stex-master.sty: sTeX 2.0[*]

Michael Kohlhase, Dennis Müller
FAU Erlangen-Nürnberg
http://kwarc.info/

November 17, 2020

Abstract

TODO

---

[*]Version v2.0 (last revised 2020/11/10)

Contents

1 Introduction

TODO

2 User commands

- ✓ \sTeX
- ✓ module
- ✓ \importmodule
- ✓ \usemodule
- ✓ \symdecl
- ✓ \notation
- ✓ verbalizations
- ? \inputref
- ? \libinput
- × \defi
- × \tref
- × omgroup/omtext

3 Implementation

```
1 ⟨∗package⟩
2 % TODO
3 \newif\if@modules@html@\@modules@html@true
4 \DeclareOption{omdocmode}{\@modules@html@false}
5 % Modules:
6 \newif\ifmod@show\mod@showfalse
7 \DeclareOption{showmods}{\mod@showtrue}
8 % sref:
9 \newif\ifextrefs\extrefsfalse
10 \DeclareOption{extrefs}{\extrefstrue}
11 %
12 \ProcessOptions

13 \RequirePackage{standalone}
14 \RequirePackage{xspace}
15 \RequirePackage{metakeys}
```

3.1 sTeX base

The sTeX logo:

```
16 \protected\def\stex{%
17    \@ifundefined{texorpdfstring}%
18    {\let\texorpdfstring\@firstoftwo}%
19    {}%
20    \texorpdfstring{\raisebox{-.5ex}S\kern-.5ex\TeX}{sTeX}\xspace%
21 }
22 \def\sTeX{\stex}
```

and a conditional for LaTeXML:

```
23 \newif\if@latexml\@latexmlfalse
```

3.2   Paths and URIs

```
24 \RequirePackage{xstring}
25 \RequirePackage{etoolbox}
```

\defpath   `\defpath[optional argument]{macro name}{base path}` defines a new macro which can take another path to formal one integrated path. For example, `\MathHub` in every `localpaths.tex` is defined as:

\defpath{MathHub}{/path/to/localmh/MathHub}

then we can use `\MathHub` to form other paths, for example,

\MathHub{source/smglom/sets}

will generate /path/to/localmh/MathHub/source/smglom/sets.

```
26 \newrobustcmd\defpath[3][]{%
27    \expandafter\newcommand\csname #2\endcsname[1]{#3/##1}%
28 }%
```

3.2.1   Path Canonicalization

We define two macros for changing the category codes of common characters in URIs, in particular #.

```
29 \def\pathsuris@setcatcodes{%
30    \edef\pathsuris@oldcatcode@hash{\the\catcode'\#}%
31    \catcode'\#=12\relax%
32    \edef\pathsuris@oldcatcode@slash{\the\catcode'\/}%
33    \catcode'\/=12\relax%
34    \edef\pathsuris@oldcatcode@colon{\the\catcode'\:}%
35    \catcode'\:=12\relax%
36    \edef\pathsuris@oldcatcode@qm{\the\catcode'\?}%
37    \catcode'\?=12\relax%
38 }
39 \def\pathsuris@resetcatcodes{%
40    \catcode'\#\pathsuris@oldcatcode@hash\relax%
41    \catcode'\/\pathsuris@oldcatcode@slash\relax%
42    \catcode'\:\pathsuris@oldcatcode@colon\relax%
43    \catcode'\?\pathsuris@oldcatcode@qm\relax%
44 }
```

We define some macros for later comparison.

```
45 \def\@ToTop{..}
46 \def\@Slash{/}
47 \def\@Colon{:}
48 \def\@Space{ }
49 \def\@QuestionMark{?}
50 \def\@Dot{.}
51 \catcode`\&=12
52 \def\@Ampersand{&}
53 \catcode`\&=4
54 \pathsuris@setcatcodes
55 \def\@Fragment{#}
56 \pathsuris@resetcatcodes
57 \catcode`\.=0
58 .catcode`.\=12
59 .let.@BackSlash\
60 .catcode`.\=0
61 \catcode`\.=12
62 \edef\old@percent@catcode{\the\catcode`\%}
63 \catcode`\%=12
64 \let\@Percent%
65 \catcode`\%=\old@percent@catcode
```

\@cpath   Canonicalizes (file) paths:

```
66 \def\@cpath#1{%
67     \edef\pathsuris@cpath@temp{#1}%
68     \def\@CanPath{}%
69     \IfBeginWith\pathsuris@cpath@temp\@Slash{%
70       \@cpath@loop%
71       \edef\@CanPath{\@Slash\@CanPath}%
72     }{%
73         \IfBeginWith\pathsuris@cpath@temp{\@Dot\@Slash}{%
74             \StrGobbleLeft\pathsuris@cpath@temp2[\pathsuris@cpath@temp]%
75             \@cpath@loop%
76         }{%
77             \ifx\pathsuris@cpath@temp\@Dot\else%
78             \@cpath@loop\fi%
79         }%
80     }%
81     \IfEndWith\@CanPath\@Slash{%
82       \ifx\@CanPath\@Slash\else%
83         \StrGobbleRight\@CanPath1[\@CanPath]%
84       \fi%
85     }{}%
86 }
87
88 \def\@cpath@loop{%
89     \IfSubStr\pathsuris@cpath@temp\@Slash{%
90         \StrCut\pathsuris@cpath@temp\@Slash\pathsuris@cpath@temp@a\pathsuris@cpath@temp%
```

```
 91            \ifx\pathsuris@cpath@temp@a\@ToTop%
 92                \ifx\@CanPath\@empty%
 93                    \edef\@CanPath{\@ToTop}%
 94                \else%
 95                    \edef\@CanPath{\@CanPath\@Slash\@ToTop}%
 96                \fi%
 97                \@cpath@loop%
 98            \else%
 99            \ifx\pathsuris@cpath@temp@a\@Dot%
100                \@cpath@loop%
101            \else%
102            \IfBeginWith\pathsuris@cpath@temp\@ToTop{%
103                \StrBehind{\pathsuris@cpath@temp}{\@ToTop}[\pathsuris@cpath@temp]%
104                \IfBeginWith\pathsuris@cpath@temp\@Slash{%
105                    \edef\pathsuris@cpath@temp{\@CanPath\pathsuris@cpath@temp}%
106                }{%
107                    \ifx\@CanPath\@empty\else%
108                        \edef\pathsuris@cpath@temp{\@CanPath\@Slash\pathsuris@cpath@temp}
109                    \fi%
110                }%
111                \def\@CanPath{}%
112                \@cpath@loop%
113            }{%
114                \ifx\@CanPath\@empty%
115                    \edef\@CanPath{\pathsuris@cpath@temp@a}%
116                \else%
117                    \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp@a}%
118                \fi%
119                \@cpath@loop
120            }%
121            \fi\fi%
122        }{
123            \ifx\@CanPath\@empty%
124                \edef\@CanPath{\pathsuris@cpath@temp}%
125            \else%
126                \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp}%
127            \fi%
128        }%
129 }
```
Test:

path	canonicalized path	expected
aaa	aaa	aaa
../../aaa	../../aaa	../../aaa
aaa/bbb	aaa/bbb	aaa/bbb
aaa/..		
../../aaa/bbb	../../aaa/bbb	../../aaa/bbb
../aaa/../bbb	../bbb	../bbb
../aaa/bbb	../aaa/bbb	../aaa/bbb
aaa/bbb/../ddd	aaa/ddd	aaa/ddd
aaa/bbb/./ddd	aaa/bbb/ddd	aaa/bbb/ddd
./		
aaa/bbb/../..		

\cpath   Implement `\cpath` to print the canonicalized path.

```
130 \newcommand\cpath[1]{%
131     \@cpath{#1}%
132     \@CanPath%
133 }
```

\path@filename

```
134 \def\path@filename#1#2{%
135     \edef\filename@oldpath{#1}%
136     \StrCount\filename@oldpath\@Slash[\filename@lastslash]%
137     \ifnum\filename@lastslash>0%
138         \StrBehind[\filename@lastslash]\filename@oldpath\@Slash[\filename@oldpath]%
139         \edef#2{\filename@oldpath}%
140     \else%
141         \edef#2{\filename@oldpath}%
142     \fi%
143 }
```

Test:
Path: /foo/bar/baz.tex
Filename: baz.tex

3.2.2   Windows

First, a conditional that tells us whether we have to use windows or unix file paths:

```
144 \newif\if@iswindows@\@iswindows@false
145 \IfFileExists{nul:}{\IfFileExists{/dev/null}{}{\@iswindows@true}}{}
```

Test:
We are on windows: no.

\windows@to@path   Converts a windows-style file path to a unix-style file path:

```
146 \newif\if@windowstopath@inpath@
147 \def\windows@to@path#1{
```

```
148     \@windowstopath@inpath@false
149     \def\windows@temp{}
150     \edef\windows@path{#1}
151     \ifx\windows@path\@empty\else
152         \expandafter\windows@path@loop\windows@path\windows@path@end
153     \fi
154     \let#1\windows@temp
155 }
156 \def\windows@path@loop#1#2\windows@path@end{
157     \def\windows@temp@b{#2}
158     \ifx\windows@temp@b\@empty
159         \def\windows@continue{}
160     \else
161         \def\windows@continue{\windows@path@loop#2\windows@path@end}
162     \fi
163     \if@windowstopath@inpath@
164         \ifx#1\@BackSlash
165             \edef\windows@temp{\windows@temp\@Slash}
166         \else
167             \edef\windows@temp{\windows@temp#1}
168         \fi
169     \else
170         \ifx#1:
171             \edef\windows@temp{\@Slash\windows@temp}
172             \@windowstopath@inpath@true
173         \else
174             \edef\windows@temp{\windows@temp#1}
175         \fi
176     \fi
177     \windows@continue
178 }
```

Test:
Input: C:\foo \bar .baz
Output: /C/foo/bar.baz

\path@to@windows Converts a unix-style file path to a windows-style file path:

```
179 \def\path@to@windows#1{
180     \@windowstopath@inpath@false
181     \def\windows@temp{}
182     \edef\windows@path{#1}
183     \edef\windows@path{\expandafter\@gobble\windows@path}
184     \ifx\windows@path\@empty\else
185         \expandafter\path@windows@loop\windows@path\windows@path@end
186     \fi
187     \let#1\windows@temp
188 }
189 \def\path@windows@loop#1#2\windows@path@end{
190     \def\windows@temp@b{#2}
191     \ifx\windows@temp@b\@empty
```

```
192          \def\windows@continue{}
193      \else
194          \def\windows@continue{\path@windows@loop#2\windows@path@end}
195      \fi
196      \if@windowstopath@inpath@
197          \ifx#1/
198              \edef\windows@temp{\windows@temp\@BackSlash}
199          \else
200              \edef\windows@temp{\windows@temp#1}
201          \fi
202      \else
203          \ifx#1/
204              \edef\windows@temp{\windows@temp:\@BackSlash}
205              \@windowstopath@inpath@true
206          \else
207              \edef\windows@temp{\windows@temp#1}
208          \fi
209      \fi
210      \windows@continue
211 }
```

