Thmtools Users' Guide

Ulrich M. Schwarz – ulmi@absatzen.de[*]

2010/07/05 v41

Abstract

The thmtools bundle is a collection of packages that is designed to provide an easier interface to theorems, and to facilitate some more advanced tasks.

If you are a first-time user and you don't think your requirements are out of the ordinary, browse the examples in chapter 1. If you're here because the other packages you've tried so far just can't do what you want, take inspiration from chapter 2. If you're a repeat customer, you're most likely to be interested in the refence section in chapter 3.

Contents

[*]who would like to thank the users for testing, encouragement, feature requests, and bug reports. In particular, Denis Bitouzé prompted further improvement when thmtools got stuck in a "good enough for me" slump.

1 Thmtools for the impatient

How to use this document

This guide consists mostly of examples and their output, sometimes with a few additional remarks. Since theorems are defined in the preamble and used in the document, the snippets are two-fold:

```
% Preamble code looks like this.
\usepackage{amsthm}
\usepackage{thmtools}
\declaretheorem{theorem}
```

```
% Document code looks like this.
\begin{theorem}[Euclid]
 \label{thm:euclid}%
 For every prime $p$, there is a prime $p'>p$.
 In particular, the list of primes,
 \begin{equation}\label{eq:1}
   2,3,5,7,\dots
 \end{equation}
 is infinite.
\end{theorem}
```

The result looks like this:

Theorem 1 (Euclid)**.** *For every prime p, there is a prime $p' > p$. In particular, the list of primes,*

$$2, 3, 5, 7, \dots \tag{1.1}$$

is infinite.

Note that in all cases, you will need a *backend* to provide the command `\newtheorem` with the usual behaviour. The LaTeX kernel has a built-in backend which cannot do very much; the most common backends these days are the amsthm and ntheorem packages. Throughout this document, we'll use amsthm, and some of the features won't work with ntheorem.

1.1 Elementary definitions

As you have seen above, the new command to define theorems is `\declaretheorem`, which in its most basic form just takes the name of the environment. All other options can be set through a key-val interface:

```
\usepackage{amsthm}
\usepackage{thmtools}
\declaretheorem[numberwithin=section]{theoremS}
```

```
\begin{theoremS}[Euclid]
  For every prime $p$, there is a prime $p'>p$.
  In particular, there are infinitely many primes.
\end{theoremS}
```

TheoremS 1.1.1 (Euclid)**.** *For every prime p, there is a prime $p' > p$. In particular, there are infinitely many primes.*

Instead of "numberwithin=", you can also use "parent=" and "within=". They're all the same, use the one you find easiest to remember.

Note the example above looks somewhat bad: sometimes, the name of the environment, with the first letter uppercased, is not a good choice for the theorem's title.

```
\usepackage{amsthm}
\usepackage{thmtools}
\declaretheorem[name=\"Ubung]{exercise}
```

```
\begin{exercise}
  Prove Euclid's Theorem.
\end{exercise}
```

Übung 1. *Prove Euclid's Theorem.*

To save you from having to look up the name of the key every time, you can also use "title=" and "heading=" instead of "name="; they do exactly the same and hopefully one of these will be easy to remember for you.

Of course, you do not have to follow the abominal practice of numbering theorems, lemmas, etc., separately:

```
\usepackage{amsthm}
\usepackage{thmtools}
\declaretheorem[sibling=theorem]{lemma}
```

```
\begin{lemma}
  For every prime $p$, there is a prime $p'>p$.
  In particular, there are infinitely many primes.
\end{lemma}
```

Lemma 2. *For every prime p, there is a prime $p' > p$. In particular, there are infinitely many primes.*

Again, instead of "sibling=", you can also use "numberlike=" and "sharecounter=".

Some theorems have a fixed name and are not supposed to get a number. To this end, amsthm provides `\newtheorem*`, which is accessible through thmtools:

```
\usepackage{amsthm}
\usepackage{thmtools}
\declaretheorem[numbered=no,
  name=Euclid's Prime Theorem]{euclid}
```

```
\begin{euclid}
  For every prime $p$, there is a prime $p'>p$.
  In particular, there are infinitely many primes.
\end{euclid}
```

Euclid's Prime Theorem. *For every prime p, there is a prime $p' > p$. In particular, there are infinitely many primes.*

As a somewhat odd frill, you can turn off the number if there's only one instance of the kind in the document. This might happen when you split and join your papers into short conference versions and longer journal papers and tech reports. Note that this doesn't combine well with the sibling key: how do you count like somebody who suddenly doesn't count anymore? Also, it takes an extra LaTeX run to settle.

```
\usepackage{amsthm}
\usepackage{thmtools}
\usepackage[unq]{unique}
\declaretheorem[numbered=unless unique]{singleton}
\declaretheorem[numbered=unless unique]{couple}
```

```
\begin{couple}
  Marc \& Anne
\end{couple}
\begin{singleton}
  Me.
\end{singleton}
\begin{couple}
  Buck \& Britta
\end{couple}
```

Couple 1. *Marc & Anne*

Singleton. *Me.*

Couple 2. *Buck & Britta*

1.2 Frilly references

In case you didn't know, you should: hyperref, nameref and cleveref offer ways of "automagically" knowing that `\label{foo}` was inside a theorem, so that a reference adds the string "Theorem". This is all done for you, but there's one catch: you have to tell thmtools what the name to add is. By default, it will use the title of the theorem, in particular, it will be uppercased. (This happens to match the guidelines of all publishers I have encountered.) But there is an alternate spelling available, denoted by a capital letter, and in any case, if you use cleveref, you should give two values separated by a comma, because it will generate plural forms if you reference many theorems in one `\cite`.

```
\usepackage{amsthm, thmtools}
\usepackage{
  nameref,%\nameref
  hyperref,%\autoref
  % n.b. \Autoref is defined by thmtools
  cleveref,% \cref
  % n.b. cleveref after! hyperref
}
\declaretheorem[name=Theorem,
  refname={theorem,theorems},
  Refname={Theorem,Theorems}]{callmeal}
```

```
\begin{callmeal}[Simon]\label{simon}
  One
\end{callmeal}
\begin{callmeal}\label{garfunkel}
  and another, and together,
  \autoref{simon}, ``\nameref{simon}'',
  and \cref{garfunkel} are referred
  to as \cref{simon,garfunkel}.
  \Cref{simon,garfunkel}, if you are at
  the beginning of a sentence.
\end{callmeal}
```

Theorem 1 (*Simon*)

One

Theorem 2

and another, and together, theorem 1, "Simon", and theorem 2 are referred to as theorems 1 and 2. Theorems 1 and 2, if you are at the beginning of a sentence.

1.3 Styling theorems

The major backends provide a command `\theoremstyle` to switch between looks of theorems. This is handled as follows:

```
\usepackage{amsthm}
\usepackage{thmtools}
\declaretheorem[style=remark]{remark}
\declaretheorem{Theorem}
```

```
\begin{Theorem}
  This is a theorem.
\end{Theorem}
\begin{remark}
  Note how it still retains the default style, ‘plain’.
\end{remark}
```

Theorem 1. *This is a theorem.*

Remark 1. Note how it still retains the default style, ‘plain’.

Thmtools also supports the shadethm and thmbox packages:

```
\usepackage{amsthm}
\usepackage{thmtools}
\usepackage[dvipsnames]{xcolor}
\declaretheorem[shaded={bgcolor=Lavender,
  textwidth=12em}]{BoxI}
\declaretheorem[shaded={rulecolor=Lavender,
  rulewidth=2pt, bgcolor={rgb}{1,1,1}}]{BoxII}
```

```
\begin{BoxI}[Euclid]
  For every prime $p$, there is a prime $p'>p$.
  In particular, there are infinitely many primes.
\end{BoxI}
\begin{BoxII}[Euclid]
  For every prime $p$, there is a prime $p'>p$.
  In particular, there are infinitely many primes.
\end{BoxII}
```

BoxI 1. *For every prime p, there is a prime $p' > p$. In particular, there are infinitely many primes.*

BoxII 1. *For every prime p, there is a prime $p' > p$. In particular, there are infinitely many primes.*

As you can see, the color parameters can take two forms: it's either the name of a color that is already defined, without curly braces, or it can start with a curly brace, in which case it is assumed that `\definecolor{colorname}`⟨*what you said*⟩ will be valid LaTeX code. In our case, we use the rbg model to manually specify white. (Shadethm's default value is some sort of gray.)

For the thmbox package, use the thmbox key:

```
\usepackage{amsthm}
\usepackage{thmtools}
\declaretheorem[thmbox=L]{boxtheorem L}
\declaretheorem[thmbox=M]{boxtheorem M}
\declaretheorem[thmbox=S]{boxtheorem S}
```

```
\begin{boxtheorem L}[Euclid]
  For every prime $p$, there is a prime $p'>p$.
  In particular, there are infinitely many primes.
\end{boxtheorem L}
\begin{boxtheorem M}[Euclid]
  For every prime $p$, there is a prime $p'>p$.
  In particular, there are infinitely many primes.
\end{boxtheorem M}
\begin{boxtheorem S}[Euclid]
  For every prime $p$, there is a prime $p'>p$.
  In particular, there are infinitely many primes.
\end{boxtheorem S}
```

Boxtheorem L 1 (*Euclid*)

For every prime p, there is a prime $p' > p$. In particular, there are infinitely many primes.

Boxtheorem M 1 (*Euclid*)

For every prime p, there is a prime $p' > p$. In particular, there are infinitely many primes.

Boxtheorem S 1 (*Euclid*)

For every prime p, there is a prime $p' > p$. In particular, there are infinitely many primes.

Note that for both thmbox and shaded keys, it's quite possible they will not cooperate with a style key you give at the same time.

1.3.1 Declaring new theoremstyles

Thmtools also offers a new command to define new theoremstyles. It is partly a frontend to the `\newtheoremstyle` command of amsthm or ntheorem, but it offers (more or less successfully) the settings of both to either. So we are talking about the same things, consider the sketch in Figure 1.1. To get a result like that, you would use something like

```
\declaretheoremstyle[
  spaceabove=6pt, spacebelow=6pt,
  headfont=\normalfont\bfseries,
  notefont=\mdseries, notebraces={(}{)},
  bodyfont=\normalfont,
  postheadspace=1em,
  qed=\qedsymbol
]{mystyle}
\declaretheorem[style=mystyle]{styledtheorem}
```

```
\begin{styledtheorem}[Euclid]
  For every prime $p$\dots
\end{styledtheorem}
```

Styledtheorem 1 (*Euclid*)

For every prime p... □

Again, the defaults are reasonable and you don't have to give values for everything.

There is one important thing you cannot see in this example: there are more keys you can pass to `\declaretheoremstyle`: if thmtools cannot figure out at all what to do with it, it will pass it on to the `\declaretheorem` commands that use that style. For example, you may use the boxed and shaded keys here.

To change the order in which title, number and note appear, there is a key headstyle. Currently, the values "margin" and "swapnumber" are supported. The daring may also try to give a macro here that uses the commands `\NUMBER`, `\NAME` and `\NOTE`. You cannot circumvent the fact that headpunct comes at the end, though, nor the fonts and braces you select with the other keys.

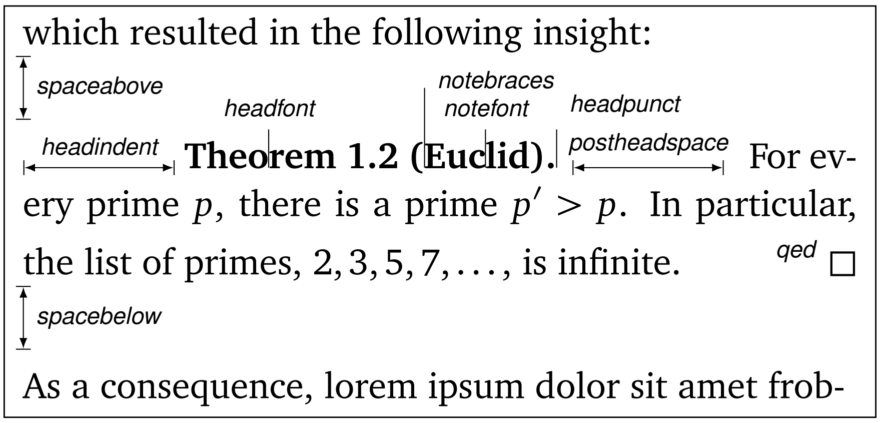
which resulted in the following insight:
spaceabove
headfont
notebraces
notefont
headpunct
headindent
postheadspace
Theorem 1.2 (Euclid). For every prime p, there is a prime $p' > p$. In particular, the list of primes, $2, 3, 5, 7, \ldots$, is infinite.
qed
spacebelow
As a consequence, lorem ipsum dolor sit amet frob-

Figure 1.1: Settable parameters of a theorem style.

