:V \l_tmpa_tl
470     \clist_if_in:NVF #2 \l_tmpa_tl
471     {
472       \clist_put_left:NV #2 \l_tmpa_tl
473       \clist_map_function:cN
474       { g__groupthm_parents_group__ \l_tmpa_tl __clist }
475       \__groupthm_push_tmpa_seq:n
476     }
477   }
478 }

```

(End of definition for `\__groupthm_flatten_groups_hierarchy:nN`.)

`\groupthm_append_to_group:nn`

`\groupthm_append_to_group:nn{<key=value list>}{<theorem group>}`

This works the same as defining a new group, except that we append to the group, overwriting the old group in case of conflicts.

```

479 \cs_new:Npn \groupthm_append_to_group:nn #1 #2
480 {
481   \__groupthm_lazy:n
482   {
483     \groupthm_new_group:nx { #1 } { __append__ \int_use:N \g__groupthm_append_groups_int
484     \groupthm_add_parent:nx { #2 } { __append__ \int_use:N \g__groupthm_append_groups_int
485     \groupthm_declare_group_rule:nnnx
486     { ?? } { #2 } { before } { __append__ \int_use:N \g__groupthm_append_groups_int }
487     \int_gincr:N \g__groupthm_append_groups_int
488   }
489 }
490 % \end{macrocode}
491 % \end{macro}
492 %
493 %
494 %
495 % \begin{macro}{\AppendToTheoremGroup}
496 % \begin{syntax}

```

```

497 %      \cs{AppendToTheoremGroup} \marg{key=value list}\marg{theorem group}
498 %      \end{syntax}
499 %
500 %
501 %
502 %      \begin{macrocode}
503 \NewDocumentCommand { \AppendToTheoremGroup } { 0{ } m }
504 {
505     \__groupthmlazy:n
506     {
507         \groupthm_append_to_group:nn { #1 } { #2 }
508     }
509 }

```

(End of definition for `\groupthm_append_to_group:nn`. This function is documented on page 11.)

## 5.6 Iterating over powersets

For generating the different variants of a theorem family, we need to iterate over over the powerset of some list. This is a collection of hacks that perform exactly this, but these are poorly documented for now.

```

\__groupthm_powerset_clist_foreach:Nn      \__groupthm_powerset_clist_foreach:Nn⟨clist⟩{⟨code⟩}
Executes ⟨code⟩ for each subset of the given clist variable. The value of the (local)
variable is changes throughout the iteration, and is thus available regularly in ⟨code⟩. Its
value is restored at the end of the iteration.

510 \clist_new:N \l__powerset_copied_clist
511 \seq_new:N \l__powerset_saved_seq
512 \cs_generate_variant:Nn \clist_remove_all:Nn { N V }
513 \cs_new:Npn \__powerset_clist_foreach_aux:Nn #1 #2
514 {
515     \clist_if_empty:NTF \l__powerset_copied_clist
516     {
517         #2
518     }
519     {
520         \clist_get:NN \l__powerset_copied_clist \l_tmpa_tl
521         \seq_push:NV \l__powerset_saved_seq \l_tmpa_tl
522         \clist_pop:NN \l__powerset_copied_clist { \l_tmpa_tl }
523         \__powerset_clist_foreach_aux:Nn #1 {#2}
524         \seq_get:NN \l__powerset_saved_seq \l_tmpa_tl
525         \clist_put_left:NV #1 \l_tmpa_tl
526         \__powerset_clist_foreach_aux:Nn #1 {#2}
527         \seq_get:NN \l__powerset_saved_seq \l_tmpa_tl
528         \clist_remove_all:NV #1 \l_tmpa_tl
529         \clist_push:NV \l__powerset_copied_clist \l_tmpa_tl
530         \seq_pop:NN \l__powerset_saved_seq \l_tmpa_tl
531     }
532 }
533 \cs_new:Npn \powerset_clist_foreach:Nn #1 #2
534 {
535     \clist_set_eq:NN \l__powerset_copied_clist #1

```

```

536 \clist_clear:N #1
537 \clist_remove_duplicates:N \l__powerset_copied_clist
538 \__powerset_clist_foreach_aux:Nn #1 {#2}
539 \clist_set_eq:NN #1 \l__powerset_copied_clist
540 }

```

(End of definition for \\_\_groupthm\_powerset\_clist\_foreach:Nn.)

## 5.7 Grouped Theorems

\\_\_groupthm\_ensure\_group\_exists:n      \\_\_groupthm\_ensure\_group\_exists:n(theorem group)  
 \\_\_groupthm\_ensure\_group\_exists:V      Checks if this group exists. If not, produces an error message.

```

541 \cs_new:Npn \__groupthm_ensure_group_exists:n #1
542 {
543   \cs_if_exist:cF { __groupthm_use_group__#1: }
544   {
545     \msg_error:nnn { groupthm } { unknown ~ group } { #1 }
546   }
547 }
548 \cs_generate_variant:Nn \__groupthm_ensure_group_exists:n { V }

```

(End of definition for \\_\_groupthm\_ensure\_group\_exists:n.)

\\_\_groupthm\_use\_group:n      \\_\_groupthm\_use\_group:n(theorem group)  
 Uses this theorem group, i.e. applies its definition by writing to the internal hooks.  
 A proper error message is emitted if the group is not defined.

```

549 \cs_new:Npn \__groupthm_use_group:n #1
550 {
551   \__groupthm_ensure_group_exists:n { #1 }
552   \use:c { __groupthm_use_group__#1: }
553 }

```

(End of definition for \\_\_groupthm\_use\_group:n.)

\\_\_groupthm\_use\_function\_on\_name:n      \\_\_groupthm\_use\_function\_on\_name:n(function)  
 The  $\langle function \rangle$  is expected to be of type :n, This applies the function to the \l\_\_-  
 groupthm\_name\_tl

```

554 \cs_new:Npn \__groupthm_use_function_on_name:n #1
555 {
556   \tl_set:Nx \l__groupthm_name_tl
557   {
558     #1 { \tl_use:N \l__groupthm_name_tl }
559   }
560 }

```

(End of definition for \\_\_groupthm\_use\_function\_on\_name:n.)

\\_\_groupthm\_define\_theorem:nnnn      \\_\_groupthm\_define\_theorem:nnnn{environment name}  
 {groups clist}{theorem name}{thmtools keys}  
 This is the internal backend that declares a grouped theorem by retrieving all the  
 group properties and issuing a (single) \declaretheorem command of thmtools.