Test:
Input: /C/foo/bar.baz
Output: C:\foo\bar.baz

3.2.3   Auxiliary methods

\trimstring   Removes initial and trailing spaces from a string:

```
212 \def\trimstring#1{%
213      \edef\pathsuris@trim@temp{#1}%
214      \IfBeginWith\pathsuris@trim@temp\@Space{%
215          \StrGobbleLeft\pathsuris@trim@temp1[#1]%
216          \trimstring{#1}%
217      }{%
218          \IfEndWith\pathsuris@trim@temp\@Space{%
219              \StrGobbleRight\pathsuris@trim@temp1[#1]%
220              \trimstring{#1}%
221          }{%
222              \edef#1{\pathsuris@trim@temp}%
223          }%
224      }%
225 }
```

Test:
»bla blubb«

\kpsewhich   Calls kpsewhich to get e.g. system variables:

```
226 \def\kpsewhich#1#2{\begingroup%
227   \edef\kpsewhich@cmd{"|kpsewhich #2"}%
228   \everyeof{\noexpand}%
```

```
229    \catcode'\\=12%
230    \edef#1{\@@input\kpsewhich@cmd\@Space}%
231    \trimstring#1%
232    \if@iswindows@\windows@to@path#1\fi%
233    \xdef#1{\expandafter\detokenize\expandafter{#1}}%
234 \endgroup}
```

Test:
/usr/share/texlive/texmf-dist/tex/latex/etoolbox/etoolbox.sty

3.2.4   sTEX input hooks

We determine the PWD of the current main document:

```
235 \edef\pwd@cmd{\if@iswindows@ -expand-var \percent CD\percent\else -var-value PWD\fi}
236 \kpsewhich\stex@maindir\pwd@cmd
237 \edef\stex@mainfile{\stex@maindir\@Slash\jobname}
238 \edef\stex@mainfile{\expandafter\detokenize\expandafter{\stex@mainfile}}
```

Test:
/home/jazzpirate/work/Software/ext/sTeX/sty/stex-master
      We keep a stack of \inputed files:

```
239 \def\stex@currfile@stack{}
240
241 \def\stex@currfile@push#1{%
242      \edef\stex@temppath{#1}%
243      \edef\stex@temppath{\expandafter\detokenize\expandafter{\stex@temppath}}%
244    \edef\stex@currfile@stack{\stex@currfile\ifx\stex@currfile@stack\@empty\else,\stex@currfile@s
245    \IfBeginWith\stex@temppath\@Slash{\@cpath{\stex@temppath}}{%
246      \@cpath{\stex@maindir\@Slash#1}%
247    }
248    \let\stex@currfile\@CanPath%
249    \path@filename\stex@currfile\stex@currfilename%
250    \StrLen\stex@currfilename[\stex@currfile@tmp]%
251    \StrGobbleRight\stex@currfile{\the\numexpr\stex@currfile@tmp+1 }[\stex@currpath]%
252    \global\let\stex@currfile\stex@currfile%
253    \global\let\stex@currpath\stex@currpath%
254    \global\let\stex@currfilename\stex@currfilename%
255 }
256 \def\stex@currfile@pop{%
257    \ifx\stex@currfile@stack\@empty%
258      \global\let\stex@currfile\stex@mainfile%
259      \global\let\stex@currpath\stex@maindir%
260      \global\let\stex@currfilename\jobname%
261    \else%
262      \StrCut\stex@currfile@stack,\stex@currfile\stex@currfile@stack%
263      \path@filename\stex@currfile\stex@currfilename%
264      \StrLen\stex@currfilename[\stex@currfile@tmp]%
265      \StrGobbleRight\stex@currfile{\the\numexpr\stex@currfile@tmp+1 }[\stex@currpath]%
266      \global\let\stex@currfile\stex@currfile%
267      \global\let\stex@currpath\stex@currpath%
```

```
268        \global\let\stex@currfilename\stex@currfilename%
269    \fi%
270 }
```

\stexinput Inputs a file by (if necessary) converting its path to a windows path first, and adding the file path to the input stack above:

```
271 \def\stexinput#1{%
272      \stexiffileexists{#1}{%
273        \stex@currfile@push\stex@temp@path%
274        \input{\stex@currfile}%
275        \stex@currfile@pop%
276      }%
277      {%
278          \PackageError{stex}{File does not exist (#1): \stex@temp@path}{}%
279      }%
280 }
281 \def\stexiffileexists#1#2#3{%
282    \edef\stex@temp@path{#1}%
283    \if@iswindows@\path@to@windows\stex@temp@path\fi%
284    \IfFileExists\stex@temp@path{#2}{#3}%
285 }
286 \stex@currfile@pop
```

Test:
This file: /home/jazzpirate/work/Software/ext/sTeX/sty/stex-master/stex-master
A test file: /home/jazzpirate/work/Software/ext/sTeX/sty/stex-master/testfile.tex

3.2.5 MathHub repositories

We read the `MATHHUB` system variable and set `\MathHub` accordingly:

```
287 \kpsewhich\mathhub@path{--var-value MATHHUB}
288 \if@iswindows@\windows@to@path\mathhub@path\fi
289 \ifx\mathhub@path\@empty%
290    \PackageWarning{stex}{MATHHUB system variable not found or wrongly set}{}
291    \defpath{MathHub}{}
292 \else\defpath{MathHub}\mathhub@path\fi
```

Test:
/home/jazzpirate/work/MathHub

\findmanifest \findmanifest{⟨*path*⟩} searches for a file `MANIFEST.MF` up and over ⟨*path*⟩ in the file system tree.

```
293 \def\findmanifest#1{
294    \@cpath{#1}
295    \ifx\@CanPath\@Slash
296      \def\manifest@mf{}
297    \else\ifx\@CanPath\@empty
298        \def\manifest@mf{}
299    \else
300      \edef\@findmanifest@path{\@CanPath/MANIFEST.MF}
```

```
301      \if@iswindows@\path@to@windows\@findmanifest@path\fi
302      \IfFileExists{\@findmanifest@path}{
303        %\message{MANIFEST.MF found at \@findmanifest@path}
304        \edef\manifest@mf{\@findmanifest@path}
305        \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
306      }{
307      \edef\@findmanifest@path{\@CanPath/META-INF/MANIFEST.MF}
308      \if@iswindows@\path@to@windows\@findmanifest@path\fi
309      \IfFileExists{\@findmanifest@path}{
310        %\message{MANIFEST.MF found at \@findmanifest@path}
311        \edef\manifest@mf{\@findmanifest@path}
312        \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
313      }{
314      \edef\@findmanifest@path{\@CanPath/meta-inf/MANIFEST.MF}
315      \if@iswindows@\path@to@windows\@findmanifest@path\fi
316      \IfFileExists{\@findmanifest@path}{
317        %\message{MANIFEST.MF found at \@findmanifest@path}
318        \edef\manifest@mf{\@findmanifest@path}
319        \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
320      }{
321        \findmanifest{\@CanPath/..}
322      }}}
323    \fi\fi
324 }
```

the next macro is a helper function for parsing `MANIFEST.MF`

```
325 \def\split@manifest@key{
326   \IfSubStr{\manifest@line}{\@Colon}{
327       \StrBefore{\manifest@line}{\@Colon}[\manifest@key]
328       \StrBehind{\manifest@line}{\@Colon}[\manifest@line]
329       \trimstring\manifest@line
330       \trimstring\manifest@key
331   }{
332       \def\manifest@key{}
333   }
334 }
```

the next helper function iterates over lines in `MANIFEST.MF`

```
335 \def\parse@manifest@loop{
336   \ifeof\@manifest
337   \else
338     \read\@manifest to \manifest@line\relax
339     \edef\manifest@line{\expandafter\detokenize\expandafter{\manifest@line}}
340     \split@manifest@key
341     % id
342     \IfStrEq\manifest@key{\detokenize{id}}{
343         \xdef\manifest@mf@id{\manifest@line}
```

12

```
344    }{
345    % narration-base
346    \IfStrEq\manifest@key{\detokenize{narration-base}}{
347        \xdef\manifest@mf@narr{\manifest@line}
348    }{
349    % namespace
350    \IfStrEq\manifest@key{\detokenize{source-base}}{
351        \xdef\manifest@mf@ns{\manifest@line}
352    }{
353    \IfStrEq\manifest@key{\detokenize{ns}}{
354        \xdef\manifest@mf@ns{\manifest@line}
355    }{
356    % dependencies
357    \IfStrEq\manifest@key{\detokenize{dependencies}}{
358        \xdef\manifest@mf@deps{\manifest@line}
359    }{
360    }}}}}
361    \parse@manifest@loop
362  \fi
363 }
```

\parsemanifest   \parsemanifest{⟨*macroname*⟩}{⟨*path*⟩} finds `MANIFEST.MF` via \findmanifest{⟨*path*⟩},
and parses the file, storing the individual fields (`id`, `narr`, `ns` and `dependencies`)
in ⟨*macroname*⟩`id`, ⟨*macroname*⟩`narr`, etc.

```
364 \newread\@manifest
365 \def\parsemanifest#1#2{%
366   \gdef\temp@archive@dir{}%
367   \findmanifest{#2}%
368   \begingroup%
369     \gdef\manifest@mf@id{}%
370     \gdef\manifest@mf@narr{}%
371     \gdef\manifest@mf@ns{}%
372     \gdef\manifest@mf@deps{}%
373     \openin\@manifest\manifest@mf%
374     \parse@manifest@loop%
375     \closein\@manifest%
376   \endgroup%
377   \if@iswindows@\windows@to@path\manifest@mf\fi%
378   \cslet{#1id}\manifest@mf@id%
379   \cslet{#1narr}\manifest@mf@narr%
380   \cslet{#1ns}\manifest@mf@ns%
381   \cslet{#1deps}\manifest@mf@deps%
382   \ifcsvoid{manifest@mf@id}{}{%
383     \cslet{#1dir}\temp@archive@dir%
384   }%
385 }
```