1.4 Repeating theorems

Sometimes, you want to repeat a theorem you have given in full earlier, for example you either want to state your strong result in the introduction and then again in the full text, or you want to re-state a lemma in the appendix where you prove it. For example, I lied about Theorem 1 on p. 2: the true code used was

```
\usepackage{thmtools, thm-restate}
\declaretheorem{theorem}
```

```
\begin{restatable}[Euclid]{theorem}{firsteuclid}
  \label{thm:euclid}%
  For every prime $p$, there is a prime $p'>p$.
  In particular, the list of primes,
  \begin{equation}\label{eq:1}
    2,3,45,7,\dots
  \end{equation}
  is infinite.
\end{restatable}
```

and to the right, I just use

```
\firsteuclid*
\vdots
\firsteuclid*
```

Theorem 1 (Euclid)**.** *For every prime p, there is a prime $p' > p$. In particular, the list of primes,*

$$2, 3, 5, 7, \ldots \tag{1.1}$$

is infinite.

$\vdots$

Theorem 1 (Euclid)**.** *For every prime p, there is a prime $p' > p$. In particular, the list of primes,*

$$2, 3, 5, 7, \ldots \tag{1.1}$$

is infinite.

Note that in spite of being a theorem-environment, it gets number one all over again. Also, we get equation number (1.1) again. The star in `\firsteuclid*` tells thmtools that it should redirect the label mechanism, so that this reference: Theorem 1 points to p. 2, where the unstarred environment is used. (You can also use a starred environment and an unstarred command, in which case the behaviour is reversed.) Also, if you use hyperref, the links will lead you to the unstarred occurence.

Just to demonstrate that we also handle more involved cases, I repeat another theorem here, but this one was numbered within its section: note we retain the section number which does not fit the current section:

```
      \euclidii*
```

TheoremS 1.1.1 (Euclid)**.** *For every prime p, there is a prime $p' > p$. In particular, there are infinitely many primes.*

1.5 Lists of theorems

To get a list of theorems with default formatting, just use `\listoftheorems`:

```
\listoftheorems
```

List of Theorems

Not everything might be of the same importance, so you can filter out things by environment name:

```
\listoftheorems[ignoreall,
  show={theorem,Theorem,euclid}]
```

List of Theorems

And you can also restrict to those environments that have an optional argument given. Note that two theorems disappear compared to the previous example. You could also say just “onlynamed”, in which case it will apply to *all* theorem environments you have defined.

```
\listoftheorems[ignoreall,
  onlynamed={theorem,Theorem,euclid}]
```

List of Theorems

As might be expected, the heading given is defined in `\listoftheoremname`.

1.6 Extended arguments to theorem environments

Usually, the optional argument of a theorem serves just to give a note that is shown in the theorem's head. Thmtools allows you to have a key-value list here as well. The following keys are known right now:

name This is what used to be the old argument. It usually holds the name of the theorem, or a source.

label This will issue a `\label` command after the head. Not very useful, more of a demo.

continues Saying `continues=foo` will cause the number that is given to be changed to `\ref{foo}`, and a text is added to the note. (The exact text is given by the macro `\thmcontinues`, which takes the label as its argument.)

```
\begin{theorem}[name=Keyed theorem,
  label=thm:key]
  This is a
  key-val theorem.
\end{theorem}
\begin{theorem}[continues=thm:key]
  And it's spread out.
\end{theorem}
```

Theorem 3 (Keyed theorem, continuing from p. 8). *This is a key-val theorem.*

Theorem 3 (continuing from p. 8). *And it's spread out.*

2 Thmtools for the extravagant

This chapter will go into detail on the slightly more technical offerings of this bundle. In particular, it will demonstrate how to use the general hooks provided to extend theorems in the way you want them to behave. Again, this is done mostly by some examples.

2.1 Understanding thmtools' extension mechanism

Thmtools draws most of its power really only from one feature: the `\newtheorem` of the backend will, for example, create a theorem environment, i.e. the commands `\theorem` and `\endtheorem`. To add functionality, four places immediately suggest themselves: "immediately before" and "immediately after" those two.

There are two equivalent ways of adding code there: one is to call `\addtotheorempreheadhook` and its brothers and sisters `...postheadhook`, `...prefoothook` and `...postfoothook`. All of these take an *optional* argument, the name of the environment, and the new code as a mandatory argument. The environment is optional because there is also a set of "generic" hooks added to every theorem that you define.

The other way is to use the keys `preheadhook` et al. in your `\declaretheorem`. (There is no way of accessing the generic hook in this way.)

The hooks are arranged in the following way: first the specific prehead, then the generic one. Then, the original `\theorem` (or whatever) will be called. Afterwards, first the specific posthead again, then the generic one. (This means that you cannot wrap the head alone in an environment this way.) At the end of the theorem, it is the other way around: first the generic, then the specific, both before and after that `\endtheorem`. This means you can wrap the entire theorem easily by adding to the prehead and the postfoot hooks. Note that thmtools does not look inside `\theorem`, so you cannot get inside the head formatting, spacing, punctuation in this way.

In many situations, adding static code will not be enough. Your code can look at `\thmt@envname`, `\thmt@thmname` and `\thmt@optarg`, which will contain the name of the environment, its title, and, if present, the optional argument (otherwise, it is `\@empty`). *However*, you should not make assumptions about the optional argument in the preheadhook: it might still be key-value, or it might already be what will be placed as a note. (This is because the key-val handling itself is added as part of the headkeys.)

2.2 Case in point: the shaded key

Let us look at a reasonably simple example: the shaded key, which we've already seen in the first section. You'll observe that we run into a problem similar to the four-hook mess: your code may either want to modify parameters that need to be set beforehand, or it wants to modify the environment after it has been created. To hide this from the user, the code you define for the key is actually executed twice, and `\thmt@trytwice{A}{B}` will execute A on the first pass, and B on the second. Here, we want to add to the hooks, and the hooks are only there in the second pass.

Mostly, this key wraps the theorem in a shadebox environment. The parameters are set by treating the value we are given as a new key-val list, see below.

```
\define@key{thmdef}{shaded}[{}]{%
\thmt@trytwice{}{%
  \RequirePackage{shadethm}%
  \RequirePackage{thm-patch}%
  \addtotheorempreheadhook[\thmt@envname]{%
    \setlength\shadedtextwidth{\linewidth}%
    \kvsetkeys{thmt@shade}{#1}\begin{shadebox}}%
  \addtotheorempostfoothook[\thmt@envname]{\end{shadebox}}%
  }%
}
```

The docs for shadethm say:

> There are some parameters you could set the default for (try them as is, first).
>
> - shadethmcolor The shading color of the background. See the documentation for the color package, but with a 'gray' model, I find .97 looks good out of my printer, while a darker shade like .92 is needed to make it copy well. (Black is 0, white is 1.)
> - shaderulecolor The shading color of the border of the shaded box. See (i). If shadeboxrule is set to 0pt then this won't print anyway.
> - shadeboxrule The width of the border around the shading. Set it to 0pt (not just 0) to make it disappear.
> - shadeboxsep The length by which the shade box surrounds the text.

So, let's just define keys for all of these.

```
\define@key{thmt@shade}{textwidth}{\setlength\shadedtextwidth{#1}}
\define@key{thmt@shade}{bgcolor}{\thmt@definecolor{shadethmcolor}{#1}}
\define@key{thmt@shade}{rulecolor}{\thmt@definecolor{shaderulecolor}{#1}}
\define@key{thmt@shade}{rulewidth}{\setlength\shadeboxrule{#1}}
\define@key{thmt@shade}{margin}{\setlength\shadeboxsep{#1}}
```

What follows is wizardry you don't have to understand. In essence, we want to support two notions of color: one is "everything that goes after `\definecolor{shadethmcolor}`", such as `{rgb}{0.8,0.85,1}`. On the other hand, we'd also like to recognize an already defined color name such as `blue`.

To handle the latter case, we need to copy the definition of one color into another. The xcolor package offers `\colorlet` for that, for the color package, we just cross our fingers.

```
\def\thmt@colorlet#1#2{%
  %\typeout{don't know how to let color '#1' be like color '#2'!}%
  \@xa\let\csname\string\color@#1\@xa\endcsname
    \csname\string\color@#2\endcsname
  % this is dubious at best, we don't know what a backend does.
}
\AtBeginDocument{%
  \ifcsname colorlet\endcsname
    \let\thmt@colorlet\colorlet
  \fi
}
```

Now comes the interesting part: we assume that a simple color name must not be in braces, and a color definition starts with an opening curly brace. (So, if `\definecolor` ever gets an optional arg, we are in a world of pain.)

If the second argument to `\thmt@definecolor` (the key) starts with a brace, then `\thmt@def@color` will have an empty second argument, delimited by the brace of the key. Hopefully, the key will have exactly enough arguments to satisfy `\definecolor`. Then, `thmt@drop@relax` will be executed and gobble the fallback values and the `\thmt@colorlet`.

If the key does not contain an opening brace, `\thmt@def@color` will drop everything up to `{gray}{0.5}`. So, first the color gets defined to a medium gray, but then, it immediately gets overwritten with the definition corresponding to the color name.

```
\def\thmt@drop@relax#1\relax{}
\def\thmt@definecolor#1#2{%
  \thmt@def@color{#1}#2\thmt@drop@relax
    {gray}{0.5}%
    \thmt@colorlet{#1}{#2}%
  \relax
}
\def\thmt@def@color#1#2#{%
  \definecolor{#1}}
```

2.3 Case in point: the thmbox key

The thmbox package does something else: instead of having a separate environment, we have to use a command different from `\newtheorem` to get the boxed style. Fortunately, thmtools stores the command as `\thmt@theoremdefiner`, so we can modify it. (One of the perks if extension writer and framework writer are the same person.) So, in contrast to the previous example, this time we need to do something before the actual `\newtheorem` is called.

```
\define@key{thmdef}{thmbox}[L]{%
  \thmt@trytwice{%
  \let\oldproof=\proof
  \let\oldendproof=\endproof
  \let\oldexample=\example
  \let\oldendexample=\endexample
  \RequirePackage[nothm]{thmbox}
  \let\proof=\oldproof
  \let\endproof=\oldendproof
  \let\example=\oldexample
  \let\endexample=\oldendexample
  \def\thmt@theoremdefiner{\newboxtheorem[#1]}%
  }{}%
}%
```

2.4 How thmtools finds your extensions

Up to now, we have discussed how to write the code that adds functionality to your theorems, but you don't know how to activate it yet. Of course, you can put it in your preamble, likely embraced by `\makeatletter` and `\makeatother`, because you are using internal macros with @ in their name (viz., `\thmt@envname` and friends). You can also put them into a package (then, without the `\makeat...`), which is simply a file ending in .sty put somewhere that LaTeX can find it, which can then be laoded with `\usepackage`. To find out where exactly that is, and if you'd need to update administrative helper files such as a filename database FNDB, please consult the documentation of your TeX distribution.

Since you most likely want to add keys as well, there is a shortcut that thmtools offers you: whenever you use a key `key` in a `\declaretheorem` command, and thmtools doesn't already know what to do with it, it will try to `\usepackage{thmdef-key}` and evaluate the key again. (If that doesn't work, thmtools will cry bitterly.)

For example, there is no provision in thmtools itself that make the `shaded` and `thmbox` keys described above special: in fact, if you want to use a different package to create frames, you just put a different `thmdef-shaded.sty` into a preferred texmf tree. Of course, if your new package doesn't offer the old keys, your old documents might break!

The behaviour for the keys in the style definition is slightly different: if a key is not known there, it will be used as a "default key" to every theorem that is defined using this style. For example, you can give the `shaded` key in a style definition.

Lastly, the key-val arguments to the theorem environments themselves need to be loaded manually, not least because inside the document it's too late to call `\usepackage`.

3 Thmtools for the completionist

This will eventually contain a reference to all known keys, commands, etc.

3.1 Known keys to `\declaretheoremstyle`

N.b. implementation for amsthm and ntheorem is separate for these, so if it doesn't work for ntheorem, try if it works with amsthm, which in general supports more things.

Also, all keys listed as known to `\declaretheorem` are valid.