```

561 \cs_new:Npn \__groupthm_define_theorem:nnnn #1 #2 #3 #4
562 {

```

First, set local variables to default values and store current name.

```

563 \tl_clear:N \l__groupthm_prefix_tl
564 \tl_set:Nn \l__groupthm_name_tl { #3 }
565 \tl_clear:N \l__groupthm_suffix_tl
566 \clist_clear:N \l__groupthm_mapname_clist
567 \clist_clear:N \l__groupthm_thmtools_clist

```

Clear all hooks

```

568 \hook_gremove_code:nn { __groupthm/prefix }{*}
569 \hook_gremove_code:nn { __groupthm/suffix }{*}
570 \hook_gremove_code:nn { __groupthm/mapname }{*}
571 \hook_gremove_code:nn { __groupthm/thmtools }{*}

```

Now, retrieve the group properties, by writing these into the hooks

```

572 \__groupthm_flatten_groups_hierarchy:nN { #2, all } \l__groupthm_group_clist
573 \clist_map_function:NN \l__groupthm_group_clist \__groupthm_use_group:n
574 \tl_log:x { Flattened ~ groups ~ '#2' ~ to ~ '\clist_use:Nn \l__groupthm_group_clist {,}'
575           defining ~ theorem ~ '#1' }

```

Execute the hooks, so that local variables will get modified according to the groups and in the order that were specified for the hooks.

```

576 \hook_use:n { __groupthm/prefix }
577 \hook_use:n { __groupthm/suffix }
578 \hook_use:n { __groupthm/mapname }
579 \hook_use:n { __groupthm/thmtools }

```

We are left with dealing with the obtained data. We first map on the name, before appending to it.

```

580 \clist_map_function:NN \l__groupthm_mapname_clist \map_use_on_name:n

```

We now glue the name together of its three parts, and pass this key directly to the thmtools arguments

```

581 \tl_set:Nn \l_tmpa_tl { name = }
582 \tl_put_right:NV \l_tmpa_tl \l__groupthm_prefix_tl
583 \tl_put_right:NV \l_tmpa_tl \l__groupthm_name_tl
584 \tl_put_right:NV \l_tmpa_tl \l__groupthm_suffix_tl
585 \clist_put_right:NV \l__groupthm_thmtools_clist \l_tmpa_tl

```

Finally, apply the additional thmtools keys for this specific theorem. Putting them last will overwrite keys that were given by the groups.

```

586 \clist_put_right:Nn \l__groupthm_thmtools_clist { #4 }

```

We can now pass our list to thmtools, declaring the theorem.

```

587 \__groupthm_thmtools_declare_theorem:Vn
588 \l__groupthm_thmtools_clist
589 { #1 }
590 }

```

(End of definition for \\_\_groupthm\_define\_theorem:nnnn.)

As usual, we provide a `new` and a `provide` variant wrapped around this that do proper error checking.

```

\groupthm_new_theorem:nnnn \groupthm_new_theorem:nnnn{environment name}
\groupthm_new_theorem:nVVV {groups clist}{theorem name}{thmtools keys}
\groupthm_new_theorem:xVnn
591 \__groupthm_define_multiple:nnnNNnn
592 { new, provide }
\groupthm_provide_theorem:nnnn
\groupthm_provide_theorem:nVVV
\groupthm_provide_theorem:xVnn

```

```

593 { grouped ~ theorem }
594 { ##1 }
595 \__groupthm_error:
596 \__groupthm_define_theorem:nnnn
597 { groupthm_#1_theorem:nnnn }
598 { { ##1 } { ##2 } { ##3 } { ##4 } }
599 \cs_generate_variant:Nn \groupthm_new_theorem:nnnn { n V V V }
600 \cs_generate_variant:Nn \groupthm_provide_theorem:nnnn { n V V V }

```

We need this extra variant here for the generation of theorem families later:

```

601 \cs_generate_variant:Nn \groupthm_new_theorem:nnnn { x V n n }
602 \cs_generate_variant:Nn \groupthm_provide_theorem:nnnn { x V n n }

```

(End of definition for `\groupthm_new_theorem:nnnn` and `\groupthm_provide_theorem:nnnn`. These functions are documented on page 11.)

```

\groupthm_new_theorem:nnn    \groupthm_new_theorem:nnn{<key=value list>}{<grouped theorem>}
\groupthm_provide_theorem:nnn {<bool>}

```

The third argument indicates whether the generated theorem(s) will be added to the unnumbered group

```

603 \__groupthm_wrap_multiple:nnn
604 { new, provide }
605 { groupthm_#1_theorem:nnn }
606 {
607   \__groupthm_lazy:n
608   {
609     \__groupthm_set_normalized_keys:nnn { ##1 } { grouped ~ theorem } { ##2 }
610     \bool_if:nT { ##3 }
611     {
612       \clist_put_left:Nn \l__groupthm_group_clist { unnumbered }
613     }
614     \use:c { groupthm_#1_theorem:nVVV }
615     { ##2 }
616     \l__groupthm_group_clist
617     \l__groupthm_name_tl
618     \l__groupthm_thmtools_clist
619     \bool_if:NT \l__groupthm_starred_version_bool
620     {
621       \clist_put_left:Nn \l__groupthm_group_clist { starred }
622       \use:c { groupthm_#1_theorem:nVVV }
623       { ##2* }
624       \l__groupthm_group_clist
625       \l__groupthm_name_tl
626       \l__groupthm_thmtools_clist
627     }
628   }
629 }

```

(End of definition for `\groupthm_new_theorem:nnn` and `\groupthm_provide_theorem:nnn`. These functions are documented on page 11.)

On top of these, we can provide the shorter versions that will generate two theorems each, one with and one without a “\*” in its environment name

Now, we can wrap these into document commands