Test:
id: FOO/BAR
ns: http://mathhub.info/FOO/BAR

dir: FOO

`\setcurrentreposinfo` `\setcurrentreposinfo{`⟨*id*⟩`}` sets the current repository to ⟨*id*⟩, checks if the `MANIFEST.MF` of this repository has already been read, and if not, find it, parses it and stores the values in `\currentrepos@`⟨*key*⟩`@`⟨*id*⟩ for later retrieval.

```
386 \def\setcurrentreposinfo#1{%
387   \edef\mh@currentrepos{#1}%
388   \ifx\mh@currentrepos\@empty%
389     \edef\currentrepos@dir{\@Dot}%
390     \def\currentrepos@narr{}%
391     \def\currentrepos@ns{}%
392     \def\currentrepos@id{}%
393     \def\currentrepos@deps{}%
394   \else%
395   \ifcsdef{mathhub@dir@\mh@currentrepos}{%
396     \@inmhrepostrue
397     \edef\mh@currentrepos{#1}%
398     \expandafter\let\expandafter\currentrepos@dir\csname mathhub@dir@#1\endcsname%
399     \expandafter\let\expandafter\currentrepos@narr\csname mathhub@narr@#1\endcsname%
400     \expandafter\let\expandafter\currentrepos@ns\csname mathhub@ns@#1\endcsname%
401     \expandafter\let\expandafter\currentrepos@deps\csname mathhub@deps@#1\endcsname%
402   }{%
403     \parsemanifest{currentrepos@}{\MathHub{#1}}%
404     \@setcurrentreposinfo%
405     \ifcsvoid{currentrepos@dir}{\PackageError{stex}{No archive with %
406       name #1 found!}{make sure that #1 is directly in your MATHHUB folder %
407       and contains a MANIFEST.MF, either directly in #1 or in a meta-inf %
408       subfolder.}}{\@inmhrepostrue}%
409   }%
410   \fi%
411 }
412
413 \def\@setcurrentreposinfo{%
414   \edef\mh@currentrepos{\currentrepos@id}%
415   \ifcsvoid{currentrepos@dir}{}{%
416     \csxdef{mathhub@dir@\currentrepos@id}{\currentrepos@dir}%
417     \csxdef{mathhub@narr@\currentrepos@id}{\currentrepos@narr}%
418     \csxdef{mathhub@ns@\currentrepos@id}{\currentrepos@ns}%
419     \csxdef{mathhub@deps@\currentrepos@id}{\currentrepos@deps}%
420   }%
421 }
```

Finally – and that is the ultimate goal of all of the above, we set the current repos.

```
422 \newif\if@inmhrepos\@inmhreposfalse
423 \ifcsvoid{stex@maindir}{}{
424 \parsemanifest{currentrepos@}\stex@maindir
425 \@setcurrentreposinfo
426 \ifcsvoid{currentrepos@dir}{\PackageWarning{stex}{Not currently in a MathHub repository}{}}{%
427   \message{Current repository: \mh@currentrepos}
```

```
428 }
429 }
```

3.3 Modules

```
430 \if@latexml\else\ifmod@show\RequirePackage{mdframed}\fi\fi
```

Aux:
```
431 \def\ignorespacesandpars{\begingroup\catcode13=10\@ifnextchar\relax{\endgroup}{\endgroup}}
```

and more adapted from `http://tex.stackexchange.com/questions/179016/`
`ignore-spaces-and-pars-after-an-environment`
```
432 \def\ignorespacesandparsafterend#1\ignorespaces\fi{#1\fi\ignorespacesandpars}
433 \def\ignorespacesandpars{\ifmode\unskip\fi\@ifnextchar\par{\expandafter\ignorespacesandpars\@g
```

Options for the `module`-environment:
```
434 \addmetakey*{module}{title}
435 \addmetakey*{module}{name}
436 \addmetakey*{module}{creators}
437 \addmetakey*{module}{contributors}
438 \addmetakey*{module}{srccite}
439 \addmetakey*{module}{ns}
440 \addmetakey*{module}{narr}
```

module@heading We make a convenience macro for the module heading. This can be customized.
```
441 \ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%
442 \newrobustcmd\module@heading{%
443   \stepcounter{module}%
444   \ifmod@show%
445   \noindent{\textbf{Module} \thesection.\themodule [\module@name]}%
446   \sref@label@id{Module \thesection.\themodule [\module@name]}%
447     \ifx\module@title\@empty :\quad\else\quad(\module@title)\hfill\\\fi%
448   \fi%
449 }%
```

Test:
Module 3.1[Test]: Foo

module Finally, we define the begin module command for the module environment. Much of the work has already been done in the keyval bindings, so this is quite simple.
```
450 \newenvironment{module}[1][]{%
451   \begin{@module}[#1]%
452   \module@heading% make the headings
453   \ignorespacesandpars\parsemodule@maybesetcodes}{%
454   \end{@module}%
455   \ignorespacesafterend%
456 }%
457 \ifmod@show\surroundwithmdframed{module@om@common}\fi%
```

Some auxiliary methods:
```
458 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}
459 \def\addto@thismodule#1{%
```

15

```
460   \@ifundefined{this@module}{}{%
461     \expandafter\g@addto@macro@safe\this@module{#1}%
462   }%
463 }
464 \def\addto@thismodulex#1{%
465 \@ifundefined{this@module}{}{%
466   \edef\addto@thismodule@exp{#1}%
467   \expandafter\expandafter\expandafter\g@addto@macro@safe%
468   \expandafter\this@module\expandafter{\addto@thismodule@exp}%
469 }}
```

@module A variant of the `module` environment that does not create printed representations
(in particular no frames).

To compute the ⟨*uri*⟩ of a module, `\set@default@ns` computes the namespace,
if none is provided as an optional argument, as follows:

If the file of the module is `/some/path/file.tex` and we are not in a `MathHub`
repository, the namespace is `file:///some/path`.

If the file of the module is `/some/path/in/mathhub/repo/sitory/source/sub/file.tex`
and `repo/sitory` is an archive in the `MathHub` root, and the `MANIFEST.MF`
of `repo/sitory` declares a namespace `http://some.namespace/foo`, then the
namespace of the module is `http://some.namespace/foo/sub`.

```
470 \newif\ifarchive@ns@empty@\archive@ns@empty@false
471 \def\set@default@ns{%
472   \edef\@module@ns@temp{\stex@currpath}%
473   \if@iswindows@\windows@to@path\@module@ns@temp\fi%
474   \archive@ns@empty@false%
475   \ifcsvoid{mh@currentrepos}{\archive@ns@empty@true}%
476   {\expandafter\ifx\csname mathhub@ns@\mh@currentrepos\endcsname\@empty\archive@ns@empty@true\fi
477   }%
478   \ifarchive@ns@empty@%
479     \edef\@module@ns@tempuri{file\@Colon\@Slash\@Slash\@module@ns@temp}%
480   \else%
481     \edef\@module@filepath@temppath{\@module@ns@temp}%
482     \edef\@module@ns@tempuri{\csname mathhub@ns@\mh@currentrepos\endcsname}%
483     \edef\@module@archivedirpath{\csname mathhub@dir@\mh@currentrepos\endcsname\@Slash source}%
484     \edef\@module@archivedirpath{\expandafter\detokenize\expandafter{\@module@archivedirpath}}%
485     \IfBeginWith\@module@filepath@temppath\@module@archivedirpath{%
486       \StrLen\@module@archivedirpath[\ns@temp@length]%
487       \StrGobbleLeft\@module@filepath@temppath\ns@temp@length[\@module@filepath@temprest]%
488       \edef\@module@ns@tempuri{\@module@ns@tempuri\@module@filepath@temprest}%
489     }{}%
490   \fi%
491   \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]
492   \setkeys{module}{ns=\@module@ns@tempuri}%
493 }
```

Test:

file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master

16

If the module is not given a `name`, `\set@next@moduleid` computes one by enumeration, e.g. `module0`, `module1`, etc.

```
494 \def\set@next@moduleid{%
495   \unless\ifcsname namespace@\module@ns @unnamedmodules\endcsname%
496       \csgdef{namespace@\module@ns @unnamedmodules}{0}%
497   \fi%
498   \edef\namespace@currnum{\csname namespace@\module@ns @unnamedmodules\endcsname}%
499   \edef\module@temp@setidname{\noexpand\setkeys{module}{name=module\namespace@currnum}}%
500   \module@temp@setidname%
501   \csxdef{namespace@\module@ns @unnamedmodules}{\the\numexpr\namespace@currnum+1}%
502 }
```

Test:
module0
module1

Finally, the `@module` environment does the actual work, i.e. setting metakeys, computing namespace/id, defining `\this@module`, etc.

For a module with name $\langle name \rangle$ (`\module@name`) and uri $\langle uri \rangle$ (`\module@uri`), this defines the following macros:

- `\module@defs@`$\langle uri \rangle$ that acts as a repository for semantic macros of the current module. It will be called by `\importmodule` to activate them.

- We will add the internal forms of the semantic macros whenever `\symdef` is invoked. To do this, we will need an unexpended form `\this@module` that expands to `\module@defs@`$\langle uri \rangle$; we define it first and then initialize `\module@defs@`$\langle uri \rangle$ as empty.

- `\module@names@`$\langle uri \rangle$ will store all symbol names declared in this module.

- `\module@imports@`$\langle uri \rangle$ will store the URIs of all modules direclty included in this module

- `\`$\langle uri \rangle$ that expands to `\invoke@module{`$\langle uri \rangle$`}` (see below).

- `\Module`$\langle name \rangle$ that expands to `\`$\langle uri \rangle$.