`spaceabove` Value: a length. Vertical space above the theorem, possibly discarded if the theorem is at the top of the page.

`spacebelow` Value: a length. Vertical space after the theorem, possibly discarded if the theorem is at the top of the page.

`headfont` Value: TEX code. Executed just before the head of the theorem is typeset, inside a group. Intended use it to put font switches here.

`notefont` Value: TEX code. Executed just before the note in the head is typeset, inside a group. Intended use it to put font switches here. Formatting also applies to the braces around the note. Not supported by ntheorem.

`bodyfont` Value: TEX code. Executed before the begin part of the theorem ends, but before all afterhead-hooks. Intended use it to put font switches here.

`headpunct` Value: TEX code, usually a single character. Put at the end of the theorem's head, prior to linebreaks or indents.

`notebraces` Value: Two characters, the opening and closing symbol to use around a theorem's note. (Not supported by ntheorem.)

`postheadspace` Value: a length. Horizontal space inserted after the entire head of the theorem, before the body. Does probably not apply (or make sense) for styles that have a linebreak after the head.

`headindent` Value: a length. Horizontal space inserted before the head. Some publishers like `\parindent` here for remarks, for example.

`headstyle` Value: LATEX code using the special placeholders `\NUMBER`, `\NAME` and `\NOTE`, which correspond to the (formatted, including the braces for `\NOTE` etc.) three parts of a theorem's head. This can be used to override the usual style "1.1 Theorem (Foo)", for example to let the numbers protude in the margin or put them after the name.

Additionally, a number of keywords are allowed here instead of LATEX code:

margin Lets the number protude in the (left) margin.

swapnumber Puts the number before the name. Currently not working so well for unnumbered theorems.

This list is likely to grow

3.2 Known keys to `\declaretheorem`

`parent` Value: a counter name. The theorem will be reset whenever that counter is incremented. Usually, this will be a sectioning level, `chapter` or `section`.

`numberwithin` Value: a counter name. The theorem will be reset whenever that counter is incremented. Usually, this will be a sectioning level, `chapter` or `section`. (Same as parent.)

`within` Value: a counter name. The theorem will be reset whenever that counter is incremented. Usually, this will be a sectioning level, `chapter` or `section`. (Same as parent.)

`sibling` Value: a counter name. The theorem will use this counter for numbering. Usually, this is the name of another theorem environment.

`numberlike` Value: a counter name. The theorem will use this counter for numbering. Usually, this is the name of another theorem environment. (Same as sibling.)

`sharenumber` Value: a counter name. The theorem will use this counter for numbering. Usually, this is the name of another theorem environment. (Same as sibling.)

`title` Value: TEX code. The title of the theorem. Default is the name of the environment, with `\MakeUppercase` prepended. You'll have to give this if your title starts with a accented character, for example.

`name` Value: TEX code. The title of the theorem. Default is the name of the environment, with `\MakeUppercase` prepended. You'll have to give this if your title starts with a accented character, for example. (Same as title.)

`heading` Value: TEX code. The title of the theorem. Default is the name of the environment, with `\MakeUppercase` prepended. You'll have to give this if your title starts with a accented character, for example. (Same as title.)

`numbered` Value: one of the keywords `yes`, `no` or `unless unique`. The theorem will be numbered, not numbered, or only numbered if it occurs more than once in the document. (The latter requires another LATEX run and will not work well combined with `sibling`.)

`style` Value: the name of a style defined with `\declaretheoremstyle` or `\newtheoremstyle`. The theorem will use the settings of this style.

`preheadhook` Value: LATEX code. This code will be executed at the beginning of the environment, even before vertical spacing is added and the head is typeset. However, it is already within the group defined by the environment.

`postheadhook` Value: LATEX code. This code will be executed after the call to the original begin-theorem code. Note that all backends seem to delay typesetting the actual head, so code here should probably enter horizontal mode to be sure it is after the head, but this will change the spacing/wrapping behaviour if your body starts with another list.

`prefoothook` Value: LATEX code. This code will be executed at the end of the body of the environment.

`postfoothook` Value: LATEX code. This code will be executed at the end of the environment, even after eventual vertical spacing, but still within the group defined by the environment.

`refname` Value: one string, or two string separated by a comma (no spaces). This is the name of the theorem as used by `\autoref`, `\cref` and friends. If it is two strings, the second is the plural form used by `\cref`. Default value is the value of `name`, i.e. usually the environment name, with .

`Refname` Value: one string, or two string separated by a comma (no spaces). This is the name of the theorem as used by `\Autoref`, `\Cref` and friends. If it is two strings, the second is the plural form used by `\Cref`. This can be used for alternate spellings, for example if your style requests no abbreviations at the beginning of a sentence. No default.

`shaded` Value: a key-value list, where the following keys are possible:

textwidth The linewidth within the theorem.

bgcolor The color of the background of the theorem. Either a color name or a color spec as accepted by `\definecolor`, such as `{gray}{0.5}`.

rulecolor The color of the box surrounding the theorem. Either a color name or a color spec.

rulewidth The width of the box surrounding the theorem.

margin The length by which the shade box surrounds the text.

`thmbox` Value: one of the characters L, M and S; see examples above.

3.3 Known keys to in-document theorems

`label` Value: a legal `\label` name. Issues a `\label` command after the theorem's head.

`name` Value: TEX code that will be typeset. What you would have put in the optional argument in the non-keyval style, i.e. the note to the head. This is *not* the same as the name key to `\declaretheorem`, you cannot override that from within the document.

`listhack` Value: doesn't matter. (But put something to trigger key-val behaviour, maybe listhack=true.) Linebreak styles in amsthm don't linebreak if they start with another list, like an `enumerate` environment. Giving the `listhack` key fixes that. *Don't* give this key for non-break styles, you'll get too little vertical space! (Just use `\leavevmode` manually there.) An all-around listhack that handles both situations might come in a cleaner rewrite of the style system.

3.4 Restatable – hints and caveats

TBD.

- Some counters are saved so that the same values appear when you re-use them. The list of these counters is stored in the macro `\thmt@innercounters` as a comma-separated list without spaces; default: equation.
- To preserve the influence of other counters (think: equation numbered per section and recall the theorem in another section), we need to know all macros that are used to turn a counter into printed output. Again, comma-separated list without spaces, without leading backslash, stored as `\thmt@counterformatters`. Default: `@alph,@Alph,@arabic,@roman,@Roman,@fnsymbol` All these only take the LATEX counter `\c@foo` as arguments. If you bypass this and use `\romannumeral`, your numbers go wrong and you get what you deserve. Important if you have very strange numbering, maybe using greek letters or somesuch.
- I think you cannot have one stored counter within another one's typeset representation. I don't think that ever occurs in reasonable circumstances, either. Only one I could think of: multiple subequation blocks that partially overlap the theorem. Dude, that doesn't even nest. You get what you deserve.

- `\label` and amsmath's `\ltx@label` are disabled inside the starred execution. Possibly, `\phantomsection` should be disabled as well?

A Thmtools for the morbidly curious

This chapter consists of the implementation of Thmtools, in case you wonder how this or that feature was implemented. Read on if you want a look under the bonnet, but you enter at your own risk, and bring an oily rag with you.

A.1 Core functionality

A.1.1 The main package

```
\DeclareOption{debug}{%
  \def\thmt@debug{\typeout}%
}
% common abbreviations and marker macros.
\let\@xa\expandafter
\let\@nx\noexpand
\def\thmt@debug{\@gobble}
\def\thmt@quark{\thmt@quark}
\newtoks\thmt@toks

\@for\opt:=lowercase,uppercase,anycase\do{%
  \@xa\DeclareOption\@xa{\opt}{%
    \@xa\PassOptionsToPackage\@xa{\CurrentOption}{thm-kv}%
  }%
}

\ProcessOptions\relax

% a scratch counter, mostly for fake hyperlinks
\newcounter{thmt@dummyctr}%
\def\theHthmt@dummyctr{dummy.\arabic{thmt@dummyctr}}%
\def\thethmt@dummyctr{}%


\RequirePackage{thm-patch, thm-kv,
  thm-autoref, thm-listof,
  thm-restate}

% Glue code for the big players.
\@ifpackageloaded{amsthm}{%
  \RequirePackage{thm-amsthm}
}{%
  \AtBeginDocument{%
  \@ifpackageloaded{amsthm}{%
    \PackageWarningNoLine{thmtools}{%
      amsthm loaded after thmtools
    }{}%
  }}%
}
\@ifpackageloaded{ntheorem}{%
  \RequirePackage{thm-ntheorem}
}{%
  \AtBeginDocument{%
  \@ifpackageloaded{ntheorem}{%
    \PackageWarningNoLine{thmtools}{%
      ntheorem loaded after thmtools
    }{}%
  }}%
}
\@ifclassloaded{beamer}{%
  \RequirePackage{thm-beamer}
}{}
\@ifclassloaded{llncs}{%
  \RequirePackage{thm-llncs}
}{}
```

A.1.2 Adding hooks to the relevant commands

This package is maybe not very suitable for the end user. It redefines `\newtheorem` in a way that lets other packages (or the user) add code to the newly-defined theorems, in a reasonably cross-compatible (with the kernel, theorem and amsthm) way.

Warning: the new `\newtheorem` is a superset of the allowed syntax. For example, you can give a star and both optional arguments, even though you cannot have an unnumbered theorem that shares a counter and yet has a different reset-regimen. At some point, your command is re-assembled and passed on to the original `\newtheorem`. This might complain, or give you the usual "Missing `\begin{document}`" that marks too many arguments in the preamble.

A call to `\addtotheorempreheadhook[`*kind*`]{`*code*`}` will insert the code to be executed whenever a kind theorem is opened, before the actual call takes place. (I.e., before the header "Kind 1.3 (Foo)" is typeset.) There are also posthooks that are executed after this header, and the same for the end of the environment, even though nothing interesting ever happens there. These are useful to put `\begin{shaded}`...`\end{shaded}` around your theorems. Note that foothooks are executed LIFO (last addition first) and headhooks are executed FIFO (first addition first). There is a special kind called generic that is called for all theorems. This is the default if no kind is given.