```

\NewGroupedTheorem      \NewGroupedTheorem[⟨key=value list⟩]{⟨theorem name⟩}
\NewGroupedTheorem*
\ProvideGroupedTheorem
\ProvideGroupedTheorem*
630 \__groupthm_wrap_multiple_document:nnnn
631 { new, provide }
632 { \__groupthm_Declarator: GroupedTheorem }
633 { s 0{} m }
634 {
635   \__groupthm_lazy:n
636   {
637     \use:c { groupthm_#1_theorem:nnn }
638     { ##2 }
639     { ##3 }
640     { ##1 }
641   }
642 }

```

(End of definition for `\NewGroupedTheorem` and others. These functions are documented on page 8.)

## 5.8 Theorem families

We now want to implement the generation of theorem families and their corresponding options. As a backend, we use the following auxiliary function

```

\__groupthm_define_family:nnnnn{⟨family name⟩}{⟨groups clist⟩}
{⟨name⟩}{⟨thmtools clist⟩}{⟨extra groups clist⟩}

```

This will generate a new grouped theorem for each union of a subset of  $\langle groups\ clist \rangle$  and the extra set  $\langle extra\ groups\ clist \rangle$ , with the given properties, that is the  $\langle nam \rangle$  and  $\langle thmtools\ clist \rangle$  will be passed to the grouped theorem. The  $\langle theorem\ name \rangle$  of the grouped theorem will be an internal name that contains the  $\langle family\ name \rangle$  and the list of groups of this variant, that will be generated in a unique manner to later retrieve the generated theorems when parsing theorem families.

```

643 \cs_new:Npn \__groupthm_define_family:nnnnn #1 #2 #3 #4 #5
644 {

```

Make a local copy of the  $\langle groups\ clist \rangle$  argument, and iterate over its powerset

```

645   \clist_set:Nn \l_tmpa_clist { #2 }
646   \powerset_clist_foreach:Nn \l_tmpa_clist
647   {

```

We read out the current value of the list, and append the extra groups This ensures that now  $\l\_groupthm\_group\_clist$  iterates over the proper subsets

```

648     \clist_set_eq:Nn \l__groupthm_group_clist \l_tmpa_clist
649     \clist_put_right:Nn \l__groupthm_group_clist { #5 }

```

This sorting is necessary so that for each theorem family and set of groups, the generated name will be unique:

```

650     \__groupthm_sort_group_names:

```

Now just declare the grouped theorem, passing the corresponding arguments

```

651     \use:c{groupthm_new_theorem:xVnn}
652     {_#1__groups_\clist_use:Nn \l__groupthm_group_clist {_}}
653     \l__groupthm_group_clist
654     { #3 }
655     { #4 }
656   }

```

We save the set of variants we generated for later error checking:

```

657 \clist_new:c { __groupthm_family__#1__group_clist }
658 \clist_set_eq:cN {__groupthm_family__#1__group_clist } \l_tmpa_clist
659 \clist_new:c { __groupthm_family__#1__always_group_clist }
660 \clist_set:cn {__groupthm_family__#1__always_group_clist } { #5 }
661 }

```

(End of definition for `\__groupthm_define_family:nnnnn`.)

```

\groupthm_new_family:nnnnn \groupthm_new_family:nnnnn{<theorem family>}
\groupthm_new_family:nVVVV {<groups1>}{<name>}{<thmtools clist>}{<groups2>}}
662 \__groupthm_define_multiple:nnnNNnn
663 { new, provide }
664 { theorem ~ family }
665 { __groupthm_family__#1__group_clist }
666 \__groupthm_error:
667 \__groupthm_define_family:nnnnn
668 { groupthm_#1_family:nnnnn }
669 { { ##1 } { ##2 } { ##3 } { ##4 } { ##5 } }
670 \cs_generate_variant:Nn \groupthm_new_family:nnnnn { n V V V V }
671 \cs_generate_variant:Nn \groupthm_provide_family:nnnnn { n V V V V }

```

(End of definition for `\groupthm_new_family:nnnnn`. This function is documented on page 12.)

```

\groupthm_new_family:nnn \groupthm_new_family:nnn{<key=value list>}{<theorem family>}{<bool>}
\groupthm_provide_family:nnn The third argument indicates whether the generated theorem(s) will all be added to
the unnumbered group.

```

```

672 \__groupthm_wrap_multiple:nnn
673 { new, provide }
674 { groupthm_#1_family:nnn }
675 {
676 \__groupthm_lazy:n
677 {
678 \__groupthm_set_normalized_keys:nnn { ##1 } { theorem ~ family } { ##2 }
679 \bool_if:nTF { ##3 }
680 {
681 \clist_set:Nn \l_tmpa_clist { unnumbered }
682 }
683 {
684 \clist_clear:N \l_tmpa_clist
685 }
686 \bool_if:NT \l__groupthm_starred_version_bool
687 {
688 \clist_put_left:Nn \l__groupthm_group_clist { starred }
689 }
690 \use:c { groupthm_#1_family:nVVVV }
691 { ##2 }
692 \l__groupthm_group_clist
693 \l__groupthm_name_tl
694 \l__groupthm_thmtools_clist
695 \l_tmpa_clist
696 }
697 }

```

(End of definition for `\groupthm_new_family:nnn` and `\groupthm_provide_family:nnn`. These functions are documented on page 12.)

Finally, we can provide document commands that make these available.

```

\NewGroupedTheoremFamily      \NewGroupedTheoremFamily[⟨key=value list⟩]{⟨family name⟩}
\NewGroupedTheoremFamily*
\ProvideGroupedTheoremFamily  698 \__groupthm_wrap_multiple_document:nnnn
                                699 { new, provide }
                                700 { \__groupthm_Declarator: GroupedTheoremFamily }
                                701 { s 0{} m }
                                702 {
                                703   \__groupthm_lazy:n
                                704   {
                                705     \use:c { groupthm_#1_family:nnn }
                                706     { ##2 }
                                707     { ##3 }
                                708     { ##1 }
                                709   }
                                710 }

```

(End of definition for `\NewGroupedTheoremFamily` and others. These functions are documented on page 9.)

## 5.9 Theorem family options

```

\groupthm_add_theorem_to_group:n      \groupthm_add_theorem_to_group:n{⟨theorem group⟩}
711 \cs_new:Npn \groupthm_add_theorem_to_group:n #1
712 {

```

As mentioned earlier, this bool will indicate whether we are executing a *⟨selection body⟩* from some family options. If used outside, we emit an error message.