If we are currently in a mathhub repository, this information will also be stored in `\module@defs@`$\langle uri \rangle$, so we can resolve includes properly when this module is activated.

```
503 \newenvironment{@module}[1][]{%
504   \metasetkeys{module}{#1}%
505   \ifcsvoid{module@name}{\let\module@name\module@id}{}% % TODO deprecate
506   \ifx\module@ns\@empty\set@default@ns\fi%
507   \ifx\module@narr\@empty%
508     \setkeys{module}{narr=\module@ns}%
509   \fi%
510   \ifcsvoid{module@name}{\set@next@moduleid}{}%
511   \let\module@id\module@name% % TODO deprecate
512   \edef\module@uri{\module@ns\@QuestionMark\module@name}%
```

17

```
513   \csgdef{module@names@\module@uri}{}%
514   \csgdef{module@imports@\module@uri}{}%
515   \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
516   \expandafter\global\expandafter\let\csname Module\module@name\expandafter\endcsname\csname\mod
517   \edef\this@module{%
518     \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
519   }%
520   \csdef{module@defs@\module@uri}{}%
521   \ifcsvoid{mh@currentrepos}{}{%
522     \@inmhrepostrue%
523     \addto@thismodulex{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\en
524       {\noexpand\mh@currentrepos}}%
525     \addto@thismodulex{\noexpand\setcurrentreposinfo{\mh@currentrepos}}%
526   }%
527 }{%
528   \if@inmhrepos%
529   \@inmhreposfalse%
530   \addto@thismodulex{\noexpand\setcurrentreposinfo{\expandafter\noexpand\csname mh@old@repos@\mo
531   \fi%
532 }%
```

Test:
Module 3.2[Foo]:
Name: Foo
URI: file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master?Foo
this@module: macro:->

    Test:
Faking a MathHub archive Foo/Bar with URI `http://foo.bar/baz`:
Module 3.3[Foo2]:
Name: Foo2
URI: http://foo.bar/baz?Foo2
this@module: macro:->\edef \mh@old@repos@http://foo.bar/baz?Foo2 {\mh@currentrepos }\setcurrentreposinfo {Foo/Bar}

    Test:
Removing the `/home/jazzpirate/work/MathHub/` system variable first:
Module 3.4[Foo]:
Name: Foo
URI: file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master?Foo
this@module: macro:->Faking a MathHub archive Foo/Bar with URI `http://foo.bar/baz`:
Module 3.5[Foo2]:
Name: Foo2
URI: http://foo.bar/baz?Foo2
this@module: macro:->\edef \mh@old@repos@http://foo.bar/baz?Foo2 {\mh@currentrepos }\setcurrentreposinfo {Foo/Bar}

A module with URI $\langle uri \rangle$ and id $\langle id \rangle$ creates two macros $\backslash\langle uri \rangle$ and \Module$\langle id \rangle$, that ultimately expand to \@invoke@module{$\langle uri \rangle$}. Currently, the only functionality is \@invoke@module{$\langle uri \rangle$}\@URI, which expands to the full uri of a module (i.e. via \Module$\langle id \rangle$\@URI). In the future, this macro can be

extended with additional functionality, e.g. accessing symbols in a macro for over-loaded (macro-)names.

```
533 \def\@URI{uri}
534 \def\@invoke@module#1#2{%
535   \ifx\@URI#2%
536     #1%
537   \else%
538     % TODO something else
539     #2%
540   \fi%
541 }
```

3.4 Inheritance

3.4.1 Selective Inclusion

The next great goal is to establish the `\requiremodules` macro, which reads an SₜᴇX file and processes all the module signature information in them, but does not produce any output. This is a tricky business, as we need to "parse" the modules and treat the module signature macros specially (we refer to this as "**sms mode**", since it is equivalent to what the – now deprecated – `sms` utility did).

In the following we introduce a lot of auxiliary functionality before we can define `\requiremodules`.

`\parsemodule@allow*` The first step is setting up a functionality for registering `\sTeX` macros and environments as part of a module signature.

```
542 \newif\if@smsmode\@smsmodefalse
543 \def\parsemodule@escapechar@allowed{true}
544 \def\parsemodule@allow#1{
545   \expandafter\let\csname parsemodule@allowedmacro@#1\endcsname\parsemodule@escapechar@allowed
546 }
547 \def\parsemodule@allowenv#1{
548   \expandafter\let\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed
549 }
550 \def\parsemodule@escapechar@beginstring{begin}
551 \def\parsemodule@escapechar@endstring{end}
```

and now we use that to actually register all the SₜᴇX functionality as relevant for `sms` mode.

```
552 \parsemodule@allow{symdef}
553 \parsemodule@allow{abbrdef}
554 \parsemodule@allow{importmodule}
555 \parsemodule@allowenv{module}
556 \parsemodule@allow{importmhmodule}
557 \parsemodule@allow{gimport}
558 \parsemodule@allowenv{modsig}
559 \parsemodule@allowenv{mhmodsig}
560 \parsemodule@allowenv{mhmodnl}
```

```
561 \parsemodule@allowenv{modnl}
562 \parsemodule@allow{symvariant}
563 \parsemodule@allow{symi}
564 \parsemodule@allow{symii}
565 \parsemodule@allow{symiii}
566 \parsemodule@allow{symiv}
567 \parsemodule@allow{notation}
568 \parsemodule@allow{verbalization}
569 \parsemodule@allow{symdecl}
570 %\parsemodule@allow{defi}
571 %\parsemodule@allow{defii}
572 %\parsemodule@allow{defiii}
573 %\parsemodule@allow{defiv}
574 %\parsemodule@allow{adefi}
575 %\parsemodule@allow{adefii}
576 %\parsemodule@allow{adefiii}
577 %\parsemodule@allow{adefiv}
578 %\parsemodule@allow{defis}
579 %\parsemodule@allow{defiis}
580 %\parsemodule@allow{defiiis}
581 %\parsemodule@allow{defivs}
582 %\parsemodule@allow{Defi}
583 %\parsemodule@allow{Defii}
584 %\parsemodule@allow{Defiii}
585 %\parsemodule@allow{Defiv}
586 %\parsemodule@allow{Defis}
587 %\parsemodule@allow{Defiis}
588 %\parsemodule@allow{Defiiis}
589 %\parsemodule@allow{Defivs}
```

To read external modules without producing output, `\requiremodules` redefines the `\`-character to be an *active* character that, instead of executing a macro, checks whether a macro name has been registered using `\parsemodule@allow` before selectively executing the corresponding macro or ignoring it. To produce the relevant code, we therefore define a macro `\@active@slash` that produces a `\`-character with category code 13 (*active*), as well as `\@open@brace` and `\@close@brace`, which produce open and closing braces with category code 12 (*other*).

```
590 \catcode'\.=0
591 .catcode'.\=13
592 .def.@active@slash{\}
593 .catcode'.<=1
594 .catcode'.>=2
595 .catcode'.{=12
596 .catcode'.}=12
597 .def.@open@brace<{>
598 .def.@close@brace<}>
599 .catcode'.\=0
600 \catcode'\.=12
```

```
601 \catcode'\{=1
602 \catcode'\}=2
603 \catcode'\<=12
604 \catcode'\>=12
```

The next two macros set and reset the category codes before/after sms mode.

```
605    \def\set@parsemodule@catcodes{%
606        \global\catcode'\\=13%
607        \global\catcode'\#=12%
608        \global\catcode'\{=12%
609        \global\catcode'\}=12%
610        \global\catcode'\$=12%$
611        \global\catcode'\^=12%
612        \global\catcode'\_=12%
613        \global\catcode'\&=12%
614        \expandafter\let\@active@slash\parsemodule@escapechar%
615    }
```

```
616    \def\reset@parsemodule@catcodes{%
617        \global\catcode'\\=0%
618        \global\catcode'\#=6%
619        \global\catcode'\{=1%
620        \global\catcode'\}=2%
621        \global\catcode'\$=3%$
622        \global\catcode'\^=7%
623        \global\catcode'\_=8%
624        \global\catcode'\&=4%
625    }
```

\parsemodule@maybesetcodes   Before a macro is executed in sms-mode, the category codes will be reset to normal, to ensure that all macro arguments are parsed correctly. Consequently, the macros need to set the category codes back to sms mode after having read all arguments iff the macro got executed in sms mode. \parsemodule@maybesetcodes takes care of that.

```
626    \def\parsemodule@maybesetcodes{%
627        \if@smsmode\set@parsemodule@catcodes\fi%
628    }
```

\parsemodule@escapechar   This macro gets called whenever a \-character occurs in sms mode. It is split into several macros that parse and store characters in \parsemodule@escape@currcs until a character with category code $\neq 11$ occurs (i.e. the macro name is complete), check whether the macro is allowed in sms mode, and then either ignore it or execute it after setting category codes back to normal. Special care needs to be taken to make sure that braces have the right category codes (1 and 2 for open and closing braces, respectively) when delimiting macro arguments.

Entry point:

```
629
630 \def\parsemodule@escapechar{%
631     \def\parsemodule@escape@currcs{}%
632     \parsemodule@escape@parse@nextchar@%
633 }%
```

The next macro simply reads the next character and checks whether it has category code 11. If so, it stores it in \parsemodule@escape@currcs. Otherwise, the macro name is complete, it stores the last character in \parsemodule@last@char and calls \parsemodule@escapechar@checkcs.

```
634 \long\def\parsemodule@escape@parse@nextchar@#1{%
635     \ifcat a#1\relax%
636         \edef\parsemodule@escape@currcs{\parsemodule@escape@currcs#1}%
637         \let\parsemodule@do@next\parsemodule@escape@parse@nextchar@%
638     \else%
639       \def\parsemodule@last@char{#1}%
640       \def\parsemodule@do@next{\parsemodule@escapechar@checkcs}%
641     \fi%
642     \parsemodule@do@next%
643 }
```

The next macro checks whether the currently stored macroname is allowed in sms mode. There are four cases that need to be considered: \begin, \end, allowed macros, and others. In the first two cases, we reinsert \parsemodule@last@char and continue with \parsemodule@escapechar@checkbeginenv or \parsemodule@escapechar@checkender, respectively, to check whether the environment being openend/closed is allowed in sms mode. In both cases, \parsemodule@last@char is an open brace with category code 12. In the third case, we need to check whether \parsemodule@last@char is an open brace, in which case we call \parsemodule@converttoproperbraces, otherwise, we set category codes to normal and execute the macro. In the fourth case, we just reinsert \parsemodule@last@char and continue.

```
644 \def\parsemodule@escapechar@checkcs{%
645     \ifx\parsemodule@escape@currcs\parsemodule@escapechar@beginstring%
646         \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@la
647     \else%
648         \ifx\parsemodule@escape@currcs\parsemodule@escapechar@endstring%
649          \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@la
650         \else%
651           \expandafter\ifx\csname parsemodule@allowedmacro@\parsemodule@escape@currcs\endcsnam
652               \parsemodule@escapechar@allowed%
653             \ifx\parsemodule@last@char\@open@brace%
654               \expandafter\let\expandafter\parsemodule@do@next@ii\csname\parsemodule@escape@cu
655               \edef\parsemodule@do@next{\noexpand\parsemodule@converttoproperbraces\@open@brac
656             \else%
657               \reset@parsemodule@catcodes%
658               \edef\parsemodule@do@next{\expandafter\noexpand\csname\parsemodule@escape@currc
659             \fi%
660           \else\def\parsemodule@do@next{\relax\parsemodule@last@char}\fi%
661         \fi%
```

```
662     \fi%
663     \parsemodule@do@next%
664 }
```

This macro simply takes an argument in braces (with category codes 12), reinserts it with "proper" braces (category codes 1 and 2), sets category codes back to normal and calls \parsemodule@do@next@ii, which has been \let as the macro to be executed.

```
665 \expandafter\expandafter\expandafter\def%
666 \expandafter\expandafter\expandafter\parsemodule@converttoproperbraces%
667 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
668     \reset@parsemodule@catcodes%
669     \parsemodule@do@next@ii{#1}%
670 }
```

The next two macros apply in the \begin and \end cases. They check whether the environment is allowed in sms mode, if so, open/close the environment, and otherwise do nothing.