The added code may examine `\thmt@thmname` to get the title, `\thmt@envname` to get the environment's name, and `\thmt@optarg` to get the extra optional title, if any.

```
\RequirePackage{parseargs}

\newif\ifthmt@isstarred
\newif\ifthmt@hassibling
\newif\ifthmt@hasparent

\def\thmt@parsetheoremargs#1{%
  \parse{%
    {\parseOpt[]{\def\thmt@optarg{##1}}{%
      \let\thmt@shortoptarg\@empty
      \let\thmt@optarg\@empty}}%
    {%
      \def\thmt@local@preheadhook{}%
      \def\thmt@local@postheadhook{}%
      \def\thmt@local@prefoothook{}%
      \def\thmt@local@postfoothook{}%
      \thmt@local@preheadhook
      \csname thmt@#1@preheadhook\endcsname
      \thmt@generic@preheadhook
      % change following to \@xa-orgy at some point?
      % forex, might have keyvals involving commands.
      %\protected@edef\tmp@args{%
      %  \ifx\@empty\thmt@optarg\else [{\thmt@optarg}]\fi
      %}%
      \ifx\@empty\thmt@optarg
        \def\tmp@args{}%
      \else
        \@xa\def\@xa\tmp@args\@xa{\@xa[\@xa{\thmt@optarg}]}%
      \fi
      \csname thmt@original@#1\@xa\endcsname\tmp@args
    %%moved down: \thmt@local@postheadhook
    %% (give postheadhooks a chance to re-set nameref data)
    \csname thmt@#1@postheadhook\endcsname
    \thmt@generic@postheadhook
    \thmt@local@postheadhook
    \let\@parsecmd\@empty
  }%
 }%
}%

\let\thmt@original@newtheorem\newtheorem
\let\thmt@theoremdefiner\thmt@original@newtheorem

\def\newtheorem{%
  \thmt@isstarredfalse
  \thmt@hassiblingfalse
  \thmt@hasparentfalse
  \parse{%
    {\parseFlag*{\thmt@isstarredtrue}{}}%
    {\parseMand{\def\thmt@envname{##1}}}%
    {\parseOpt[]{\thmt@hassiblingtrue\def\thmt@sibling{##1}}{}}%
    {\parseMand{\def\thmt@thmname{##1}}}%
    {\parseOpt[]{\thmt@hasparenttrue\def\thmt@parent{##1}}{}}%
    {\let\@parsecmd\thmt@newtheoremiv}%
  }%
}

\newcommand\thmt@newtheoremiv{%
  \thmt@newtheorem@predefinition
  % whee, now reassemble the whole shebang.
  \protected@edef\thmt@args{%
    \@nx\thmt@theoremdefiner%
    \ifthmt@isstarred *\fi
    {\thmt@envname}%
    \ifthmt@hassibling [\thmt@sibling]\fi
    {\thmt@thmname}%
    \ifthmt@hasparent [\thmt@parent]\fi
  }
  \thmt@args
  \thmt@newtheorem@postdefinition
}

\newcommand\thmt@newtheorem@predefinition{}
\newcommand\thmt@newtheorem@postdefinition{}

\g@addto@macro\thmt@newtheorem@predefinition{%
  \@xa\thmt@providetheoremhooks\@xa{\thmt@envname}%
}
\g@addto@macro\thmt@newtheorem@postdefinition{%
  \@xa\thmt@addtheoremhook\@xa{\thmt@envname}%
  \ifthmt@isstarred\@namedef{the\thmt@envname}{}\fi
  \protected@edef\thmt@tmp{%
    \def\@nx\thmt@envname{\thmt@envname}%
    \def\@nx\thmt@thmname{\thmt@thmname}%
  }%
  \@xa\addtotheorempreheadhook\@xa[\@xa\thmt@envname\@xa]\@xa{%
    \thmt@tmp
  }%
}
\newcommand\thmt@providetheoremhooks[1]{%
  \@namedef{thmt@#1@preheadhook}{}%
  \@namedef{thmt@#1@postheadhook}{}%
  \@namedef{thmt@#1@prefoothook}{}%
  \@namedef{thmt@#1@postfoothook}{}%
  \def\thmt@local@preheadhook{}%
  \def\thmt@local@postheadhook{}%
  \def\thmt@local@prefoothook{}%
  \def\thmt@local@postfoothook{}%
}
\newcommand\thmt@addtheoremhook[1]{%
  % this adds two command calls to the newly-defined theorem.
  \@xa\let\csname thmt@original@#1\@xa\endcsname
          \csname#1\endcsname
  \@xa\renewcommand\csname #1\endcsname{%
    \thmt@parsetheoremargs{#1}%
  }%
  \@xa\let\csname thmt@original@end#1\@xa\endcsname\csname end#1\endcsname
  \@xa\def\csname end#1\endcsname{%
    % these need to be in opposite order of headhooks.
    \csname thmtgeneric@prefoothook\endcsname
    \csname thmt@#1@prefoothook\endcsname
    \csname thmt@local@prefoothook\endcsname
    \csname thmt@original@end#1\endcsname
    \csname thmt@generic@postfoothook\endcsname
    \csname thmt@#1@postfoothook\endcsname
    \csname thmt@local@postfoothook\endcsname
  }%
}
\newcommand\thmt@generic@preheadhook{\refstepcounter{thmt@dummyctr}}
\newcommand\thmt@generic@postheadhook{}
\newcommand\thmt@generic@prefoothook{}
\newcommand\thmt@generic@postfoothook{}

\def\thmt@local@preheadhook{}
\def\thmt@local@postheadhook{}
\def\thmt@local@prefoothook{}
\def\thmt@local@postfoothook{}


\providecommand\g@prependto@macro[2]{%
  \begingroup
    \toks@\@xa{\@xa{#1}{#2}}%
    \def\tmp@a##1##2{##2##1}%
    \@xa\@xa\@xa\gdef\@xa\@xa\@xa#1\@xa\@xa\@xa{\@xa\tmp@a\the\toks@}%
  \endgroup
}

\newcommand\addtotheorempreheadhook[1][generic]{%
  \expandafter\g@addto@macro\csname thmt@#1@preheadhook\endcsname%
}
\newcommand\addtotheorempostheadhook[1][generic]{%
  \expandafter\g@addto@macro\csname thmt@#1@postheadhook\endcsname%
}

\newcommand\addtotheoremprefoothook[1][generic]{%
  \expandafter\g@prependto@macro\csname thmt@#1@prefoothook\endcsname%
}
\newcommand\addtotheorempostfoothook[1][generic]{%
  \expandafter\g@prependto@macro\csname thmt@#1@postfoothook\endcsname%
}

```

Since rev1.16, we add hooks to the proof environment as well, if it exists. If it doesn't exist at this point, we're probably using ntheorem as backend, where it goes through the regular theorem mechanism anyway.

```
\ifx\proof\endproof\else% yup, that's a quaint way of doing it :)
  % FIXME: this assumes proof has the syntax of theorems, which
  % usually happens to be true (optarg overrides "Proof" string).
  % FIXME: refactor into thmt@addtheoremhook, but we really don't want to
  % call the generic-hook...
  \let\thmt@original@proof=\proof
  \renewcommand\proof{%
    \thmt@parseproofargs%
  }%
  \def\thmt@parseproofargs{%
    \parse{%
      {\parseOpt[]{\def\thmt@optarg{##1}}{\let\thmt@optarg\@empty}}%
      {%
        \thmt@proof@preheadhook
        %\thmt@generic@preheadhook
        \protected@edef\tmp@args{%
          \ifx\@empty\thmt@optarg\else [\thmt@optarg]\fi
        }%
        \csname thmt@original@proof\@xa\endcsname\tmp@args
        \thmt@proof@postheadhook
        %\thmt@generic@postheadhook
        \let\@parsecmd\@empty
      }%
    }%
  }%

  \let\thmt@original@endproof=\endproof
  \def\endproof{%
    % these need to be in opposite order of headhooks.
    %\csname thmtgeneric@prefoothook\endcsname
    \thmt@proof@prefoothook
    \thmt@original@endproof
    %\csname thmt@generic@postfoothook\endcsname
    \thmt@proof@postfoothook
  }%
  \@namedef{thmt@proof@preheadhook}{}%
  \@namedef{thmt@proof@postheadhook}{}%
  \@namedef{thmt@proof@prefoothook}{}%
  \@namedef{thmt@proof@postfoothook}{}%
\fi
```

A.1.3 The key-value interfaces

```

\let\@xa\expandafter
\let\@nx\noexpand

\DeclareOption{lowercase}{%
  \PackageInfo{thm-kv}{Theorem names will be lowercased}%
  \global\let\thmt@modifycase\MakeLowercase}

\DeclareOption{uppercase}{%
  \PackageInfo{thm-kv}{Theorem names will be uppercased}%
  \global\let\thmt@modifycase\MakeUppercase}

\DeclareOption{anycase}{%
  \PackageInfo{thm-kv}{Theorem names will be unchanged}%
  \global\let\thmt@modifycase\@empty}
```

```

\ExecuteOptions{uppercase}
\ProcessOptions\relax

\RequirePackage{keyval,kvsetkeys,thm-patch}

\@ifpackagelater{kvsetkeys}{2010/07/02}{%
  % assume Heiko goes along with my patch...
}{%
  \RequirePackage{etex}
  \PackageInfo{thm-kv}{kvsetkeys patch applied}%
  \long\def\kv@processor@default#1#2#3{%
    \protected@edef\kvsu@fam{#1}% new
    \@onelevel@sanitize\kvsu@fam% new
    \protected@edef\kvsu@key{#2}% new
    \@onelevel@sanitize\kvsu@key% new
    \unless\ifcsname KV@#1@\kvsu@key\endcsname
      \unless\ifcsname KVS@#1@handler\endcsname
        \kv@error@unknownkey{#1}{\kvsu@key}%
      \else
        \csname KVS@#1@handler\endcsname{#2}{#3}%
      % still using #2 #3 here is intentional: handler might
      % be used for strange stuff like implementing key names
      % that contain strange characters or other strange things.
        \relax
      \fi
    \else
      \ifx\kv@value\relax
        \unless\ifcsname KV@#1@\kvsu@key @default\endcsname
          \kv@error@novalue{#1}{\kvsu@key}%
        \else
          \csname KV@#1@\kvsu@key @default\endcsname
          \relax
        \fi
      \else
        \csname KV@#1@\kvsu@key\endcsname{#3}%
      \fi
    \fi
  }
}

% useful key handler defaults.
\newcommand\thmt@mkignoringkeyhandler[1]{%
  \kv@set@family@handler{#1}{%
    \thmt@debug{Key ‘##1’ with value ‘##2’ ignored by #1.}%
  }%
}
\newcommand\thmt@mkextendingkeyhandler[3]{%
% #1: family
% #2: prefix for file
% #3: key hint for error
  \kv@set@family@handler{#1}{%
    \thmt@selfextendingkeyhandler{#1}{#2}{#3}%
      {##1}{##2}%
  }%
}

\newcommand\thmt@selfextendingkeyhandler[5]{%
  % #1: family
  % #2: prefix for file
  % #3: key hint for error
  % #4: actual key
  % #5: actual value
  \IfFileExists{#2-#4.sty}{%
    \PackageInfo{thmtools}%
      {Automatically pulling in ‘#2-#4’}%
    \RequirePackage{#2-#4}%
    \ifcsname KV@#1@#4\endcsname
      \csname KV@#1@#4\endcsname{#5}%
    \else
      \PackageError{thmtools}%
      {#3 ‘#4’ not known}
      {I don’t know what that key does.\MessageBreak
       I’ve even loaded the file ‘#2-#4.sty’, but that didn’t help.
      }%
    \fi
  }{%
    \PackageError{thmtools}%
    {#3 ‘#4’ not known}
    {I don’t know what that key does by myself,\MessageBreak
     and no file ‘#2-#4.sty’ to tell me seems to exist.
    }%
  }%
}


\newif\if@thmt@firstkeyset

% many keys are evaluated twice, because we don’t know
% if they make sense before or after, or both.
\def\thmt@trytwice{%
  \if@thmt@firstkeyset
    \@xa\@firstoftwo
  \else
    \@xa\@secondoftwo
  \fi
}

\@for\keyname:=parent,numberwithin,within\do{%
\define@key{thmdef}{\keyname}{\thmt@trytwice{\thmt@setparent{#1}}{}}%
}

\@for\keyname:=sibling,numberlike,sharenumber\do{%
\define@key{thmdef}{\keyname}{\thmt@trytwice{\thmt@setsibling{#1}}{}}%
}

\@for\keyname:=title,name,heading\do{%
\define@key{thmdef}{\keyname}{\thmt@trytwice{\thmt@setthmname{#1}}{}}%
}

\@for\keyname:=unnumbered,starred\do{%
\define@key{thmdef}{\keyname}[]{\thmt@trytwice{\thmt@isnumberedfalse}{}}%
}

\def\thmt@YES{yes}
\def\thmt@NO{no}
\def\thmt@UNIQUE{unless unique}
\define@key{thmdef}{numbered}[\thmt@YES]{
  \def\thmt@tmp{#1}%
  \thmt@trytwice{%
    \ifx\thmt@tmp\thmt@YES
      \thmt@isnumberedtrue
    \else\ifx\thmt@tmp\thmt@NO
      \thmt@isnumberedfalse
    \else\ifx\thmt@tmp\thmt@UNIQUE
      \RequirePackage[unq]{unique}
      \ifuniq{\thmt@envname}{%
        \thmt@isnumberedfalse
      }{%
        \thmt@isnumberedtrue
      }%
    \else
      \PackageError{thmtools}{Unknown value ‘#1’ to key numbered}{}%
    \fi\fi\fi
  }{% trytwice: after definition
    \ifx\thmt@tmp\thmt@UNIQUE
      \addtotheorempreheadhook[\thmt@envname]{\setuniqmark{\thmt@envname}}%
      \addtotheorempreheadhook[\thmt@envname]{\def\thmt@dummyctrautorefname{\thmt@thmname\
    \fi
  }%
}


\define@key{thmdef}{preheadhook}{\thmt@trytwice{}{\addtotheorempreheadhook[\thmt@envname]{
\define@key{thmdef}{postheadhook}{\thmt@trytwice{}{\addtotheorempostheadhook[\thmt@envname]
\define@key{thmdef}{prefoothook}{\thmt@trytwice{}{\addtotheoremprefoothook[\thmt@envname]{
\define@key{thmdef}{postfoothook}{\thmt@trytwice{}{\addtotheorempostfoothook[\thmt@envname]

\define@key{thmdef}{style}{\thmt@trytwice{\thmt@setstyle{#1}}{}}

% ugly hack: style needs to be evaluated first so its keys
% are not overridden by explicit other settings
\define@key{thmdef0}{style}{%
  \ifcsname thmt@style #1@defaultkeys\endcsname
    \thmt@toks{\kvsetkeys{thmdef}}%
    \@xa\@xa\@xa\the\@xa\@xa\@xa\thmt@toks\@xa\@xa\@xa{%
      \csname thmt@style #1@defaultkeys\endcsname}%
  \fi
}
\thmt@mkignoringkeyhandler{thmdef0}

% fallback definition.
% actually, only the kernel does not provide \theoremstyle.
% is this one worth having glue code for the theorem package?
\def\thmt@setstyle#1{%
  \PackageWarning{thm-kv}{%
    Your backend doesn’t have a ‘\string\theoremstyle’ command.
  }%
}

\ifcsname theoremstyle\endcsname
  \let\thmt@originalthmstyle\theoremstyle
  \def\thmt@outerstyle{plain}
  \renewcommand\theoremstyle[1]{%
    \def\thmt@outerstyle{#1}%
    \thmt@originalthmstyle{#1}%
  }
  \def\thmt@setstyle#1{%
    \thmt@originalthmstyle{#1}%
  }
  \g@addto@macro\thmt@newtheorem@postdefinition{%
    \thmt@originalthmstyle{\thmt@outerstyle}%
  }
```

```
\fi

\newif\ifthmt@isnumbered
\newcommand\thmt@setparent[1]{%
  \def\thmt@parent{#1}%
}
\newcommand\thmt@setsibling{%
  \def\thmt@sibling
}
\newcommand\thmt@setthmname{%
  \def\thmt@thmname
}

\thmt@mkextendingkeyhandler{thmdef}{thmdef}{\string\declaretheorem\space key}

\let\thmt@newtheorem\newtheorem

\newcommand\declaretheorem[2][]{%
  % why was that here?
  %\let\thmt@theoremdefiner\thmt@original@newtheorem
  \def\thmt@envname{#2}%
  \thmt@setthmname{\thmt@modifycase #2}%
  \thmt@setparent{}%
  \thmt@setsibling{}%
  \thmt@isnumberedtrue%
  \@thmt@firstkeysettrue%
  \kvsetkeys{thmdef0}{#1}%
  \kvsetkeys{thmdef}{#1}%
  \protected@edef\thmt@tmp{%
    \@nx\thmt@newtheorem
    \ifthmt@isnumbered\else *\fi
    {#2}%
    \ifx\thmt@sibling\@empty\else [\thmt@sibling]\fi
    {\thmt@thmname}%
    \ifx\thmt@parent\@empty\else [\thmt@parent]\fi
    \relax% added so we can delimited-read everything later
    % (recall newtheorem is patched)
  }%\show\thmt@tmp
  \thmt@tmp
  % uniquely ugly kludge: some keys make only sense
  % afterwards.
  % and it gets kludgier: again, the default-inherited
  % keys need to have a go at it.
  \@thmt@firstkeysetfalse%
  \kvsetkeys{thmdef0}{#1}%
  \kvsetkeys{thmdef}{#1}%
}
\@onlypreamble\declaretheorem

\providecommand\thmt@quark{\thmt@quark}

% in-document keyval, i.e. \begin{theorem}[key=val,key=val]

\thmt@mkextendingkeyhandler{thmuse}{thmuse}{\thmt@envname\space optarg key}

\addtotheorempreheadhook{%
  \ifx\thmt@optarg\@empty\else
    \@xa\thmt@garbleoptarg\@xa{\thmt@optarg}\fi
}%

\newif\ifthmt@thmuse@iskv
```

```

\providecommand\thmt@garbleoptarg[1]{%
  \thmt@thmuse@iskvfalse
  \def\thmt@newoptarg{\@gobble}%
  \def\thmt@newoptargextra{}%
  \def\thmt@warn@unusedkeys{}%
  \@for\thmt@fam:=\thmt@thmuse@families\do{%
    \kvsetkeys{\thmt@fam}{#1}%
  }%
  \ifthmt@thmuse@iskv
    \protected@edef\thmt@optarg{%
      \@xa\thmt@newoptarg
      \thmt@newoptargextra\@empty
    }%
    \protected@edef\thmt@shortoptarg{\thmt@newoptarg\@empty}%
    \thmt@warn@unusedkeys
  \else
    \def\thmt@optarg{#1}%
    \def\thmt@shortoptarg{#1}%
  \fi
}
\def\thmt@splitopt#1=#2\thmt@quark{%
  \def\thmt@tmpkey{#1}%
  \ifx\thmt@tmpkey\@empty
    \def\thmt@tmpkey{\thmt@quark}%
  \fi
  \@onelevel@sanitize\thmt@tmpkey
}

\def\thmt@thmuse@families{thm@track@keys}

\kv@set@family@handler{thm@track@keys}{%
  \@onelevel@sanitize\kv@key
  \@namedef{thmt@unusedkey@\kv@key}{%
    \PackageWarning{thmtools}{Unused key ‘#1’}%
  }%
  \@xa\g@addto@macro\@xa\thmt@warn@unusedkeys\@xa{%
    \csname thmt@unusedkey@\kv@key\endcsname
  }
}

% key, code.
\def\thmt@define@thmuse@key#1#2{%
  \g@addto@macro\thmt@thmuse@families{,#1}%
  \define@key{#1}{#1}{\thmt@thmuse@iskvtrue
    \@namedef{thmt@unusedkey@#1}{}%
    #2}%
  \thmt@mkignoringkeyhandler{#1}%
}

\thmt@define@thmuse@key{label}{%
  \addtotheorempostheadhook[local]{\label{#1}}%
}
\thmt@define@thmuse@key{name}{%
  \def\thmt@newoptarg{#1\@iden}%
}

\providecommand\thmt@suspendcounter[2]{%
  \@xa\protected@edef\csname the#1\endcsname{#2}%
  \@xa\let\csname c@#1\endcsname\c@thmt@dummyctr
}
```

```

\providecommand\thmcontinues[1]{%
  \ifcsname hyperref\endcsname
    \hyperref[#1]{continuing}
  \else
    continuing
  \fi
  from p.\,\pageref{#1}%
}

\thmt@define@thmuse@key{continues}{%
  \thmt@suspendcounter{\thmt@envname}{\thmt@trivialref{#1}{??}}%
  \g@addto@macro\thmt@newoptarg{{, }%
    \thmcontinues{#1}%
    \@iden}%
}