```

713   \__groupthm_lazy:n
714   {
715     \bool_if:NTF \l__groupthm_in_family_options_environment_bool
716     {
717       \clist_put_left:Nn \l__groupthm_group_clist { #1 }
718     }
719     {
720       \msg_error:nn { groupthm } { misuse ~ add ~ theorem ~ to ~ group }
721     }
722   }
723 }

```

(End of definition for `\groupthm_add_theorem_to_group:n`. This function is documented on page 12.)

```

\AddTheoremToGroup      \AddTheoremToGroup{⟨theorem group⟩}
724 \NewDocumentCommand { \AddTheoremToGroup } { m }
725 {
726   \groupthm_add_theorem_to_group:n { #1 }
727 }

```

(End of definition for `\AddTheoremToGroup`. This function is documented on page 9.)

`\__groupthm_define_family_options:nnnn`

```
\__groupthm_define_family_options:nnnn{<theorem family>}
{<argument specification>}{<selection body>}{<extra groups>}
```

This declares a new theorem variant option parser, i.e. introduces the environment `<theorem family>` with signature `{<argument specification>}`.

The `<selection body>` will be executed and selects some groups the environment shall have. The `<extra groups>` will be added regardless of the arguments given to the `<theorem family>`. The `<declaring backend>` is one of `New`, `Renew`, `Provide` and `Declare` and is given to the `DocumentEnvironment` command from `xparse`.

```
728 \cs_new:Npn \__groupthm_define_family_options:nnnn #1 #2 #3 #4
729 {
730   \DeclareDocumentEnvironment
731     { #1 }
732     { #2 }
733     {
```

We can now clear the group list and execute the `<selection body>` that populates this list again. Additionally, we add the groups that should always be present and activate the `\AddTheoremToGroup` macro by setting the bool.

```
734   \clist_clear:N \l__groupthm_group_clist
735   \bool_set_true:N \l__groupthm_in_family_options_environment_bool
736   #3
737   \bool_set_false:N \l__groupthm_in_family_options_environment_bool
738   \clist_put_right:Nn \l__groupthm_group_clist { #4 }
```

We now got the list of groups parsed. We sort this and start the corresponding environment that has been generated by a `\NewGroupedTheoremFamily` command or similar.

```
739   \__groupthm_sort_group_names:
740   \cs_if_exist:cTF { __#1__groups_ \clist_use:Nn \l__groupthm_group_clist { _ } }
741   {
742     \begin { __#1__groups_ \clist_use:Nn \l__groupthm_group_clist { _ } }
743   }
744   {
745     \msg_error:nnxx { groupthm } { undefined ~ theorem ~ variant }
746     { #1 }
747     { \clist_use:Nnnn \l__groupthm_group_clist { ~ and ~ } { , ~ } { , ~ and ~ } }
748   }
749   }
750   {
```

At the end of the environment, we have to do the same parsing again.

```
751   \clist_clear:N \l__groupthm_group_clist
752   \bool_set_true:N \l__groupthm_in_family_options_environment_bool
753   #3
754   \bool_set_false:N \l__groupthm_in_family_options_environment_bool
755   \clist_put_right:Nn \l__groupthm_group_clist { #4 }
```

End the corresponding environment.

```
756   \__groupthm_sort_group_names:
757   \end { __#1__groups_ \clist_use:Nn \l__groupthm_group_clist { _ } }
758 }
```

Cache this definition for re-runs of `LATEX`.

```
759 \__groupthm_cache_slow:n
760 {
```

```

761         \csname __groupthm_define_family_options:nnnn \endcsname { #1 } { #2 } { #3 } { #4 }
762     }
763 }

```

(End of definition for `\__groupthm_define_family_options:nnnn`.)

All other macros are now essentially wrappers around this aux macro, passing different *<extra groups>* to them

```

\groupthm_new_family_options:nnnn      \groupthm_new_family_options:nnnn{<theorem family>}
\groupthm_renew_family_options:nnnn    {<signature>}{<selection body>}{<groups>}
\groupthm_provide_family_options:nnnn
\groupthm_declare_family_options:nnnn

764 \__groupthm_define_multiple:nnnNnn
765 { new, renew, provide, declare }
766 { theorem ~ family ~ options }
767 { ##1 }
768 \use_none:n
769 \__groupthm_define_family_options:nnnn
770 { groupthm_#1_family_options:nnnn }
771 { { ##1 } { ##2 } { ##3 } { ##4 } }
772 \cs_generate_variant:Nn \groupthm_new_family_options:nnnn { n n n V }
773 \cs_generate_variant:Nn \groupthm_renew_family_options:nnnn { n n n V }
774 \cs_generate_variant:Nn \groupthm_provide_family_options:nnnn { n n n V }
775 \cs_generate_variant:Nn \groupthm_declare_family_options:nnnn { n n n V }

```

(End of definition for `\groupthm_new_family_options:nnnn` and others. These functions are documented on page 12.)

It remains to wrap these into document commands

```

\NewGrouppedTheoremFamilyOptions      \NewGrouppedTheoremFamilyOptions{<family name>}{<signature>}
\NewGrouppedTheoremFamilyOptions*     {<selection body>}
\RenewGrouppedTheoremFamilyOptions
\RenewGrouppedTheoremFamilyOptions*
\ProvideGrouppedTheoremFamilyOptions
\ProvideGrouppedTheoremFamilyOptions*
\DeclareGrouppedTheoremFamilyOptions
\DeclareGrouppedTheoremFamilyOptions*

776 \__groupthm_wrap_multiple_document:nnnn
777 { new, renew, provide, declare }
778 { \__groupthm_Declarator: GrouppedTheoremFamilyOptions }
779 { s O{} m m m }
780 {
781     \__groupthm_lazy:n
782     {
783         \keys_set:nn { groupthm / theorem ~ family ~ options } { starred ~ version }
784         \keys_set:nn { groupthm / theorem ~ family ~ options } { ##2 }
785         \bool_if:nTF { ##1 }
786         {
787             \clist_set:Nn \l_tmpa_clist { unnumbered }
788         }
789         {
790             \clist_clear:N \l_tmpa_clist
791         }
792         \use:c { groupthm_#1_family_options:nnnV }
793         { ##3 }
794         { ##4 }
795         { ##5 }
796         \l_tmpa_clist
797         \bool_if:NT \l__groupthm_starred_version_bool
798         {
799             \use:c { groupthm_#1_family_options:nnnV }
800             { ##3* }

```

```

801         { ##4 }
802         {
803             ##5
804             \groupthm_add_theorem_to_group:n { starred }
805         }
806         \l_tmpa_clist
807     }
808 }
809 }

```

(End of definition for `\NewGroupedTheoremFamilyOptions` and others. These functions are documented on page 10.)

## 5.10 Caching

We differentiate between caching and non-caching mode here. First, we provide some wrappers for writing to the aux file:

```

810 \bool_if:NTF \g__groupthm_cache_bool
811 {

```

`\__groupthm_write_auxout:n` Immediate write to aux file.