Notably, \parsemodule@escapechar@checkendenv does not set category codes back to normal, since \end{environment} never takes additional arguments that need to be parsed anyway.

```
671 \expandafter\expandafter\expandafter\def%
672 \expandafter\expandafter\expandafter\parsemodule@escapechar@checkbeginenv%
673 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
674     \expandafter\ifx\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed%
675         \reset@parsemodule@catcodes%
676         \def\parsemodule@do@next{\begin{#1}}%
677     \else%
678         \def\parsemodule@do@next{#1}%
679     \fi%
680     \parsemodule@do@next%
681 }
682 \expandafter\expandafter\expandafter\def%
683 \expandafter\expandafter\expandafter\parsemodule@escapechar@checkendenv%
684 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
685     \expandafter\ifx\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed%
686         %\reset@parsemodule@catcodes%
687         \def\parsemodule@do@next{\end{#1}}%
688     \else%
689        \def\parsemodule@do@next{#1}%
690     \fi%
691     \parsemodule@do@next%
692 }
```

\@requiremodules   the internal version of \requiremodules for use in the *.aux file. We disable it at the end of the document, so that when the aux file is read again, nothing is loaded.

```
693 \newrobustcmd\@requiremodules[1]{%
694     \if@tempswa\requiremodules{#1}\fi%
```

```
695 }%
```

`\requiremodules` This macro loads the module signatures in a file using the `\requiremodules@smsmode` above. We set the flag `\mod@showfalse` in the local group, so that the macros know now to pollute the result.

```
696   \newrobustcmd\requiremodules[1]{%
697     \mod@showfalse%
698     \edef\mod@path{#1}%
699     \edef\mod@path{\expandafter\detokenize\expandafter{\mod@path}}%
700     \requiremodules@smsmode{#1}%
701   }%
```

`\requiremodules@smsmode` this reads sTeX modules by setting the category codes for `sms` mode, `\inputting` the required file and wrapping it in a `\vbox` that gets stored away and ignored, in order to not produce any output. It also sets `\hbadness`, `\hfuzz` and friends to values that suppress overfull and underfull hbox messages.

```
702   \newbox\modules@import@tempbox
703   \def\requiremodules@smsmode#1{%
704     \setbox\modules@import@tempbox\vbox{%
705       \@smsmodetrue%
706       \set@parsemodule@catcodes%
707       \hbadness=100000\relax%
708       \hfuzz=10000pt\relax%
709       \vbadness=100000\relax%
710       \vfuzz=10000pt\relax%
711       \stexinput{#1.tex}%
712       \reset@parsemodule@catcodes%
713     }%
714     \parsemodule@maybesetcodes%
715   }
```

Test:
parsing `FOO/testmodule.tex`
macro:->\@invoke@module {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master/FOO?testmodule}

3.4.2   importmodule

`\importmodule@bookkeeping`

```
716 \newif\if@importmodule@switchrepos\@importmodule@switchreposfalse
717 \def\importmodule@bookkeeping#1#2#3{%
718   \@importmodule@switchreposfalse%
719   \metasetkeys{importmodule}{#1}%
720   \ifcsvoid{importmodule@mhrepos}{%
721     \ifcsvoid{currentrepos@dir}{%
722       \let\importmodule@dir\stex@maindir%
723     }{%
724       \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
725     }%
```

```
726  }{%
727    \@importmodule@switchrepostrue%
728    \expandafter\let\csname importmodule@oldrepos@#2\endcsname\mh@currentrepos%
729    \setcurrentreposinfo\importmodule@mhrepos%
730    \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
731  }%
732  \StrCut{#2}\@QuestionMark\importmodule@subdir\importmodule@modulename%
733  \ifx\importmodule@modulename\@empty%
734    \let\importmodule@modulename\importmodule@subdir%
735    \let\importmodule@subdir\@empty%
736  \else%
737    \ifx\importmodule@subdir\@empty\else%
738      \edef\importmodule@dir{\importmodule@dir\@Slash\importmodule@subdir}%
739    \fi%
740  \fi%
741  \begingroup#3\endgroup%
742  \if@importmodule@switchrepos%
743    \expandafter\setcurrentreposinfo\csname importmodule@oldrepos@#2\endcsname%
744  \fi%
745  \ignorespacesandpars%
746 }
```

\importmodule

```
747 %\srefaddidkey{importmodule}
748 \addmetakey{importmodule}{mhrepos}
749 \newcommand\importmodule[2][]{\@@importmodule[#1]{#2}{export}}
750 \newcommand\@@importmodule[3][]{%
751   \importmodule@bookkeeping{#1}{#2}{%
752     \@importmodule[\importmodule@dir]\importmodule@modulename{#3}%
753   }%
754 }
```

\@importmodule \@importmodule[⟨*filepath*⟩]{⟨*mod*⟩}{⟨*export?*⟩} loads ⟨*filepath*⟩.tex and acti-
vates the module ⟨*mod*⟩. If ⟨*export?*⟩ is export, then it also re-exports the
\symdefs from ⟨*mod*⟩.

First \@load will store the base file name with full path, then check if
\module@⟨*mod*⟩@path is defined. If this macro is defined, a module of this name
has already been loaded, so we check whether the paths coincide, if they do, all
is fine and we do nothing otherwise we give a suitable error. If this macro is
undefined we load the path by \requiremodules.

```
755 \newcommand\@importmodule[3][]{%
756 {%
757   \edef\@load{#1}%
758   \edef\@importmodule@name{#2}
759   \if@smsmode\else\ifcsvoid{Module\@importmodule@name}{%
760     \stexiffileexists\@load{\requiremodules\@load}{%
761       \requiremodules{\@load\@Slash\@importmodule@name}%
762     }%
763   }{}\fi%
```

```
764    \ifx\@load\@empty\else%
765      {% TODO
766 %        \edef\@path{\csname module@#2@path\endcsname}%
767 %        \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do noth:
768 %        {\PackageError{stex}% else signal an error
769 %          {Module Name Clash\MessageBreak%
770 %            A module with name #2 was already loaded under the path "\@path"\MessageBreak%
771 %            The imported path "\@load" is probably a different module with the\MessageBreak%
772 %            same name; this is dangerous -- not importing}%
773 %          {Check whether the Module name is correct}%
774 %        }%
775      }%
776   \fi%
777   \global\let\@importmodule@load\@load%
778 }%
779 \edef\@export{#3}\def\@@export{export}%prepare comparison
780 %\ifx\@export\@@export\export@defs{#2}\fi% export the module
781 \ifx\@export\@@export\addto@thismodulex{%
782   \noexpand\@importmodule[\@importmodule@load]{#2}{noexport}%
783 }%
784 \if@smsmode\else
785 \ifcsvoid{this@module}{}{%
786   \ifcsvoid{module@imports@\module@uri}{
787     \csxdef{module@imports@\module@uri}{%
788       \csname Module#2\endcsname\@URI%
789     }%
790   }{%
791     \csxdef{module@imports@\module@uri}{%
792       \csname Module#2\endcsname\@URI,%
793       \csname module@imports@\module@uri\endcsname%
794     }%
795   }%
796 }%
797 \fi\fi%
798 \if@smsmode\else\activate@defs{#2}\fi% activate the module
799 }%
```

Test:

\importmodule {testmoduleimporta}:
macro:->\@invoke@module {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master?testmoduleimporta}
undefined

Test:

\importmodule {testmoduleimportb?importb}:
macro:->\@invoke@module {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master?importb}
macro:->\protect \bar

Test:

macro:->\@invoke@module {http://mathhub.info/smglom/algebra?band}

macro:->\@invoke@module {http://mathhub.info/smglom/algebra?idempotent} undefined
macro:->\@ifstar \@gimport@star \@gimport@nostar

\activate@defs   To activate the \symdefs from a given module ⟨*mod*⟩, we call the macro \module@defs@⟨*mod*⟩. But to make sure that every module is activated only once, we only activate if the macro \module@defs@⟨*mod*⟩ is undefined, and define it directly afterwards to prohibit further activations.

```
800 \def\activate@defs#1{%
801   \ifcsundef{Module#1}{
802     \PackageError{stex}{No module with name #1 loaded}{Probably missing an
803       \detokenize{\importmodule} (or variant) somewhere?
804     }
805   }{%
806     \ifcsundef{module@\csname Module#1\endcsname\@URI @activated}%
807       {\csname module@defs@\csname Module#1\endcsname\@URI\endcsname}{}%
808     \@namedef{module@\csname Module#1\endcsname\@URI @activated}{true}%
809   }%
810 }%
```

\usemodule   \usemodule acts like \importmodule, except that it does not re-export the semantic macros in the modules it loads.

```
811 \newcommand\usemodule[2][]{\@@importmodule[#1]{#2}{noexport}}
```

Test:
Module 3.26[Foo]:
Module 3.27[Bar]: undefined
Module 3.28[Baz]:   undefined
macro:->\protect \bar

\inputref@*skip   hooks for spacing customization, they are empty by default.

```
812 \def\inputref@preskip{}
813 \def\inputref@postskip{}
```

\inputref   \inputref{⟨*path to the current file without extension*⟩} supports both absolute path and relative path, meanwhile, records the path and the extension (not for relative path).

```
814 \newrobustcmd\inputref[2][]{%
815   \importmodule@bookkeeping{#1}{#2}{%
816     %\inputreftrue
817     \inputref@preskip%
818     \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
819     \inputref@postskip%
820   }%
821 }%
```

3.5   Symbols/Notations/Verbalizations

\if@symdeflocal   A flag whether a symbol declaration is local (i.e. does not get exported) or not.

```
822 \newif\if@symdeflocal\@symdeflocalfalse
```

\define@in@module   calls \edef\#1{#2} and adds the macro definition to \this@module

```
823 \def\define@in@module#1#2{
824   \expandafter\edef\csname #1\endcsname{#2}%
825   \edef\define@in@module@temp{%
826     \def\expandafter\noexpand\csname#1\endcsname%
827     {#2}%
828   }%
829   \if@symdeflocal\else%
830     \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
831     \expandafter\endcsname\expandafter{\define@in@module@temp}%
832   \fi%
833 }
```