```

Defining new theorem styles; keys are in opt-arg even though not having any doesn't make much sense. It doesn't do anything exciting here, it's up to the glue layer to provide keys.

```
\def\thmt@declaretheoremstyle@setup{}
\def\thmt@declaretheoremstyle#1{%
  \PackageWarning{thmtools}{Your backend doesn't allow styling theorems}{}
}
\newcommand\declaretheoremstyle[2][]{%
  \def\thmt@style{#2}%
  \@xa\def\csname thmt@style \thmt@style @defaultkeys\endcsname{}%
  \thmt@declaretheoremstyle@setup
  \kvsetkeys{thmstyle}{#1}%
  \thmt@declaretheoremstyle{#2}%
}
\@onlypreamble\declaretheoremstyle

\kv@set@family@handler{thmstyle}{%
  \@onelevel@sanitize\kv@value
  \@onelevel@sanitize\kv@key
  \PackageInfo{thmtools}{%
    Key `\kv@key' (with value `\kv@value')\MessageBreak
    is not a known style key.\MessageBreak
    Will pass this to every \string\declaretheorem\MessageBreak
    that uses `style=\thmt@style'%
  }%
  \ifx\kv@value\relax% no value given, don't pass on {}!
    \@xa\g@addto@macro\csname thmt@style \thmt@style @defaultkeys\endcsname{%
      #1,%
    }%
  \else
    \@xa\g@addto@macro\csname thmt@style \thmt@style @defaultkeys\endcsname{%
      #1={#2},%
    }%
  \fi
}
```

A.1.4 Lists of theorems

This package provides two main commands: `\listoftheorems` will generate, well, a list of all theorems, lemmas, etc. in your document. This list is hyperlinked if you use hyperref, and it will list the optional argument to the theorem.

Currently, some options can be given as an optional argument keyval list:

numwidth The width allocated for the numbers, default 2.3em. Since you are more likely to have by-section numbering than with figures, this needs to be accessible.

ignore=foo,bar A last-second call to `\ignoretheorems`, see below.

onlynamed=foo,bar Only list those foo and bar environments that had an optional title. This weeds out unimportant definitions, for example. If no argument is given, this applies to all environments defined by `\newtheorem` and `\declaretheorem`.

show=foo,bar Undo a previous `\ignoretheorems` and restore default formatting for these environments. Useful in combination with ignoreall.

ignoreall

showall Like applying ignore or show with a list of all theorems you have defined.

The heading name is stored in the macro `\listtheoremname` and is “List of Theorems” by default. All other formatting aspects are taken from `\listoffigures`. (As a matter of fact, `\listoffigures` is called internally.)

`\ignoretheorems{`*remark,example,...*`}` can be used to suppress some types of theorem from the LoTh. Be careful not to have spaces in the list, those are currently *not* filtered out.

There’s currently no interface to change the look of the list. If you’re daring, the code for the theorem type “lemma” is in `\l@lemma` and so on.

```
\let\@xa=\expandafter
\let\@nx=\noexpand
\RequirePackage{thm-patch,keyval,kvsetkeys}

\def\thmtlo@oldchapter{0}%
\newcommand\thmtlo@chaptervspacehack{}
\ifcsname chapter\endcsname
  \def\thmtlo@chaptervspacehack{%
    \ifnum \value{chapter}>\thmtlo@oldchapter\relax
      % new chapter, add vspace to loe.
      \addtocontents{loe}{\protect\addvspace{10\p@}}%
      \xdef\thmtlo@oldchapter{\arabic{chapter}}%
    \fi
  }%
\fi

\providecommand\listtheoremname{List of Theorems}
\newcommand\listoftheorems[1][]{%
  %% much hacking here to pick up the definition from the class
  %% without oodles of conditionals.
  \bgroup
  \setlisttheoremstyle{#1}%
  \let\listfigurename\listtheoremname
  \def\contentsline##1{%
    \csname thmt@contentsline@##1\endcsname{##1}%
  }%
  \@for\thmt@envname:=\thmt@allenvs\do{%
  \@xa\protected@edef\csname l@\thmt@envname\endcsname{% CHECK: why p@edef?
    \@nx\@dottedtocline{1}{1.5em}{\@nx\thmt@listnumwidth}%
  }%
  }%
  \let\thref@starttoc\@starttoc
  \def\@starttoc##1{\thref@starttoc{loe}}%
  % new hack: to allow multiple calls, we defer the opening of the
  % loe file to AtEndDocument time. This is before the aux file is
  % read back again, that is early enough.
  % TODO: is it? crosscheck include/includeonly!
  \@fileswfalse
  \AtEndDocument{%
    \if@filesw
      \@ifundefined{tf@loe}{%
        \expandafter\newwrite\csname tf@loe\endcsname
        \immediate\openout \csname tf@loe\endcsname \jobname.loe\relax
      }{}%
    \fi
  }%
  %\expandafter
  \listoffigures
  \egroup
}

\newcommand\setlisttheoremstyle[1]{%
  \kvsetkeys{thmt-listof}{#1}%
}
\define@key{thmt-listof}{numwidth}{\def\thmt@listnumwidth{#1}}
\define@key{thmt-listof}{ignore}[\thmt@allenvs]{\ignoretheorems{#1}}
\define@key{thmt-listof}{onlynamed}[\thmt@allenvs]{\onlynamedtheorems{#1}}
\define@key{thmt-listof}{show}[\thmt@allenvs]{\showtheorems{#1}}
\define@key{thmt-listof}{ignoreall}[true]{\ignoretheorems{\thmt@allenvs}}
\define@key{thmt-listof}{showall}[true]{\showtheorems{\thmt@allenvs}}

\providecommand\thmt@listnumwidth{2.3em}

\providecommand\thmtformatoptarg[1]{ (#1)}

\newcommand\thmt@mklistcmd{%
  \@xa\protected@edef\csname l@\thmt@envname\endcsname{% CHECK: why p@edef?
    \@nx\@dottedtocline{1}{1.5em}{\@nx\thmt@listnumwidth}%
  }%
  \ifthmt@isstarred
    \@xa\def\csname ll@\thmt@envname\endcsname{%
      \protect\numberline{\protect\let\protect\autodot\protect\@empty}%
      \thmt@thmname
      \ifx\@empty\thmt@shortoptarg\else\protect\thmtformatoptarg{\thmt@shortoptarg}\fi
    }%
  \else
    \@xa\def\csname ll@\thmt@envname\endcsname{%
      \protect\numberline{\csname the\thmt@envname\endcsname}%
      \thmt@thmname
      \ifx\@empty\thmt@shortoptarg\else\protect\thmtformatoptarg{\thmt@shortoptarg}\fi
    }%
  \fi
  \@xa\gdef\csname thmt@contentsline@\thmt@envname\endcsname{%
    \thmt@contentslineShow% default:show
  }%
}
\def\thmt@allenvs{\@gobble}
\newcommand\thmt@recordenvname{%
  \edef\thmt@allenvs{\thmt@allenvs,\thmt@envname}%
}
\g@addto@macro\thmt@newtheorem@predefinition{%
  \thmt@mklistcmd
  \thmt@recordenvname
}

\addtotheorempostheadhook{%
  \thmtlo@chaptervspacehack
  \addcontentsline{loe}{\thmt@envname}{%
    \csname ll@\thmt@envname\endcsname
  }%
}

\newcommand\showtheorems[1]{%
  \@for\thm:=#1\do{%
    \typeout{showing \thm}%
    \@xa\let\csname thmt@contentsline@\thm\endcsname
      =\thmt@contentslineShow
  }%
}

\newcommand\ignoretheorems[1]{%
  \@for\thm:=#1\do{%
    \@xa\let\csname thmt@contentsline@\thm\endcsname
      =\thmt@contentslineIgnore
  }%
}
\newcommand\onlynamedtheorems[1]{%
  \@for\thm:=#1\do{%
    \global\@xa\let\csname thmt@contentsline@\thm\endcsname
      =\thmt@contentslineIfNamed
  }%
}

\AtBeginDocument{%
\@ifpackageloaded{hyperref}{%
  \let\thmt@hygobble\@gobble
}{%
  \let\thmt@hygobble\@empty
}
\let\thmt@contentsline\contentsline
}

\def\thmt@contentslineIgnore#1#2#3{%
  \thmt@hygobble
}
\def\thmt@contentslineShow{%
  \thmt@contentsline
}

\def\thmt@contentslineIfNamed#1#2#3{%
  \thmt@ifhasoptname #2\thmtformatoptarg\@nil{%
    \thmt@contentslineShow{#1}{#2}{#3}%
  }{%
    \thmt@contentslineIgnore{#1}{#2}{#3}%
    %\thmt@contentsline{#1}{#2}{#3}%
  }
}

\def\thmt@ifhasoptname #1\thmtformatoptarg#2\@nil{%
  \ifx\@nil#2\@nil
    \@xa\@secondoftwo
  \else
    \@xa\@firstoftwo
  \fi
}
```