```

\__groupthm_write_auxout:x
812     \cs_new:Npn \__groupthm_write_auxout:n
813     {
814         \iow_now:cn { @auxout }
815     }
816     \cs_generate_variant:Nn \__groupthm_write_auxout:n { x }

```

(End of definition for `\__groupthm_write_auxout:n`.)

`\__groupthm_dump_auxfile:n` Dump to auxfile at end of document. This is used to dump to the aux file at any point of the document without messing with preamble / document treatment etc.

```

\__groupthm_dump_auxfile:x
817     \cs_new:Npn \__groupthm_dump_auxfile:n
818     {
819         \clist_gput_right:Nn \g__groupthm_dump_auxfile_clist
820     }
821     \cs_generate_variant:Nn \__groupthm_dump_auxfile:n { x }

```

(End of definition for `\__groupthm_dump_auxfile:n`.)

`\__groupthm_cache_auxfile:n` This also pipes data through the auxfile, but will cache it to make it available in the next run.

```

\__groupthm_cache_auxfile:V
822     \cs_new:Npn \__groupthm_cache_auxfile:n
823     {
824         \clist_gput_right:Nn \g__groupthm_dump_cache_clist
825     }
826     \cs_generate_variant:Nn \__groupthm_cache_auxfile:n { V }

```

(End of definition for `\__groupthm_cache_auxfile:n`.)

This handles the dumping to the auxfile at the end of the document:

```

827     \hook_gput_code:nnn { enddocument } { groupthm }
828     {
829     <benchmark>
830         \__groupthm_benchmark_once:nn

```

```

831         {
832             Benchmarking ~ writing ~ to ~ auxfile:
833         }
834     {
835 </benchmark>

```

First, perform the regular dump:

```

836         \clist_map_function:NN \g__groupthm_dump_auxfile_clist
837         \__groupthm_write_auxout:n

```

Now, we want to dump the cached clist, but wrap it into the the corresponding macro call for storing it in `\g__groupthm_lazy_auxfile_tl` on re-reading the auxfile. For this, we have to put two single braces into the auxfile separately

```

838         \__groupthm_write_auxout:n
839         {
840             \csname tl_gput_right:cn \endcsname
841             {
842                 g__groupthm_lazy_auxfile_tl
843             }
844         }
845         \__groupthm_write_auxout:x { \str_use:N \c_left_brace_str }

```

Now, dump the list

```

846         \clist_map_function:NN \g__groupthm_dump_cache_clist
847         \__groupthm_write_auxout:n
848         \__groupthm_write_auxout:x { \str_use:N \c_right_brace_str }
849 <*benchmark>
850     }
851 </benchmark>
852 }

```

This makes sure that we restore the cache version on the next run of LaTeX

```

853     \__groupthm_dump_auxfile:x
854     {
855         \ExplSyntaxOn
856         \int_gset:Nn \exp_not:N \g__groupthm_cache_version_aux_int
857         {
858             \int_use:N \g__groupthm_cache_version_document_int
859         }
860         \ExplSyntaxOff
861     }

```

We can now provide the two main macros responsible for the caching feature.

```

\__groupthm_lazy:n         \__groupthm_lazy:n{<code>}
Lazy execution will just append to the corresponding token list

862     \cs_new:Npn \__groupthm_lazy:n
863     {
864         \tl_gput_right:Nn \g__groupthm_lazy_document_tl
865     }

```

(End of definition for `\__groupthm_lazy:n`.)

`\__groupthm_cache:n`

`\__groupthm_cache:n{<code>}`

We have to replace parameter-tokens of category code 6 with regular “#” tokens of category code 12 so that we don’t double them when writing to the aux file. See <https://tex.stackexchange.com/questions/632294/writing-nested-argument-specifier-to-aux-file> for details.

866 `\cs_set_eq:NN \__groupthm_cache:n \__groupthm_cache_auxfile:n`

(End of definition for `\__groupthm_cache:n`.)

`\__groupthm_cache_slow:n`

867 `\cs_new:Npn \__groupthm_cache_slow:n #1`  
868 `{`  
869 `\tl_set:Nn \l_tmpa_tl`  
870 `{`  
871 `{ #1 }`  
872 `}`  
873 `\regex_replace_all:nnN { \cP\# } { \cO\# } \l_tmpa_tl`  
874 `\__groupthm_cache_auxfile:V \l_tmpa_tl`  
875 `}`

(End of definition for `\__groupthm_cache_slow:n`.)

Finally, at the beginning of the document, call the correct code for execution

876 `\hook_gput_code:nnn { begindocument } { groupthm }`  
877 `{`  
878 `\cs_set_eq:NN \__groupthm_lazy:n \use:n`  
879 `\int_compare:nNnTF`  
880 `\g__groupthm_cache_version_aux_int < \g__groupthm_cache_version_document_int`  
881 `{`  
882 `< *benchmark >`  
883 `\__groupthm_benchmark_once:nn`  
884 `{`  
885 `Benchmarking ~ declarations ~ of ~ theorems ~ from ~ document:`  
886 `}`  
887 `{`  
888 `< /benchmark >`  
889 `\tl_use:N \g__groupthm_lazy_document_tl`  
890 `< *benchmark >`  
891 `}`  
892 `< /benchmark >`  
893 `}`  
894 `{`  
895 `< *benchmark >`  
896 `\__groupthm_benchmark_once:nn`  
897 `{`  
898 `Benchmarking ~ declarations ~ of ~ theorems ~ from ~ aux ~ file:`  
899 `}`  
900 `{`  
901 `< /benchmark >`  
902 `\tl_use:N \g__groupthm_lazy_auxfile_tl`  
903 `< *benchmark >`  
904 `}`  
905 `< /benchmark >`  
906 `}`  
907 `}`

It remains to handle the non-caching case, in which we only have to define `\__groupthm_cache:n` and `\__groupthm_lazy:n` to be simple wrappers.