\symdecl   \symdecl[name=foo]{bar} Declares a new symbol in the current module with URI ⟨*module-uri*⟩?foo and defines new macros \⟨*uri*⟩ and \bar. If no optional name is given, bar is used as a name.

```
834 \addmetakey{symdecl}{name}%
835 \addmetakey{symdecl}{verbalization}%
836
837 % constructs a symbol name and a verbalization by splitting at exclamation
838 % points - e.g. \symdecl{symmetric!group} leads to name=symmetric-group
839 % and verbalization "symmetric group".
840 \def\symdecl@constructname#1{%
841   \def\symdecl@name{}%
842   \def\symdecl@verb{}%
843   \edef\symdecl@tempname{#1}%
844   \symdecl@constructname@loop%
845 }
846
847 \def\symdecl@constructname@loop{%
848   \ifx\symdecl@tempname\@empty\else%
849     \StrCut\symdecl@tempname!\symdecl@tempfirst\symdecl@tempname%
850     \ifx\symdecl@name\@empty%
851       \let\symdecl@name\symdecl@tempfirst%
852       \let\symdecl@verbalization\symdecl@tempfirst%
853       \symdecl@constructname@loop%
854     \else%
855       \edef\symdecl@name{\symdecl@name-\symdecl@tempfirst}%
856       \edef\symdecl@verbalization{\symdecl@verbalization\@Space\symdecl@tempfirst}%
857       \symdecl@constructname@loop%
858     \fi%
859   \fi%
860 }
861
862 \newcommand\symdecl[2][]{%
863   \ifcsdef{this@module}{%
864     \metasetkeys{symdecl}{#1}%
```

```
865    \ifcsvoid{symdecl@name}{%
866      \ifcsvoid{symdecl@verbalization}{%
867        \symdecl@constructname{#2}%
868      }{%
869        \edef\symdecl@name{#2}%
870      }%
871    }{%
872      \ifcsvoid{symdecl@verbalization}{\edef\symdecl@verbalization{#2}}{}%
873    }%
874    \edef\symdef@uri{\module@uri\@QuestionMark\symdecl@name}%
875    \ifcsvoid{\symdef@uri}{
876      \ifcsvoid{module@names@\module@uri}{%
877        \csxdef{module@names@\module@uri}{\symdecl@name}%
878      }{%
879        \csxdef{module@names@\module@uri}{\symdecl@name,%
880          \csname module@names@\module@uri\endcsname}%
881      }%
882    }{%
883    % not compatible with circular dependencies, e.g. test/omdoc/07-modules/smstesta.tex
884      \PackageWarning{stex}{symbol already defined: \symdef@uri}{%
885        You need to pick a fresh name for your symbol%
886      }%
887    }%
888    \define@in@module\symdef@uri{\noexpand\@invoke@symbol{\symdef@uri}}%
889    \define@in@module{#2}{\noexpand\@invoke@symbol{\symdef@uri}}%
890    \global\expandafter\let\csname\symdef@uri\@Fragment verb\@Fragment\endcsname\symdecl@verbali
891  }{%
892    \PackageError{stex}{\detokenize{\symdecl} not in a module}{You need to be in a module%
893    in order to declare a new symbol}
894  }%
895    \if@insymdef@\else\parsemodule@maybesetcodes\fi%
896  }
```

Test:
Module 3.29[foo]: \symdecl {bar}
Yields: macro:->\@invoke@symbol {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master?foo?bar}

3.5.1   Notations

\modules@getURIfromName   This macro searches for the full URI given a symbol name and stores it in \notation@uri. Used by e.g. \notation[...]{foo}{...} to figure out what symbol foo refers to:

```
897 \def\modules@getURIfromName#1{%
898   \def\notation@uri{}%
899   \edef\modules@getURI@name{#1}%
900   \if@isuri\modules@getURI@name{%
901     \let\notation@uri\isuri@uri%
902   }{%
903     \ifcsvoid{this@module}{}{%
```

```
904        \expandafter\modules@getURIfromModule\expandafter{\module@uri}%
905        \ifx\notation@uri\@empty%
906          \edef\modules@getURI@modules{\csname module@imports@\module@uri\endcsname}%
907          \expandafter\@for\expandafter\@I\expandafter:\expandafter=\modules@getURI@modules\do{%
908            \ifx\notation@uri\@empty%
909              \expandafter\modules@getURIfromModule\expandafter{\@I}%
910            \fi%
911          }%
912        \fi%
913        \ifx\notation@uri\@empty%
914          \def\notation@extract@uri@currcs{}%
915          \notation@extracturifrommacro{#1}%
916        \fi%
917        \ifx\notation@uri\@empty%
918          \PackageError{stex}{No symbol with name, URI or macroname \detokenize{#1} found!}{}%
919        \fi%
920      }%
921    }%
922 }
923
924 \def\if@isuri#1#2#3{%
925    \StrCount{#1}\@QuestionMark[\isuri@number]%
926    \ifnum\isuri@number=1 %
927      \StrCut{#1}\@QuestionMark\@isuri@mod\@isuri@name%
928      \ifcsvoid{Module\@isuri@mod}{#3}{%
929        \edef\isuri@uri{\csname Module\@isuri@mod\endcsname\@URI\@QuestionMark\@isuri@name}%
930        #2%
931      }%
932    \else%
933      \ifnum\isuri@number=2 %
934        \edef\isuri@uri{#1}#2\else#3%
935      \fi%
936    \fi%
937 }
938
939 \def\modules@getURIfromModule#1{%
940    \edef\modules@getURI@names{\csname module@names@#1\endcsname}%
941    \expandafter\@for\expandafter\@I\expandafter:\expandafter=%
942    \modules@getURI@names\do{%
943      \ifx\notation@uri\@empty%
944        \ifx\@I\modules@getURI@name%
945          \edef\notation@uri{#1\@QuestionMark\@I}%
946        \fi%
947      \fi%
948    }%
949 }
950
951 % extracts the full URI from \foo or anything being \ifx-equal to \foo,
952 % by expanding until we reach \@invoke@symbol{<uri>}
953 \def\notation@extracturifrommacro#1{%
```

```
954    \ifcsvoid{#1}{}{%
955      \expandafter\let\expandafter\notation@extract@uri@nextcs\csname#1\endcsname%
956      \ifx\notation@extract@uri@nextcs\notation@extract@uri@currcs\else%
957        \let\notation@extract@uri@currcs\notation@extract@uri@nextcs%
958        \expandafter\notation@extract@uriII\notation@extract@uri@nextcs\notation@end%
959      \fi%
960    }%
961 }
962 \long\def\notation@extract@uriII#1#2\notation@end{%
963    \def\notation@extract@check@temp{#2}
964    \ifx\@invoke@symbol#1%
965      \edef\notation@uri{#2}%
966    \else%
967      \ifx\notation@extract@check@temp\@empty\else%
968        \expandafter\def\expandafter\notation@extract@uri@nextcs\expandafter{#1{#2}}%
969        \notation@extract@uri{notation@extract@uri@nextcs}%
970      \fi%
971    \fi%
972 }
```

\notation   Adds a new notation to a symbol foo, as in: \notation[lang=en,arity=0,variant=op]{foo}{...}
\notation[variant=bar]{foo}[2]{...} \notation[args=aia,prec=500;50x49x51]{foo}{#1 bla #2
    TODO with brackets, e.g. \notation[withbrackets={\langle,\rangle}]{foo}{...}

```
973 \newif\if@inverbalization\@inverbalizationfalse
974 % parses the first two arguments:
975 \providerobustcmd\notation[2][]{%
976    \edef\notation@first{#1}%
977    \edef\notation@second{#2}%
978    \notation@%
979 }
980
981 \providerobustcmd\verbalization{%
982    \@inverbalizationtrue%
983    \notation%
984 }
985
986 % parses the last two arguments
987 \newcommand\notation@[2][0]{%
988    \edef\notation@donext{\noexpand\notation@@[\notation@first]%
989      {\notation@second}[#1]}%
990    \notation@donext{#2}%
991 }
992
993 % parses the notation arguments and wraps them in
994 % \notation@assoc and \notation@argprec for flexary arguments and precedences
995 \def\notation@@[#1]#2[#3]#4{%
996    \modules@getURIfromName{#2}%
997    \notation@parse@params{#1}{#3}
998    \let\notation@curr@todo@args\notation@curr@args%
```

```
999    \def\notation@temp@notation{}%
1000   \StrLen\notation@curr@args[\notation@temp@arity]%
1001   \expandafter\renewcommand\expandafter\notation@temp@notation%
1002     \expandafter[\notation@temp@arity]{#4}%
1003   % precedence
1004   \IfSubStr\notation@curr@precs;{%
1005     \StrCut\notation@curr@precs;\notation@curr@prec\notation@curr@precs%
1006     \ifx\notation@curr@prec\@empty\def\notation@curr@prec{0}\fi%
1007   }{%
1008     \ifx\notation@curr@precs\@empty%
1009       \ifnum\notation@temp@arity=0\relax%
1010         \edef\notation@curr@prec{\infprec}%
1011       \else%
1012         \def\notation@curr@prec{0}%
1013       \fi%
1014     \else%
1015       \edef\notation@curr@prec{\notation@curr@precs}%
1016       \def\notation@curr@precs{}%
1017     \fi%
1018   }%
1019   % arguments
1020   \def\notation@curr@extargs{}
1021   \def\notation@nextarg@index{1}%
1022   \notation@do@args%
1023 }
1024
1025 % parses additional notation components for (associative) arguments
1026 \def\notation@do@args{%
1027   \def\notation@nextarg@temp{}%
1028   \ifx\notation@curr@todo@args\@empty%
1029     \notation@after%
1030   \else%
1031     % argument precedence
1032     \IfSubStr\notation@curr@precs{x}{%
1033       \StrCut\notation@curr@precs{x}\notation@curr@argprec\notation@curr@precs%
1034     }{%
1035       \edef\notation@curr@argprec{\notation@curr@precs}%
1036       \def\notation@curr@precs{}%
1037     }%
1038     \ifx\notation@curr@argprec\@empty%
1039       \let\notation@curr@argprec\notation@curr@prec%
1040     \fi%
1041     \StrChar\notation@curr@todo@args1[\notation@argchar]%
1042     \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1043     \expandafter\ifx\notation@argchar i%
1044       % normal argument
1045       \edef\notation@nextarg@temp{{\noexpand\notation@argprec{\notation@curr@argprec}{########\x
1046       \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }
1047       \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
1048         \expandafter{\notation@nextarg@temp}%
```