A.1.5 Re-using environments

Only one environment is provided: `restatable`, which takes one optional and two mandatory arguments.

The first mandatory argument is the type of the theorem, i.e. if you want `\begin{lemma}` to be called on the inside, give `lemma`. The second argument is the name of the macro that the text should be stored in, for example `mylemma`. Be careful not to specify existing command names! The optional argument will become the optional argument to your theorem command. Consider the following example:

```
\documentclass{article}
\usepackage{amsmath, amsthm, thm-restate}
\newtheorem{lemma}{Lemma}
\begin{document}
  \begin{restatable}[Zorn]{lemma}{zornlemma}\label{thm:zorn}
    If every chain in $X$ is upper-bounded,
    $X$ has a maximal element.

    It's true, you know!
  \end{restatable}
  \begin{lemma}
    This is some other lemma of no import.
  \end{lemma}
  And now, here's Mr. Zorn again: \zornlemma*
\end{document}
```

which yields

Lemma 3 (Zorn). *If every chain in X is upper-bounded, X has a maximal element.*
It's true, you know!

Lemma 4. *This is some other lemma of no import.*

Actually, we have set a label in the environment, so we know that it's Lemma 3 on page 3. And now, here's Mr. Zorn again:

Lemma 3 (Zorn). *If every chain in X is upper-bounded, X has a maximal element.*
It's true, you know!

Since we prevent the label from being set again, we find that it's still Lemma 3 on page 3, even though it occurs later also.

As you can see, we use the starred form `\mylemma*`. As in many cases in LaTeX, the star means "don't give a number", since we want to retain the original number. There is also a starred variant of the `restatable` environment, where the first call doesn't determine the number, but a later call to `\mylemma` without star would. Since the number is carried around using LaTeX' `\label` machanism, you'll need a rerun for things to settle.

A.1.6 Restrictions

The only counter that is saved is the one for the theorem number. So, putting floats inside a restatable is not advised: they will appear in the LoF several times with new numbers. Equations should work, but the code handling them might turn out to be brittle, in particular when you add/remove hyperref. In the same vein, numbered equations within the statement appear again and are numbered again, with new numbers. (This is vaguely non-trivial to do correctly if equations are not numbered consecutively, but per-chapter, or there are multiple numbered equations.) Note that you cannot successfully reference the equations since all labels are disabled in the starred appearance. (The reference will point at the unstarred occurence.)

You cannot nest restatables either. You *can* use the `\restatable...\endrestatable` version, but everything up to the next matching `\end{...}` is scooped up. I've also probably missed many border cases.

```

\let\@xa\expandafter
\let\@nx\noexpand
\@ifundefined{c@thmt@dummyctr}{%
  \newcounter{thmt@dummyctr}%
  }{}
```

```
\gdef\theHthmt@dummyctr{dummy.\arabic{thmt@dummyctr}}%
\gdef\thethmt@dummyctr{}%
\long\def\thmt@collect@body#1#2\end#3{%
  \@xa\thmt@toks\@xa{\the\thmt@toks #2}%
  \def\thmttmpa{#3}%\def\thmttmpb{restatable}%
  \ifx\thmttmpa\@currenvir%thmttmpb
    \@xa\@firstoftwo% this is the end of the environment.
  \else
    \@xa\@secondoftwo% go on collecting
  \fi{% this is the end, my friend, drop the \end.
  % and call #1 with the collected body.
    \@xa#1\@xa{\the\thmt@toks}%
  }{% go on collecting
    \@xa\thmt@toks\@xa{\the\thmt@toks\end{#3}}%
    \thmt@collect@body{#1}%
  }%
}
```

A totally ignorant version of `\ref`, defaulting to #2 if label not known yet. Otherwise, return the formatted number.

```
\def\thmt@trivialref#1#2{%
  \ifcsname r@#1\endcsname
    \@xa\@xa\@xa\thmt@trivi@lr@f\csname r@#1\endcsname\relax\@nil
  \else #2\fi
}
\def\thmt@trivi@lr@f#1#2\@nil{#1}
```

Counter safeties: some counters' values should be stored, such as equation, so we don't get a new number. (We cannot reference it anyway.) We cannot store everything, though, think page counter or section number! There is one problem here: we have to remove all references to other counters from `\theequation`, otherwise your equation could get a number like (3.1) in one place and (4.1) in another section.

The best solution I can come up with is to override the usual macros that counter display goes through, to check if their argument is one that should be fully-expanded away or retained.

The following should only be called from within a group, and the sanitized `\thectr` must not be called from within that group, since it needs the original `\@arabic` et al.

```
\def\thmt@innercounters{%
  equation}
\def\thmt@counterformatters{%
  @alph,@Alph,@arabic,@roman,@Roman,@fnsymbol}

\@for\displ:=\thmt@counterformatters\do{%
  \@xa\let\csname thmt@\displ\@xa\endcsname\csname \displ\endcsname
}%
\def\thmt@sanitizethe#1{%
  \@for\displ:=\thmt@counterformatters\do{%
    \@xa\protected@edef\csname\displ\endcsname##1{%
      \@nx\ifx\@xa\@nx\csname c@#1\endcsname ##1%
        \@xa\protect\csname \displ\endcsname{##1}%
      \@nx\else
        \@nx\csname thmt@\displ\endcsname{##1}%
      \@nx\fi
    }%
  }%
  \expandafter\protected@edef\csname the#1\endcsname{\csname the#1\endcsname}%
  \ifcsname theH#1\endcsname
    \expandafter\protected@edef\csname theH#1\endcsname{\csname theH#1\endcsname}%
  \fi
}

\def\thmt@rst@storecounters#1{%
```

```
  \bgroup
        % ugly hack: save chapter,..subsection numbers
        % for equation numbers.
  \refstepcounter{thmt@dummyctr}% why is this here?
  \def\@currentlabel{}%
  \@for\ctr:=\thmt@innercounters\do{%
    \thmt@sanitizethe{\ctr}%
    \protected@edef\@currentlabel{%
      \@currentlabel
      \protect\def\@xa\protect\csname the\ctr\endcsname{\csname the\ctr\endcsname}%
      \ifcsname theH\ctr\endcsname
        \protect\def\@xa\protect\csname theH\ctr\endcsname{%
          (restate \protect\theHthmt@dummyctr)\csname theH\ctr\endcsname}%
      \fi
      \protect\setcounter{\ctr}{\number\csname c@\ctr\endcsname}%
    }%
  }%
  \label{thmt@@#1@data}%
  \egroup
}%
```

Now, the main business.

```
\newif\ifthmt@thisistheone
\newenvironment{thmt@restatable}[3][]{%
  \thmt@toks{}% will hold body
%
  \stepcounter{thmt@dummyctr}% used for data storage label.
%
  \long\def\thmrst@store##1{%
    \@xa\gdef\csname #3\endcsname{%
      \@ifstar{%
        \thmt@thisistheonefalse\csname thmt@stored@#3\endcsname
      }{%
        \thmt@thisistheonetrue\csname thmt@stored@#3\endcsname
      }%
    }%
    \@xa\long\@xa\gdef\csname thmt@stored@#3\@xa\endcsname\@xa{%
      \begingroup
      \ifthmt@thisistheone
        % these are the valid numbers, store them for the other
        % occasions.
        \thmt@rst@storecounters{#3}%
      \else
        % this one should use other numbers...
        % first, fake the theorem number.
        \@xa\protected@edef\csname the#2\endcsname{%
          \thmt@trivialref{thmt@@#3}{??}}%
        % if the number wasn't there, have a "re-run to get labels right"
        % warning.
        \ifcsname r@thmt@@#3\endcsname\else
          \G@refundefinedtrue
        \fi
        % prevent stepcountering the theorem number,
        % but still, have some number for hyperref, just in case.
        \@xa\let\csname c@#2\endcsname=\c@thmt@dummyctr
        \@xa\let\csname theH#2\endcsname=\theHthmt@dummyctr
        % disable labeling.
        \let\label=\@gobble
        \let\ltx@label=\@gobble% amsmath needs this
        % We shall need to restore the counters at the end
        % of the environment, so we get
      % (4.2) [(3.1 from restate)] (4.3)
      \def\thmt@restorecounters{}%
      \@for\ctr:=\thmt@innercounters\do{%
        \protected@edef\thmt@restorecounters{%
          \thmt@restorecounters
          \protect\setcounter{\ctr}{\arabic{\ctr}}%
        }%
      }%
      % pull the new semi-static definition of \theequation et al.
      % from the aux file.
      \thmt@trivialref{thmt@@#3@data}{}%
    \fi
    % call the proper begin-env code, possibly with optional argument
    \csname #2\@xa\endcsname\ifx\@nx#1\@nx\else[{#1}]\fi
    \ifthmt@thisistheone
      % store a label so we can pick up the number later.
      \label{thmt@@#3}%
    \fi
    % this will be the collected body.
    ##1
    \csname end#2\endcsname
    % if we faked the counter values, restore originals now.
    \ifthmt@thisistheone\else\thmt@restorecounters\fi
    \endgroup
  }% thmt@stored@#3
  % in either case, now call the just-created macro,
  \csname #3\@xa\endcsname\ifthmt@thisistheone\else*\fi
  % and artificially close the current environment.
  \@xa\end\@xa{\@currenvir}
 }% thm@rst@store
 \thmt@collect@body\thmrst@store
}{%
 %% now empty, just used as a marker.
}

\newenvironment{restatable}{%
 \thmt@thisistheonetrue\thmt@restatable
}{%
 \endthmt@restatable
}
\newenvironment{restatable*}{%
 \thmt@thisistheonefalse\thmt@restatable
}{%
 \endthmt@restatable
}
```