```
908 }
909 {
```

```
\__groupthm_cache:n
```

```
910 \cs_set_eq:NN \__groupthm_cache:n \use_none:n
```

*(End of definition for \\_\_groupthm\_cache:n.)*

```
\__groupthm_cache_slow:n
```

```
911 \cs_set_eq:NN \__groupthm_cache_slow:n \use_none:n
```

*(End of definition for \\_\_groupthm\_cache\_slow:n.)*

```
\__groupthm_lazy:n
```

```
912 \cs_set_eq:NN \__groupthm_lazy:n \use:n
```

*(End of definition for \\_\_groupthm\_lazy:n.)*

This ends the non-caching case.

```
913 }
```

We also provide the three default groups:

```

all
starred
unnumbered
914 \groupthm_new_group:nnnnn { all      } { } { } { } { } { }
915 \groupthm_new_group:nnnnn { starred  } { } { } { } { } { } { numbered = no }
916 \groupthm_new_group:nnnnn { unnumbered } { } { } { } { } { } { numbered = no }
```

*(End of definition for all, starred, and unnumbered. These variables are documented on page 7.)*

## 5.11 Benchmarking

Just some utility macro for benchmarking some parts of the package.

```
\__groupthm_benchmark:nn
```

```
917 <*benchmark>
918 \cs_new:Npn \__groupthm_benchmark_once:nn #1
919 {
920   \iow_term:n { [groupthm] ~ #1 }
921   \iow_log:n { [groupthm] ~ #1 }
922   \benchmark_once:n
923 }
924 </benchmark>
```

*(End of definition for \\_\_groupthm\_benchmark:nn.)*