32

```
1049      \expandafter\expandafter\expandafter\notation@do@args%
1050    \else%
1051      % associative argument
1052      \expandafter\expandafter\expandafter\notation@parse@assocarg%
1053    \fi%
1054  \fi%
1055 }
1056
1057 \def\notation@parse@assocarg#1{%
1058   \edef\notation@nextarg@temp{{\noexpand\notation@argprec{\notation@curr@argprec}{\noexpand\nota
1059   \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }%
1060   \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
1061   \expandafter{\notation@nextarg@temp}%
1062   \notation@do@args%
1063 }
1064
1065 \protected\def\safe@newcommand#1{%
1066   \ifdefined#1\expandafter\renewcommand\else\expandafter\newcommand\fi#1%
1067 }
1068
1069 % finally creates the actual macros
1070 \def\notation@after{
1071   \let\ex\expandafter%
1072   \ex\ex\ex\def\ex\ex\ex\notation@temp@notation\ex\ex\ex%
1073     {\ex\notation@temp@notation\notation@curr@extargs}%
1074   \edef\notation@temp@notation{\noexpand\notation@symprec{\notation@curr@prec}{\ex\unexpanded\ex
1075   \def\notation@temp@fragment{}%
1076   \ifx\notation@curr@arity\@empty\else%
1077     \edef\notation@temp@fragment{arity=\notation@curr@arity}
1078   \fi%
1079   \ifx\notation@curr@lang\@empty\else%
1080     \ifx\notation@temp@fragment\@empty%
1081       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1082     \else%
1083       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}
1084     \fi%
1085   \fi%
1086   \ifx\notation@curr@variant\@empty\else%
1087     \ifx\notation@temp@fragment\@empty%
1088       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1089     \else%
1090       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1091     \fi%
1092   \fi%
1093   \if@inverbalization\@inverbalizationfalse\verbalization@final%
1094   \else\notation@final\fi%
1095   \parsemodule@maybesetcodes%
1096 }
1097
1098 \def\notation@final{%
```

```
1099    \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1100    \ifcsvoid{\notation@csname}{%
1101      \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1102        \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@temp@arity\ex]%
1103        \ex{\notation@temp@notation}%
1104      \edef\symdecl@temps{%
1105        \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@temp@ari
1106      }%
1107      \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1108      \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1109    }{%
1110      \PackageWarning{stex}{notation already defined: \notation@csname}{%
1111        Choose a different set of notation options (variant,lang,arity)%
1112      }%
1113    }%
1114 }
1115
1116 \def\verbalization@final{%
1117    \edef\notation@csname{\notation@uri\@Fragment verb\@Fragment\notation@temp@fragment}%
1118    \ifcsvoid{\notation@csname}{%
1119      \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1120        \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@temp@arity\ex]%
1121        \ex{\notation@temp@notation}%
1122      \edef\symdecl@temps{%
1123        \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@temp@ari
1124      }%
1125      \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1126      \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1127    }{%
1128      \PackageWarning{stex}{verbalization already defined: \notation@csname}{%
1129        Choose a different set of verbalization options (variant,lang,arity)%
1130      }%
1131    }%
1132 }
1133
1134 % parses optional parameters
1135 \def\notation@parse@params#1#2{%
1136    \def\notation@curr@precs{}%
1137    \def\notation@curr@args{}%
1138    \def\notation@curr@variant{}%
1139    \def\notation@curr@arity{}%
1140    \def\notation@curr@provided@arity{#2}
1141    \def\notation@curr@lang{}%
1142    \def\notation@options@temp{#1}
1143    \notation@parse@params@%
1144    \ifx\notation@curr@args\@empty%
1145      \ifx\notation@curr@provided@arity\@empty%
1146        \notation@num@to@ia\notation@curr@arity%
1147      \else%
1148        \notation@num@to@ia\notation@curr@provided@arity%
```

34

```
1149      \fi%
1150    \fi%
1151 }
1152 \def\notation@parse@params@{%
1153    \IfSubStr\notation@options@temp,{%
1154      \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
1155      \notation@parse@param%
1156      \notation@parse@params@%
1157    }{\ifx\notation@options@temp\@empty\else%
1158      \let\notation@option@temp\notation@options@temp%
1159      \notation@parse@param%
1160    \fi}%
1161 }
1162
1163 %parses an individual optional argument/key-value-pair
1164 \def\notation@parse@param{%
1165    \trimstring\notation@option@temp%
1166    \ifx\notation@option@temp\@empty\else%
1167      \IfSubStr\notation@option@temp={%
1168        \StrCut\notation@option@temp=\notation@key\notation@value%
1169        \trimstring\notation@key%
1170        \trimstring\notation@value%
1171        \IfStrEq\notation@key{prec}{%
1172          \edef\notation@curr@precs{\notation@value}%
1173        }{%
1174        \IfStrEq\notation@key{args}{%
1175          \edef\notation@curr@args{\notation@value}%
1176        }{%
1177        \IfStrEq\notation@key{lang}{%
1178          \edef\notation@curr@lang{\notation@value}%
1179        }{%
1180        \IfStrEq\notation@key{variant}{%
1181          \edef\notation@curr@variant{\notation@value}%
1182        }{%
1183        \IfStrEq\notation@key{arity}{%
1184          \edef\notation@curr@arity{\notation@value}%
1185        }{%
1186        }}}}}%
1187      }{%
1188        \edef\notation@curr@variant{\notation@option@temp}%
1189      }%
1190    \fi%
1191 }
1192
1193 % converts an integer to a string of 'i's, e.g. 3 => iii,
1194 % and stores the result in \notation@curr@args
1195 \def\notation@num@to@ia#1{%
1196    \IfInteger{#1}{
1197      \notation@num@to@ia@#1%
1198    }{%
```

```
1199       %
1200    }%
1201 }
1202 \def\notation@num@to@ia@#1{%
1203    \ifnum#1>0%
1204       \edef\notation@curr@args{\notation@curr@args i}%
1205       \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
1206    \fi%
1207 }
```

The following macros take care of precedences, parentheses/bracketing, associative (flexary) arguments etc. in presentation:

```
1208 \def\notation@assoc#1#2{% function, argv
1209    \let\@tmpop=\relax% do not print the function the first time round
1210    \@for\@I:=#2\do{\@tmpop% print the function
1211       % write the i-th argument with locally updated precedence
1212       \@I%
1213       \def\@tmpop{#1}%
1214    }%
1215 }%
1216
1217 \def\notation@lparen{(}
1218 \def\notation@rparen{)}
1219 \def\infprec{1000000}
1220 \def\neginfprec{-\infprec}
1221
1222 \newcount\notation@downprec
1223 \notation@downprec=\neginfprec
1224
1225 % patching displaymode
1226 \newif\if@displaymode\@displaymodefalse
1227 \expandafter\everydisplay\expandafter{\the\everydisplay\@displaymodetrue}
1228 \let\old@displaystyle\displaystyle
1229 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1230
1231 \def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1232    \def\notation@innertmp{#1}%
1233    \let\ex\expandafter%
1234    \if@displaymode%
1235       \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1236       \ex\notation@resetbrackets\ex\notation@innertmp%
1237       \ex\right\notation@rparen%
1238    \else%
1239       \ex\ex\ex\notation@lparen%
1240       \ex\notation@resetbrackets\ex\notation@innertmp%
1241       \notation@rparen%
1242    \fi%
1243 }
1244
1245 \def\withbrackets#1#2#3{%
```

```
1246    \edef\notation@lparen{#1}%
1247    \edef\notation@rparen{#2}%
1248    #3%
1249    \notation@resetbrackets%
1250 }
1251
1252 \def\notation@resetbrackets{%
1253    \def\notation@lparen{(}%
1254    \def\notation@rparen{)}%
1255 }
1256
1257 \def\notation@symprec#1#2{%
1258    \ifnum#1>\notation@downprec\relax%
1259       \notation@resetbrackets#2%
1260    \else%
1261       \ifnum\notation@downprec=\infprec\relax%
1262          \notation@resetbrackets#2%
1263       \else
1264          \if@inparray@
1265             \notation@resetbrackets#2
1266          \else\dobrackets{#2}\fi%
1267    \fi\fi%
1268 }
1269
1270 \newif\if@inparray@\@inparray@false
1271
1272 \def\notation@argprec#1#2{%
1273    \def\notation@innertmp{#2}
1274    \edef\notation@downprec@temp{\number#1}%
1275    \notation@downprec=\expandafter\notation@downprec@temp%
1276    \expandafter\relax\expandafter\notation@innertmp%
1277    \expandafter\notation@downprec\expandafter=\number\notation@downprec\relax%
1278 }
```

\@invoke@symbol   after \symdecl{foo}, \foo expands to \@invoke@symbol{<uri>}:

```
1279 \protected\def\@invoke@symbol#1{%
1280    \def\@invoke@symbol@first{#1}%
1281    \symbol@args%
1282 }
```

takes care of the optional notation-option-argument, and either invokes \@invoke@symbol@math for symbolic presentation or \@invoke@symbol@text for verbalization (TODO)

```
1283 \newcommand\symbol@args[1][]{%
1284    \notation@parse@params{#1}{}%
1285    \def\notation@temp@fragment{}%
1286    \ifx\notation@curr@arity\@empty\else%
1287       \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1288    \fi%
1289    \ifx\notation@curr@lang\@empty\else%
```

```
1290    \ifx\notation@temp@fragment\@empty%
1291       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1292    \else%
1293       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1294    \fi%
1295   \fi%
1296   \ifx\notation@curr@variant\@empty\else%
1297     \ifx\notation@temp@fragment\@empty%
1298       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1299     \else%
1300       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1301     \fi%
1302   \fi%
1303   %
1304   \ifmmode\def\invoke@symbol@next{\@invoke@symbol@math\@invoke@symbol@first\notation@temp@fragm
1305   \else\def\invoke@symbol@next{\@invoke@symbol@text\@invoke@symbol@first\notation@temp@fragment]
1306   \invoke@symbol@next%
1307 }
```

This finally gets called with both uri and notation-option, convenient for e.g.
a LaTeXML binding:

```
1308 \def\@invoke@symbol@math#1#2{%
1309   \csname #1\@Fragment#2\endcsname%
1310 }
```

TODO:

```
1311 \def\@invoke@symbol@text#1#2{%
1312     \csname #1\@Fragment verb\@Fragment#2\endcsname%
1313 }
```

TODO: To set notational options (globally or locally) generically:

```
1314 \def\setstexlang#1{%
1315   \def\stex@lang{#1}%
1316 }%
1317 \setstexlang{en}
1318 \def\setstexvariant#1#2{%
1319   % TODO
1320 }
1321 \def\setstexvariants#1{%
1322   \def\stex@variants{#1}%
1323 }
```

Test:
Module 3.30[FooBar]: \symdecl {barbar}
\notation [arity=0]{barbar}{\psi }
\notation [prec=50;\infprec]{barbar}[1]{\barbar [arity=0]\dobrackets {##1}}