A.1.7 Fixing autoref and friends

hyperref's \autoref command does not work well with theorems that share a counter: it'll always think it's a Lemma even if it's a Remark that shares the Lemma counter. Load this package to fix it. No further intervention needed.

```

\RequirePackage{thm-patch, aliasctr, parseargs, keyval}

\let\@xa=\expandafter
\let\@nx=\noexpand

\newcommand\thmt@autorefsetup{%
 \@xa\def\csname\thmt@envname autorefname\@xa\endcsname\@xa{\thmt@thmname}%
 \ifthmt@hassibling
```

```
    \@counteralias{\thmt@envname}{\thmt@sibling}%
    \@xa\def\@xa\thmt@autoreffix\@xa{%
      \@xa\let\csname the\thmt@envname\@xa\endcsname
        \csname the\thmt@sibling\endcsname
      \def\thmt@autoreffix{}%
    }%
    \protected@edef\thmt@sibling{\thmt@envname}%
  \fi
}
\g@addto@macro\thmt@newtheorem@predefinition{\thmt@autorefsetup}%
\g@addto@macro\thmt@newtheorem@postdefinition{\csname thmt@autoreffix\endcsname}%

\def\thmt@refnamewithcomma #1#2#3,#4,#5\@nil{%
  \@xa\def\csname\thmt@envname #1utorefname\endcsname{#3}%
  \ifcsname #2refname\endcsname
    \csname #2refname\endcsname{\thmt@envname}{#3}{#4}%
  \fi
}
\define@key{thmdef}{refname}{\thmt@trytwice{}{%
  \thmt@refnamewithcomma{a}{c}#1,\textbf{?? (pl. #1)},\@nil
}}
\define@key{thmdef}{Refname}{\thmt@trytwice{}{%
  \thmt@refnamewithcomma{A}{C}#1,\textbf{?? (pl. #1)},\@nil
}}


\ifcsname Autoref\endcsname\else
\let\thmt@HyRef@testreftype\HyRef@testreftype
\def\HyRef@Testreftype#1.#2\\{%
  \ltx@IfUndefined{#1Autorefname}{%
    \thmt@HyRef@testreftype#1.#2\\%
  }{%
    \edef\HyRef@currentHtag{%
      \expandafter\noexpand\csname#1Autorefname\endcsname
      \noexpand~%
    }%
  }%
}


\let\thmt@HyPsd@@autorefname\HyPsd@@autorefname
\def\HyPsd@@Autorefname#1.#2\@nil{%
  \tracingall
  \ltx@IfUndefined{#1Autorefname}{%
    \thmt@HyPsd@@autorefname#1.#2\@nil
  }{%
    \csname#1Autorefname\endcsname\space
  }%
}%
\def\Autoref{%
  \parse{%
  {\parseFlag*{\def\thmt@autorefstar{*}}{\let\thmt@autorefstar\@empty}}%
  {\parseMand{%
    \bgroup
    \let\HyRef@testreftype\HyRef@Testreftype
    \let\HyPsd@@autorefname\HyPsd@@Autorefname
    \@xa\autoref\thmt@autorefstar{##1}%
    \egroup
    \let\@parsecmd\@empty
  }}%
  }%
```

```
}
\fi % ifcsname Autoref

% not entirely appropriate here, but close enough:
\AtBeginDocument{%
  \@ifpackageloaded{nameref}{%
    \addtotheorempostheadhook{%
      \expandafter\NR@gettitle\expandafter{\thmt@shortoptarg}%
  }}{}
}

\AtBeginDocument{%
  \@ifpackageloaded{cleveref}{%
    \@ifpackagelater{cleveref}{2010/04/30}{%
    % OK, new enough
    }{%
      \PackageWarningNoLine{thmtools}{%
        Your version of cleveref is too old!\MessageBreak
        Update to version 0.16.1 or later%
      }
    }
  }{}
}
```

A.2 Glue code for different backends

A.2.1 amsthm

```
\define@key{thmstyle}{spaceabove}{%
  \def\thmt@style@spaceabove{#1}%
}
\define@key{thmstyle}{spacebelow}{%
  \def\thmt@style@spacebelow{#1}%
}
\define@key{thmstyle}{headfont}{%
  \def\thmt@style@headfont{#1}%
}
\define@key{thmstyle}{bodyfont}{%
  \def\thmt@style@bodyfont{#1}%
}
\define@key{thmstyle}{notefont}{%
  \def\thmt@style@notefont{#1}%
}
\define@key{thmstyle}{headpunct}{%
  \def\thmt@style@headpunct{#1}%
}
\define@key{thmstyle}{notebraces}{%
  \def\thmt@style@notebraces{\thmt@embrace#1}%
}
\define@key{thmstyle}{break}[]{%
  \def\thmt@style@postheadspace{\newline}%
}
\define@key{thmstyle}{postheadspace}{%
  \def\thmt@style@postheadspace{#1}%
}
\define@key{thmstyle}{headindent}{%
  \def\thmt@style@headindent{#1}%
}

\newtoks\thmt@style@headstyle
```

```
\define@key{thmstyle}{headformat}[]{%
  \thmt@style@headstyle{%
    \def\NAME{\the\thm@headfont ##1}%
    \def\NUMBER{\bgroup\@upn{##2}\egroup}%
    \def\NOTE{\if=##3=\else\bgroup\ \the\thm@notefont(##3)\egroup\fi}%
  }%
  \def\thmt@tmp{#1}%
  \@onelevel@sanitize\thmt@tmp
  %\tracingall
  \ifcsname thmt@headstyle@\thmt@tmp\endcsname
    \thmt@style@headstyle\@xa{%
      \the\thmt@style@headstyle
      \csname thmt@headstyle@#1\endcsname
    }%
  \else
    \thmt@style@headstyle\@xa{%
      \the\thmt@style@headstyle
      #1
    }%
  \fi
  %\showthe\thmt@style@headstyle
}
% examples:
\def\thmt@headstyle@margin{%
  \makebox[0pt][r]{\NUMBER\ }\NAME\NOTE
}
\def\thmt@headstyle@swapnumber{%
  \NUMBER\ \NAME\NOTE
}



\def\thmt@embrace#1#2(#3){#1#3#2}

\def\thmt@declaretheoremstyle@setup{%
  \let\thmt@style@notebraces\@empty%
  \thmt@style@headstyle{}%
  \kvsetkeys{thmstyle}{%
    spaceabove=3pt,
    spacebelow=3pt,
    headfont=\bfseries,
    bodyfont=\normalfont,
    headpunct={.},
    postheadspace={ },
    headindent={},
    notefont={\fontseries\mddefault\upshape}
  }%
}
\def\thmt@declaretheoremstyle#1{%
  %\show\thmt@style@spaceabove
  \thmt@toks{\newtheoremstyle{#1}}%
  \thmt@toks\@xa\@xa\@xa{\@xa\the\@xa\thmt@toks\@xa{\thmt@style@spaceabove}}%
  \thmt@toks\@xa\@xa\@xa{\@xa\the\@xa\thmt@toks\@xa{\thmt@style@spacebelow}}%
  \thmt@toks\@xa\@xa\@xa{\@xa\the\@xa\thmt@toks\@xa{\thmt@style@bodyfont}}%
  \thmt@toks\@xa\@xa\@xa{\@xa\the\@xa\thmt@toks\@xa{\thmt@style@headindent}}% indent1 FIXM
  \thmt@toks\@xa\@xa\@xa{\@xa\the\@xa\thmt@toks\@xa{\thmt@style@headfont}}%
  \thmt@toks\@xa\@xa\@xa{\@xa\the\@xa\thmt@toks\@xa{\thmt@style@headpunct}}%
  \thmt@toks\@xa\@xa\@xa{\@xa\the\@xa\thmt@toks\@xa{\thmt@style@postheadspace}}%
  \thmt@toks\@xa\@xa\@xa{\@xa\the\@xa\thmt@toks\@xa{\the\thmt@style@headstyle}}% headspec I
  \the\thmt@toks
  %1 Indent amount: empty = no indent, \parindent = normal paragraph indent
  %2 Space after theorem head: { } = normal interword space; \newline = linebreak
  %% BUGFIX: amsthm ignores notefont setting altogether:
  \thmt@toks\@xa\@xa\@xa{\csname th@#1\endcsname}%
  \thmt@toks
  \@xa\@xa\@xa\@xa\@xa\@xa\@xa{%
  \@xa\@xa\@xa\@xa\@xa\@xa\@xa\thm@notefont
  \@xa\@xa\@xa\@xa\@xa\@xa\@xa{%
  \@xa\@xa\@xa\thmt@style@notefont
  \@xa\thmt@style@notebraces
  \@xa}\the\thmt@toks}%
  \@xa\def\csname th@#1\@xa\endcsname\@xa{\the\thmt@toks}%
%  \@xa\def\csname th@#1\@xa\@xa\@xa\@xa\@xa\@xa\@xa\endcsname
%    \@xa\@xa\@xa\@xa\@xa\@xa\@xa{%
%    \@xa\@xa\@xa\@xa\@xa\@xa\@xa\thm@notefont
%    \@xa\@xa\@xa\@xa\@xa\@xa\@xa{%
%    \@xa\@xa\@xa\thmt@style@notefont
%    \@xa\@xa\@xa\thmt@style@notebraces
%    \@xa\@xa\@xa}\csname th@#1\endcsname
%  }
}

\define@key{thmdef}{qed}[\qedsymbol]{%
  \thmt@trytwice{}{%
    \addtotheorempostheadhook[\thmt@envname]{%
      \pushQED{\qed}%
    }%
    \addtotheoremprefoothook[\thmt@envname]{%
      \protected@edef\qedsymbol{#1}%
      \popQED
    }%
  }%
}

\def\thmt@amsthmlistbreakhack{%
  \leavevmode
  \vspace{-\baselineskip}%
  \par
  \everypar{\setbox\z@\lastbox\everypar{}}%
}

\define@key{thmuse}{listhack}[\relax]{%
  \addtotheorempostheadhook[local]{%
    \thmt@amsthmlistbreakhack
  }%
}

```

A.2.2 beamer

```
\newif\ifthmt@hasoverlay
\def\thmt@parsetheoremargs#1{%
  \parse{%
    {\parseOpt<>{\thmt@hasoverlaytrue\def\thmt@overlay{##1}}{}}%
    {\parseOpt[]{\def\thmt@optarg{##1}}{%
      \let\thmt@shortoptarg\@empty
      \let\thmt@optarg\@empty}}%
    {\ifthmt@hasoverlay\expandafter\@gobble\else\expandafter\@firstofone\fi
        {\parseOpt<>{\thmt@hasoverlaytrue\def\thmt@overlay{##1}}{}}%
    }%
    {%
      \def\thmt@local@preheadhook{}%
      \def\thmt@local@postheadhook{}%
      \def\thmt@local@prefoothook{}%
      \def\thmt@local@postfoothook{}%
      \thmt@local@preheadhook
      \csname thmt@#1@preheadhook\endcsname
      \thmt@generic@preheadhook
      \protected@edef\tmp@args{%
        \ifthmt@hasoverlay <\thmt@overlay>\fi
        \ifx\@empty\thmt@optarg\else [{\thmt@optarg}]\fi
      }%
      \csname thmt@original@#1\@xa\endcsname\tmp@args
      \thmt@local@postheadhook
      \csname thmt@#1@postheadhook\endcsname
      \thmt@generic@postheadhook
      \let\@parsecmd\@empty
    }%
  }
}%
```