```
925 </package>
```

# Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	
\#	873
A	
\AddTheoremGroupParent	6, 11, 26, <u>447</u>
\AddTheoremToGroup	9, 10, 12, 14, 15, 34, 35, <u>724</u>
all	7, <u>914</u>
\AppendToTheoremGroup	7, 11, 495, 503
B	
\begin	378, 382, 495, 496, 502, 742
benchmark commands:	
\benchmark_once:n	922
bool commands:	
\bool_if:NTF	619, 686, 715, 797, 810
\bool_if:nTF	266, 277, 288, 301, 610, 679, 785
\bool_new:N	61, 75, 77
\bool_set_eq:NN	161
\bool_set_false:N	737, 754
\bool_set_true:N	735, 752
\bool_until_do:nn	463
C	
clist commands:	
\clist_clear:N	196, 461, 536, 566, 567, 684, 734, 751, 790
\clist_get:NN	520
\clist_gput_left:Nn	225, 443
\clist_gput_right:Nn	819, 824
\clist_gremove_all:Nn	235
\clist_if_empty:NTF	515
\clist_if_in:NnTF	470
\clist_map_function:NN	25, 195, 473, 573, 580, 836, 846
\clist_map_function:nN	335, 358, 395
\clist_new:N	65, 66, 67, 71, 72, 73, 74, 83, 84, 221, 510, 657, 659
\clist_pop:NN	522
\clist_push:Nn	529
\clist_put_left:Nn	189, 472, 525, 612, 621, 688, 717
\clist_put_right:Nn	213, 217, 585, 586, 649, 738, 755
\clist_remove_all:Nn	512, 528
\clist_remove_duplicates:N	537
\clist_set:Nn	645, 660, 681, 787
\clist_set_eq:NN	164, 165, 166, 535, 539, 648, 658
\clist_use:Nn	574, 652, 740, 742, 757
\clist_use:Nnnn	747
\l_tmpa_clist	645, 646, 648, 658, 681, 684, 695, 787, 790, 796, 806
\cO	873
\cP	873
\cs	380, 497
cs commands:	
\cs_generate_variant:Nn	18, 60, 315, 345, 346, 347, 348, 372, 387, 429, 446, 512, 548, 599, 600, 601, 602, 670, 671, 772, 773, 774, 775, 816, 821, 826
\cs_if_exist:NTF	543, 740
\cs_if_exist_p:N	270, 281, 294, 303
\cs_new:Nn	320, 353
\cs_new:Npn	9, 19, 142, 168, 176, 184, 192, 199, 201, 227, 264, 316, 349, 383, 388, 407, 437, 455, 459, 479, 513, 533, 541, 549, 554, 561, 643, 711, 728, 812, 817, 822, 862, 867, 918
\cs_set:Nn	392
\cs_set:Npn	21, 318, 351, 390
\cs_set_eq:NN	866, 878, 910, 911, 912
\cs_undefine:N	230, 236
\csname	15, 761, 840
D	
\Declarator	25
\declarator	24, 25
\DeclareDocumentEnvironment	730
\DeclareFoo	4
\DeclareGroupedTheoremFamilyOptions	10, <u>776</u>
\DeclareGroupedTheoremFamilyOptions*	10, <u>776</u>
\DeclareHookRule	6
\declaretheorem	4, 5, 12, 29, 12
\DeclareTheoremGroup	5, 11, <u>397</u>
\DeclareTheoremGroupRule	6, 11, 26, <u>430</u>
E	
\end	373, 490, 491, 498, 757
\endcsname	15, 761, 840
exp commands:	
\exp_not:N	856

\ExplSyntaxOff	860	\__groupthm_add_to_sort_hook:n	18, 184, 184, 195
\ExplSyntaxOn	855	\g_groupthm_append_groups_int	76, 483, 484, 486, 487
<b>F</b>			
\Foo	4	\__groupthm_benchmark:nn	917
fun commands:			
\fun:n	5	\__groupthm_benchmark_once:nn	830, 883, 896, 918
<b>G</b>			
groupthm commands:			
\groupthm<declarator>_group:nn	25	\__groupthm_cache:n	39, 40, 13, 866, 866, 910, 910
\groupthm_add_parent:nn	11, 26, 437, 437, 446, 451, 484	\__groupthm_cache_auxfile:n	822, 822, 826, 866, 874
\groupthm_add_theorem_to_group:n	12, 34, 711, 711, 726, 804	\g_groupthm_cache_bool	77, 87, 810
\groupthm_append_to_group:nn	11, 27, 479, 479, 507	\__groupthm_cache_slow:n	759, 867, 867, 911, 911
\groupthm_declare_family_		\g_groupthm_cache_version_aux_	78, 856, 880
options:nnnn	12, 764, 775	int	78, 89, 858, 880
\groupthm_declare_group:nn	11, 360	\g_groupthm_cache_version_	
\groupthm_declare_group:nnnn	11, 22, 337, 348	document_int	78, 89, 858, 880
\groupthm_declare_group_rule:nnnn	11, 25, 407, 407, 429, 434, 485	\__groupthm_Declarator:	392, 399, 632, 700, 778
\groupthm_new_family:nnn	12, 33, 672	\__groupthm_define:nnnNNNn	21, 22, 264, 264, 315, 324
\groupthm_new_family:nnnnn	12, 33, 662, 670	\__groupthm_define:nnnnNNN	23
\groupthm_new_family_options:nnnn	12, 36, 764, 772	\__groupthm_define_family:nnnnn	32, 643, 643, 667
\groupthm_new_group:nn	11, 24, 360, 372, 483	\__groupthm_define_family_	
\groupthm_new_group:nnnnn	11, 22, 337, 345, 914, 915, 916	options:nnnn	35, 728, 728, 769
\groupthm_new_theorem:nnn	11, 31, 603	\__groupthm_define_group:nnnnn	19, 199, 199, 342
\groupthm_new_theorem:nnnn	11, 30, 591, 599, 601	\__groupthm_define_multiple:nnnNNnn	22, 316, 316, 337, 591, 662, 764
\groupthm_provide_family:nnn	12, 672	\__groupthm_define_theorem:nnnn	29, 561, 561, 596
\groupthm_provide_family:nnnnn	12, 671	\g_groupthm_defined_groups_	
\groupthm_provide_family_		clist	74, 172, 180, 225, 235, 239, 244, 249, 254, 259
options:nnnn	12, 764, 774	\__groupthm_dump_auxfile:n	817, 817, 821, 853
\groupthm_provide_group:nn	11, 360	\g_groupthm_dump_auxfile_clist	83, 819, 836
\groupthm_provide_group:nnnnn	11, 22, 337, 347	\g_groupthm_dump_cache_clist	15, 83, 824, 846
\groupthm_provide_theorem:nnn	11, 603	\__groupthm_ensure_group_	
\groupthm_provide_theorem:nnnn	11, 591, 600, 602	exists:n	29, 441, 442, 469, 541, 541, 548, 551
\groupthm_renew_family_options:nnnn	12, 764, 773	\__groupthm_error:	595, 666
\groupthm_renew_group:nn	11, 360	\__groupthm_flatten_groups_	
\groupthm_renew_group:nnnnn	11, 22, 337, 346	hierarchy:nN	27, 455, 459, 572
groupthm internal commands:			
\__groupthm_add_to_group_		\l_groupthm_group_clist	18, 19, 32, 68, 164, 189, 195, 196, 572, 573, 574, 612, 616, 621, 624, 648, 649, 652, 653, 688, 692, 717, 734, 738, 740, 742, 747, 751, 755, 757
ordering:n	18, 168, 168, 220		

<code>\__groupthm_hook_gset_rule_-</code>		<code>\l__groupthm_thmtools_clist</code>	....
<code>foreach:nNnn</code>	13, 19, 19, 170, 178, 237, 242, 247, 252, 257	.....	68, 166, 217, 370, 567, 585, 586, 588, 618, 626, 694
<code>\l__groupthm_in_family_options_-</code>		<code>\__groupthm_thmtools_declare_-</code>	
<code>environment_bool</code>	..... 75, 715, 735, 737, 752, 754	<code>theorem:nn</code>	..... 12, 9, 9, 18, 587
<code>\l__groupthm_key_group_clist</code>	...	<code>\__groupthm_undefine_group:n</code>	...
.....	61, 112, 125, 164	.....	20, 227, 227, 341
<code>\l__groupthm_key_mapname_clist</code>	..	<code>\__groupthm_use_function_on_-</code>	
.....	61, 165	<code>name:n</code>	..... 29, 554, 554