A.2.3 ntheorem

```

% actually, ntheorem's so-called style is nothing like a style at all...
\def\thmt@declaretheoremstyle@setup{}
\def\thmt@declaretheoremstyle#1{%
  \ifcsname th@#1\endcsname\else
    \@xa\let\csname th@#1\endcsname\th@plain
  \fi
}

\def\thmt@notsupported#1#2{%
  \PackageWarning{thmtools}{Key '#2' not supported by #1}{}%
}

\define@key{thmstyle}{spaceabove}{%
  \setlength\theorempreskipamount{#1}%
}
\define@key{thmstyle}{spacebelow}{%
  \setlength\theorempostskipamount{#1}%
}
\define@key{thmstyle}{headfont}{%
  \theoremheaderfont{#1}%
}
\define@key{thmstyle}{bodyfont}{%
  \theorembodyfont{#1}%
}
% not supported in ntheorem.
\define@key{thmstyle}{notefont}{%
  \thmt@notsupported{ntheorem}{notefont}%
}
\define@key{thmstyle}{headpunct}{%
  \theoremseparator{#1}%
}
% not supported in ntheorem.
\define@key{thmstyle}{notebraces}{%
  \thmt@notsupported{ntheorem}{notebraces}%
}
\define@key{thmstyle}{break}{%
  \theoremstyle{break}%
}
% not supported in ntheorem...
```

```
\define@key{thmstyle}{postheadspace}{%
  %\def\thmt@style@postheadspace{#1}%
  \@xa\g@addto@macro\csname thmt@style \thmt@style @defaultkeys\endcsname{%
      postheadhook={\hspace{-\labelsep}\hspace*{#1}},%
  }%
}

% not supported in ntheorem
\define@key{thmstyle}{headindent}{%
  \thmt@notsupported{ntheorem}{headindent}%
}
% sorry, only style, not def with ntheorem.
\define@key{thmstyle}{qed}[\qedsymbol]{%
  \@ifpackagewith{ntheorem}{thmmarks}{%
    \theoremsymbol{#1}%
  }{%
    \thmt@notsupported
      {ntheorem without thmmarks option}%
      {headindent}%
  }%
}

\let\@upn=\textup
\define@key{thmstyle}{headformat}[]{%
  \def\thmt@tmp{#1}%
  \@onelevel@sanitize\thmt@tmp
  %\tracingall
  \ifcsname thmt@headstyle@\thmt@tmp\endcsname
    \newtheoremstyle{\thmt@style}{%
      \item[\hskip\labelsep\theorem@headerfont%
        \def\NAME{\theorem@headerfont ####1}%
        \def\NUMBER{\bgroup\@upn{####2}\egroup}%
        \def\NOTE{}%
        \csname thmt@headstyle@#1\endcsname
        \theorem@separator
      ]
    }{%
      \item[\hskip\labelsep\theorem@headerfont%
        \def\NAME{\theorem@headerfont ####1}%
        \def\NUMBER{\bgroup\@upn{####2}\egroup}%
        \def\NOTE{\if=####3=\else\bgroup\ (####3)\egroup\fi}%
        \csname thmt@headstyle@#1\endcsname
        \theorem@separator
      ]
    }
  \else
    \newtheoremstyle{\thmt@style}{%
      \item[\hskip\labelsep\theorem@headerfont%
        \def\NAME{\the\thm@headfont ####1}%
        \def\NUMBER{\bgroup\@upn{####2}\egroup}%
        \def\NOTE{}%
        #1%
        \theorem@separator
      ]
    }{%
      \item[\hskip\labelsep\theorem@headerfont%
        \def\NAME{\the\thm@headfont ####1}%
        \def\NUMBER{\bgroup\@upn{####2}\egroup}%
        \def\NOTE{\if=####3=\else\bgroup\ (####3)\egroup\fi}%
        #1%
        \theorem@separator
      ]
    }
  \fi
}

\def\thmt@headstyle@margin{%
  \makebox[0pt][r]{\NUMBER\ }\NAME\NOTE
}
\def\thmt@headstyle@swapnumber{%
  \NUMBER\ \NAME\NOTE
}



```

A.3 Generic tools

A.3.1 A generalized argument parser

The main command provided by the package is `\parse{`*spec*`}`. *spec* consists of groups of commands. Each group should set up the command `\@parsecmd` which is then run. The important point is that `\@parsecmd` will pick up its arguments from the running text, not from the rest of *spec*. When it's done storing the arguments, `\@parsecmd` must call `\@parse` to continue with the next element of *spec*. The process terminates when we run out of spec.

Helper macros are provided for the three usual argument types: mandatory, optional, and flag.

```

\newtoks\@parsespec
\def\parse@endquark{\parse@endquark}
\newcommand\parse[1]{%
  \@parsespec{#1\parse@endquark}\@parse}

\newcommand\@parse{%
  \edef\p@tmp{\the\@parsespec}%
  \ifx\p@tmp\parse@endquark
    \expandafter\@gobble
  \else
%     \typeout{parsespec remaining: \the\@parsespec}%
    \expandafter\@firstofone
  \fi{%
    \@parsepop
  }%
}
\def\@parsepop{%
  \expandafter\p@rsepop\the\@parsespec\@nil
  \@parsecmd
}
\def\p@rsepop#1#2\@nil{%
  #1%
  \@parsespec{#2}%
}

\newcommand\parseOpt[4]{%
  %\parseOpt{openchar}{closechar}{yes}{no}
%   \typeout{attemping #1#2...}%
  \def\@parsecmd{%
    \@ifnextchar#1{\@@reallyparse}{#4\@parse}%
  }%
  \def\@@reallyparse#1##1#2{%
    #3\@parse
  }%
}

\newcommand\parseMand[1]{%
  %\parseMand{code}
  \def\@parsecmd##1{#1\@parse}%
}

\newcommand\parseFlag[3]{%
  %\parseFlag{flagchar}{yes}{no}
  \def\@parsecmd{%
    \@ifnextchar#1{#2\expandafter\@parse\@gobble}{#3\@parse}%
  }%
}
```

A.3.2 Different counters sharing the same register

`\@counteralias{#1}{#2}` makes #1 a counter that uses #2's count register. This is useful for things like hyperref's `\autoref`, which otherwise can't distinguish theorems and definitions if they share a counter.

For detailed information, see Die TeXnische Komödie 3/2006.

add `\@elt{#1}` to `\cl@#2`. This differs from the kernel implementation insofar as we trail the cl lists until we find one that is empty or starts with `\@elt`.

```
\def\aliasctr@f@llow#1#2\@nil#3{%
  \ifx#1\@elt
  \noexpand #3%
  \else
  \expandafter\aliasctr@f@llow#1\@elt\@nil{#1}%
  \fi
}
\newcommand\aliasctr@follow[1]{%
  \expandafter\aliasctr@f@llow
```

Don't be confused: the third parameter is ignored here, we always have recursion here since the *token* `\cl@#1` is (hopefully) not `\@elt`.

```
  \csname cl@#1\endcsname\@elt\@nil{\csname cl@#1\endcsname}%
}
\renewcommand*\@addtoreset[2]{\bgroup
   \edef\aliasctr@@truelist{\aliasctr@follow{#2}}%
  \let\@elt\relax
  \expandafter\@cons\aliasctr@@truelist{{#1}}%
\egroup}
```

This code has been adapted from David Carlisle's remreset. We load that here only to prevent it from being loaded again.

```
\RequirePackage{remreset}
\renewcommand*\@removefromreset[2]{\bgroup
  \edef\aliasctr@@truelist{\aliasctr@follow{#2}}%
  \expandafter\let\csname c@#1\endcsname\@removefromreset
  \def\@elt##1{%
    \expandafter\ifx\csname c@##1\endcsname\@removefromreset
    \else
      \noexpand\@elt{##1}%
    \fi}%
  \expandafter\xdef\aliasctr@@truelist{%
    \aliasctr@@truelist}
\egroup}
```

make #1 a counter that uses counter #2's count register.

```
\newcommand\@counteralias[2]{{%
```

```
    \def\@@gletover##1##2{%
      \expandafter\global
      \expandafter\let\csname ##1\expandafter\endcsname
      \csname ##2\endcsname
    }%
    \@ifundefined{c@#2}{\@nocounterr{#2}}{%
      \@ifdefinable{c@#1}{%
```

Four values make a counter foo:

- the count register accessed through `\c@foo`,
- the output macro `\thefoo`,
- the prefix macro `\p@foo`,
- the reset list `\cl@foo`.

hyperref adds `\theHfoo` in particular.

```
        \@@gletover{c@#1}{c@#2}%
        \@@gletover{the#1}{the#2}%
```

I don't see counteralias being called hundreds of times, let's just unconditionally create `\theHctr`-macros for hyperref.

```
        \@@gletover{theH#1}{theH#2}%
        \@@gletover{p@#1}{p@#2}%
        \expandafter\global
        \expandafter\def\csname cl@#1\expandafter\endcsname
        \expandafter{\csname cl@#2\endcsname}%
```

It is not necessary to save the value again: since we share a count register, we will pick up the restored value of the original counter.

```
        %\@addtoreset{#1}{@ckpt}%
      }%
    }%
}}
```

A.3.3 Tracking occurences: none, one or many

Two macros are provided: `\setuniqmark` takes a single parameter, the name, which should be a string of letters. `\ifuniqmark` takes three parameters: a name, a true-part and a false-part. The true part is executed if and only if there was exactly one call to `\setuniqmark` with the given name during the previous LaTeX run.

Example application: legal documents are often very strongly numbered. However, if a section has only a single paragraph, this paragraph is not numbered separately, this only occurs from two paragraphs onwards.

It's also possible to not-number the single theorem in your paper, but fall back to numbering when you add another one.

```

\DeclareOption{unq}{%
  \newwrite\uniq@channel
  \InputIfFileExists{\jobname.unq}{}{}%
  \immediate\openout\uniq@channel=\jobname.unq
  \AtEndDocument{%
    \immediate\closeout\uniq@channel%
  }
}
\DeclareOption{aux}{%
  \let\uniq@channel\@auxout
}

```

Call this with a name to set the corresponding uniqmark. The name must be suitable for `\csname`-constructs, i.e. fully expansible to a string of characters. If you use some counter values to generate this, it might be a good idea to try and use hyperref's `\theH...` macros, which have similar restrictions. You can check whether a particular `\setuniqmark` was called more than once during *the last run* with `\ifuniq`.

```
\newcommand\setuniqmark[1]{%
  \expandafter\ifx\csname uniq@now@#1\endcsname\relax
  \global\@namedef{uniq@now@#1}{\uniq@ONE}%
  \else
  \expandafter\ifx\csname uniq@now@#1\endcsname\uniq@MANY\else
  \immediate\write\uniq@channel{%
    \string\uniq@setmany{#1}%
  }%
  \ifuniq{#1}{%
    \uniq@warnnotunique{#1}%
  }{}%
  \fi
  \global\@namedef{uniq@now@#1}{\uniq@MANY}%
  \fi
}
```

Companion to `\setuniqmark`: if the uniqmark given in the first argument was called more than once, execute the second argument, otherwise execute the first argument. Note than no call to `\setuniqmark` for a particular uniqmark at all means that this uniqmark is unique.

This is a lazy version: we could always say false if we already had two calls to setuniqmark this run, but we have to rerun for any ifuniq prior to the first setuniqmark anyway, so why bother?

```
\newcommand\ifuniq[1]{%
  \expandafter\ifx\csname uniq@last@#1\endcsname\uniq@MANY
  \expandafter \@secondoftwo
  \else
  \expandafter\@firstoftwo
  \fi
}
```

Two quarks to signal if we have seen an uniqmark more than once.

```
\def\uniq@ONE{\uniq@ONE}
\def\uniq@MANY{\uniq@MANY}
```

Flag: suggest a rerun?

```
\newif\if@uniq@rerun
```

Helper macro: a call to this is written to the .aux file when we see an uniqmark for the second time. This sets the right information for the next run. It also checks on subsequent runs if the number of uniqmarks drops to less than two, so that we'll need a rerun.

```
\def\uniq@setmany#1{%
  \global\@namedef{uniq@last@#1}{\uniq@MANY}%
  \AtEndDocument{%
    \uniq@warnifunique{#1}%
  }%
}
```

Warning if something is unique now. This always warns if the setting for this run is not "many", because it was generated by a setmany from the last run.

```
\def\uniq@warnifunique#1{%
  \expandafter\ifx\csname uniq@now@#1\endcsname\uniq@MANY\else
  \PackageWarningNoLine{uniq}{%
    '#1' is unique now.\MessageBreak
    Rerun LaTeX to pick up the change%
  }%
  \@uniq@reruntrue
  \fi
}
```

Warning if we have a second uniqmark this run around. Since this is checked immediately, we could give the line of the second occurence, but we do not do so for symmetry.

```
\def\uniq@warnnotunique#1{%
  \PackageWarningNoLine{uniq}{%
    ‘#1’ is not unique anymore.\MessageBreak
    Rerun LaTeX to pick up the change%
  }%
  \@uniq@reruntrue
}
```

Maybe advise a rerun (duh!). This is executed at the end of the second reading of the aux-file. If you manage to set uniqmarks after that (though I cannot imagine why), you might need reruns without being warned, so don’t to that.

```
\def\uniq@maybesuggestrerun{%
  \if@uniq@rerun
  \PackageWarningNoLine{uniq}{%
    Uniquenesses have changed. \MessageBreak
    Rerun LaTeX to pick up the change%
  }%
  \fi
}
```

Make sure the check for rerun is pretty late in processing, so it can catch all of the uniqmarks (hopefully).

```
\AtEndDocument{%
  \immediate\write\@auxout{\string\uniq@maybesuggestrerun}%
}
\ExecuteOptions{aux}
\ProcessOptions\relax
```