<code>\l__groupthm_key_name_tl</code>	.....	<code>\__groupthm_use_group:n</code>	.....
.....	61, 110, 123, 151, 159	.....	29, 549, 549, 573
<code>\l__groupthm_key_prefix_tl</code>	61, 95, 162	<code>\__groupthm_use_group_\meta{theorem_group}:</code>	
<code>\l__groupthm_key_starred_-</code>		.....	201
<code>version_bool</code>	61, 116, 129, 136, 161	<code>\__groupthm_wrap_multiple:nnn</code>	...
<code>\l__groupthm_key_suffix_tl</code>	61, 97, 163	.....	24, 25, 349, 349, 360, 603, 672
<code>\l__groupthm_key_thmtools_clist</code>	..	<code>\__groupthm_wrap_multiple_-</code>	
.....	61, 103, 114, 127, 166	<code>document:nnnn</code>	.....
<code>\__groupthm_lazy:n</code>	38, 40, 322, 402, 409, 432, 439, 481, 607, 635, 676, 703, 713, 781, 862, 862, 878, 912, 912	.....	25, 388, 388, 397, 630, 698, 776
<code>\g__groupthm_lazy_auxfile_tl</code>	...	<code>\__groupthm_write_auxout:n</code>	.....
.....	15, 38, 81, 902	.....	812, 812, 816, 837, 838, 845, 847, 848
<code>\g__groupthm_lazy_document_tl</code>	...	groupthmlazy internal commands:	
.....	81, 864, 889	<code>\__groupthmlazy:n</code>	..... 449, 505
<code>\__groupthm_map_aux:n</code>	.....		
..	21, 25, 318, 335, 351, 358, 390, 395		
<code>\l__groupthm_mapname_clist</code>	.....		
.....	68, 100, 165, 213, 369, 566, 580		
<code>\l__groupthm_name_tl</code>	.....		
.....	17, 29, 68, 153, 159, 556, 558, 564, 583, 617, 625, 693		
<code>\__groupthm_new_document_-</code>			
<code>command:Nnn</code>	.... 378, 383, 387, 393		
<code>\g__groupthm_parents_group_\meta{theorem_group}_clist</code>	.. 221		
<code>\__groupthm_powerset_clist_-</code>			
<code>foreach:Nn</code>	..... 28, 510		
<code>\l__groupthm_prefix_tl</code>	.....		
.....	68, 162, 205, 367, 563, 582		
<code>\__groupthm_push_tmpa_seq:n</code>	....		
.....	26, 454, 455, 475		
<code>\__groupthm_remove_from_group_-</code>			
<code>ordering:n</code>	..... 18, 176, 176, 262		
<code>\__groupthm_set_normalized_-</code>			
<code>keys:nnn</code>	.....		
.....	17, 24, 142, 142, 364, 609, 678		
<code>\__groupthm_sort_group_names:</code>	...		
.....	192, 192, 650, 739, 756		
<code>\l__groupthm_starred_version_-</code>			
<code>bool</code>	..... 68, 161, 619, 686, 797		
<code>\l__groupthm_suffix_tl</code>	.....		
.....	68, 163, 209, 368, 565, 584		

## H

hook commands:

<code>\hook_gput_code:nnn</code>	.....
.....	186, 203, 207, 211, 215, 827, 876
<code>\hook_gremove_code:nn</code>	..... 194,
.....	231, 232, 233, 234, 568, 569, 570, 571
<code>\hook_gset_rule:nnnn</code>	..... 13,
.....	14, 23, 60, 60, 222, 223, 224, 422, 425
<code>\hook_new:n</code>	..... 55, 56, 57, 58, 59
<code>\hook_use:n</code>	... 197, 576, 577, 578, 579

## I

int commands:

<code>\int_compare:nNnTF</code>	..... 879
<code>\int_gincr:N</code>	..... 487
<code>\int_gset:Nn</code>	..... 856
<code>\int_new:N</code>	..... 76, 78, 79
<code>\int_set:Nn</code>	..... 80
<code>\int_use:N</code>	..... 483, 484, 486, 858

iow commands:

<code>\iow_log:n</code>	..... 921
<code>\iow_now:Nn</code>	..... 814
<code>\iow_term:n</code>	..... 920

## K

keys commands:

<code>\keys_define:nn</code>	.. 85, 93, 108, 121, 134
<code>\l_keys_key_str</code>	... 106, 119, 132, 139
<code>\keys_set:nn</code>	144, 146, 148, 150, 783, 784

## L

<code>\LaTeX</code>	..... 376
---------------------	-----------

<b>M</b>		
map commands:		
\map_use_on_name:n	580	
\marg	497	
msg commands:		
\msg_error:nn	14, 720	
\msg_error:nnn	13, 106, 119, 132, 139, 545	
\msg_error:nnnn	745	
\msg_error:nnnnn	13, 274, 285	
\msg_line_context:	29, 33, 37, 42, 46	
\msg_new:nnn	27, 31, 35, 39	
\msg_new:nnnn	44	
\msg_see_documentation_text:n	53	
<b>N</b>		
\NewDocumentCommand	385, 430, 447, 503, 724	
\NewFoo	4	
\NewGroupedTheorem	5, 7–9, 11, 32, 630	
\NewGroupedTheorem*	8, 630	
\NewGroupedTheoremFamily	7, 9, 10, 12, 34, 35, 698	
\NewGroupedTheoremFamily*	9, 10, 698	
\NewGroupedTheoremFamilyOptions	7, 9, 10, 12, 36, 776	
\NewGroupedTheoremFamilyOptions*	10, 12, 776	
\newtheorem	4	
\NewTheoremGroup	5, 7, 11, 397	
<b>P</b>		
powerset commands:		
\powerset_clist_foreach:Nn	533, 646	
powerset internal commands:		
\_powerset_clist_foreach_aux:Nn	513, 523, 526, 538	
\l_powerset_copied_clist	510, 515, 520, 522, 529, 535, 537, 539	
\l_powerset_saved_seq	511, 521, 524, 527, 530	
\ProcessKeysOptions	141	
\ProvideFoo	4	
\ProvideGroupedTheorem	8, 630	
\ProvideGroupedTheorem*	8, 630	
\ProvideGroupedTheoremFamily	9, 698	
\ProvideGroupedTheoremFamily*	9, 698	
\ProvideGroupedTheoremFamilyOptions	10, 776	
\ProvideGroupedTheoremFamilyOptions*	10, 776	
\ProvideTheoremGroup	5, 11, 397	
<b>R</b>		
regex commands:		
\regex_replace_all:nnN	873	
<b>S</b>		
seq commands:		
\seq_get:NN	524, 527	
\seq_if_empty_p:N	465	
\seq_new:N	511	
\seq_pop:NN	468, 530	
\seq_push:Nn	457, 521	
\seq_set_from_clist:Nn	462	
\l_tmpa_seq	457, 462, 465, 468	
\starred	7	
starred	7, 914	
str commands:		
\c_left_brace_str	845	
\c_right_brace_str	848	
\str_if_eq:nnTF	412, 416, 420	
\str_if_eq_p:nn	268, 279, 291, 292, 305	
\str_set:Nn	411, 414, 418	
\str_use:N	106, 119, 132, 139, 845, 848	
\l_tmpa_str	411, 412, 414, 416, 418, 422, 425	
<b>T</b>		
text commands:		
\text_titlecase_first:n	155, 392	
tl commands:		
\c_empty_tl	96, 98	
\c_novalue_tl	111, 124, 151	
\tl_clear:N	563, 565	
\tl_gput_right:Nn	864	
\tl_if_eq:NnTF	151	
\tl_log:n	11, 229, 273, 284, 297, 298, 308, 311, 574	
\tl_new:N	62, 63, 64, 68, 69, 70, 81, 82	
\tl_put_left:Nn	205	
\tl_put_right:Nn	209, 582, 583, 584	
\tl_set:Nn	153, 556, 564, 581, 869	
\tl_set_eq:NN	159, 162, 163	
\tl_trim_spaces:n	411	
\tl_use:N	558, 889, 902	
\l_tmpa_tl	468, 469, 470, 472, 474, 520, 521, 522, 524, 525, 527, 528, 529, 530, 581, 582, 583, 584, 585, 869, 873, 874	
<b>U</b>		
unnumbered	7, 914	

use commands:		<code>\use:n</code> . . . . . 878, 912
<code>\use:N</code> . . . . . 365, 404, 552,		
614, 622, 637, 651, 690, 705, 792, 799	<code>\use_none:n</code> . . . . . 768, 910, 911	