\notation [arity=0,variant=cap]{barbar}{\Psi }
\notation [variant=cap]{barbar}[1]{\barbar [arity=0,variant=cap]\dobrackets {##1}}

$\barbar \{A\}$: $\psi(A)$
$\barbar [variant=cap]\{A\}$: $\Psi(A)$

\symdecl {plus}
\symdecl {times}
\symdecl {vara}
\symdecl {varb}
\symdecl {varc}
\symdecl {vard}
\symdecl {vare}
\notation {vara}{a}
\notation {varb}{b}
\notation {varc}{c}
\notation {vard}{d}
\notation {vare}{e}
\notation [prec=500;500,args=a]{plus}{\withbrackets \langle \rangle {##1}}{+}

\notation [prec=600;600,args=a]{times}{##1}{\cdot }

$\times {\frac \vara \varb ,\plus {\frac \vara {\frac \vara \varb },\times {\varc ,\plus {\vard ,\vare }}}}$:
$\frac{a}{b} \cdot (\frac{a}{\frac{a}{b}} + c \cdot (d+e))$

\[\times {\frac \vara \varb ,\plus {\frac \vara {\frac \vara \varb },\times {\varc ,\plus {\vard ,\vare }}}}\]:

$$\frac{a}{b} \cdot \left(\frac{a}{\frac{a}{b}} + c \cdot (d+e) \right)$$

\symdecl {foo!bar}
\foo !bar: foo bar
\symdecl [verbalization={finite group}]{finitegroup}
\verbalization [variant=oforder]{finitegroup}[1]{finite group of order ##1}
\finitegroup [oforder]{n}:   finite group of order n

3.6   Term References

\ifhref

```
1324 \newif\ifhref\hreffalse%
1325 \AtBeginDocument{%
1326   \@ifpackageloaded{hyperref}{%
1327     \hreftrue%
1328   }{%
1329     \hreffalse%
1330   }%
1331 }
```

`\termref@maketarget`   This macro creates a hypertarget `sref@`⟨*symbol URI*⟩`@target` and defines `\sref@`⟨*symbol URI*⟩`#1` to create a hyperlink to here on the text `#1`.

```
1332 \def\termref@maketarget#1#2{%
1333   % #1: symbol URI
1334   % #2: text
1335   \ifhref%
1336     \hypertarget{sref@#1@target}{#2}%
1337   \fi%
1338   \expandafter\edef\csname sref@#1\endcsname##1{%
1339     \ifhref\noexpand\hyperlink{sref@#1@target}{##1}\fi%
1340   }%
1341 }
```

`\@termref`

```
1342 \def\@termref#1#2{%
1343   % #1: symbol URI
1344   % #2: text
1345   \ifcvoid{#1}{%
1346     \StrCut{#1}\@QuestionMark\termref@mod\termref@name%
1347     \ifcsvoid{\termref@mod}{%
1348       \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
1349     }{%
1350       \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
1351         contains no symbol with name \termref@name.%
1352       }{}%
1353     }%
1354   }{%
1355     \ifcsvoid{sref@#1}{%
1356       % TODO: No reference point exists!
1357     }{%
1358       \csname sref@#1\endcsname{#2}%
1359     }%
1360   }%
1361 }
```

3.7 sref

We find out whether the `hyperref` package is loaded, since we may want to use it for cross-references, for which we set up some internal macros that gracefully degrade if `hyperref` is not loaded.

`\sref@*@ifh`

```
1362 \newif\ifhref\hreffalse%
1363 \AtBeginDocument{%
1364   \@ifpackageloaded{hyperref}{%
1365     \hreftrue%
1366   }{%
1367     \hreffalse%
1368   }%
```

```
1369 }%
1370 \newcommand\sref@href@ifh[2]{%
1371   \ifhref%
1372     \href{#1}{#2}%
1373   \else%
1374     #2%
1375   \fi%
1376 }%
1377 \newcommand\sref@hlink@ifh[2]{%
1378   \ifhref%
1379     \hyperlink{#1}{#2}%
1380   \else%
1381     #2%
1382   \fi%
1383 }%
1384 \newcommand\sref@target@ifh[2]{%
1385   \ifhref%
1386     \hypertarget{#1}{#2}%
1387   \else%
1388     #2%
1389   \fi%
1390 }%
```

Then we provide some macros for SₜₑX-specific crossreferencing

\sref@target   The next macro uses this and makes an target from the current `sref@id` declared
by a `id` key.

```
1391 \def\sref@target{%
1392   \ifx\sref@id\@empty%
1393     \relax%
1394   \else%
1395     \edef\@target{sref@\ifcsundef{sref@part}{}{\sref@part @}\sref@id @target}%
1396     \sref@target@ifh\@target{}%
1397   \fi%
1398 }%
```

\srefaddidkey   \srefaddidkey[⟨keyval⟩]{⟨group⟩} extends the metadata keys of the group
⟨group⟩ with an `id` key. In the optional key/value pairs in ⟨keyval⟩ the
`prefix` key can be used to specify a prefix. Note that the `id` key defined by
\srefaddidkey[⟨keyval⟩]{⟨group⟩} not only defines \sref@id, which is used for
referencing by the `sref` package, but also \⟨group⟩@id, which is used for showing
metadata via the `showmeta` option of the `metakeys` package.

```
1399 \addmetakey{srefaddidkey}{prefix}
1400 \newcommand\srefaddidkey[2][]{%
1401   \metasetkeys{srefaddidkey}{#1}%
1402   \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
1403   \metakeys@ext@clear@keys{#2}{id}{}%
1404   \metakeys@ext@showkeys{#2}{id}%
1405   \define@key{#2}{id}{%
```

```
1406      \edef\sref@id{\srefaddidkey@prefix ##1}%
1407      %\expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
1408      \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1409    }%
1410 }%
```

\@sref@def   This macro stores the value of its last argument in a custom macro for reference.

```
1411 \newcommand\@sref@def[3]{\csgdef{sref@#1@#2}{#3}}
```

The next step is to set up a file to which the references are written, this is normally the .aux file, but if the extref option is set, we have to use an .ref file.

```
1412 \ifextrefs%
1413    \newwrite\refs@file%
1414 \else%
1415    \def\refs@file{\@auxout}%
1416 \fi%
```

\sref@def   This macro writes an \@sref@def command to the current aux file and also executes it.

```
1417 \newcommand\sref@def[3]{%
1418    \protected@write\refs@file{}{\string\@sref@def{#1}{#2}{#3}}%
1419 }%
```

\sref@label   The \sref@label macro writes a label definition to the auxfile.

```
1420 \newcommand\sref@label[2]{%
1421    \sref@def{\ifcsundef{sref@part}{}{\sref@part @}#2}{page}{\thepage}%
1422    \sref@def{\ifcsundef{sref@part}{}{\sref@part @}#2}{label}{#1}%
1423 }%
```

\sreflabel   The \sreflabel macro is a semantic version of \label, it combines the categorization given in the first argument with LaTeX's \@currentlabel.

```
1424 \newcommand\sreflabel[2]{\sref@label{#1 \@currentlabel}{#2}}
```

\sref@label@id   The \sref@label@id writes a label definition for the current \sref@id if it is defined.

```
1425 \def\sref@id{} % make sure that defined
1426 \newcommand\sref@label@id[1]{%
1427    \ifx\sref@id\@empty%
1428      \relax%
1429    \else%
1430      \sref@label{#1}{\sref@id}%
1431    \fi%
1432 }%
```

\sref@label@id@arg   The \sref@label@id@arg writes a label definition for the second argument if it is defined.

```
1433 \newcommand\sref@label@id@arg[2]{%
1434    \def\@@id{#2}
```

```
1435    \ifx\@@id\@empty%
1436      \relax%
1437    \else%
1438      \sref@label{#1}{\@@id}%
1439    \fi%
1440 }%
```

3.8 smultiling

modsig The modsig environment is just a layer over the module environment. We also redefine macros that may occur in module signatures so that they do not create markup. Finally, we set the flag \mod@⟨mod⟩@multiling to true.

```
1441 \newenvironment{modsig}[2][]{\def\@test{#1}%
1442 \ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1443 \expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
1444 \ignorespacesandpars}
1445 {\end{module}\ignorespacesandparsafterend}
```

3.9 smglom

\gimport Just a shortcut, we have a starred and unstarred version, the first one is conservative. For example, if we execute:

```
\gimport[smglom/numberfields]{naturalnumbers}
```

First we are redirected to \@gimport@nostar, we store the smglom/numberfields⟨*the repo's path*⟩ in \@test, then store \mh@currentrepos⟨*current directory*⟩ in \mh@repos. If no repo's path is offered, that means the module to import is under the same directory, so we let mhrepos=\mh@repos and pass bunch of parameters to \importmhmodule, which is defined in module.sty. If there's a repo's path, then we let mhrepos=⟨*the repo's path*⟩. Finally we use \mhcurrentrepos(defined in module.sty) to change the \mh@currentrepos.

```
1446 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1447 \newrobustcmd\@gimport@star[2][]{\def\@test{#1}%
1448 \edef\mh@@repos{\mh@currentrepos}%
1449 \ifx\@test\@empty%
1450 \importmhmodule[conservative,mhrepos=\mh@@repos,path=#2]{#2}%
1451 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1452 \setcurrentreposinfo{\mh@@repos}%
1453 \ignorespacesandpars\parsemodule@maybesetcodes}
1454 \newrobustcmd\@gimport@nostar[2][]{\def\@test{#1}%
1455 \edef\mh@@repos{\mh@currentrepos}%
1456 \ifx\@test\@empty%
1457 \importmhmodule[mhrepos=\mh@@repos,path=#2]{#2}%
1458 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1459 \setcurrentreposinfo{\mh@@repos}%
1460 \ignorespacesandpars\parsemodule@maybesetcodes}
```

3.10 mathhub

\libinput the \libinput macro inputs from the `lib` directory of the MathHub repository
and then the `meta-inf/lib` repository of the group, if they exist. Since in prac-
tice nested libinputs may occur, we make sure that we stash the old values of
\mh@inffile and \mh@libfile and restore them at the end.

```
1461 \def\modules@@first#1/#2;{#1}
1462 \newcommand\libinput[1]{%
1463 \ifcsvoid{mh@currentrepos}{%
1464    \PackageError{mathhub}{current MathHub repository not found}{}}%
1465    {}
1466 \edef\@mh@group{\expandafter\modules@@first\mh@currentrepos;}
1467 \let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
1468 \def\mh@inffile{\MathHub{\@mh@group/meta-inf/lib/#1}}
1469 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1470 \IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
1471 \IfFileExists\mh@inffile{}{\IfFileExists\mh@libfile{}{%
1472    {\PackageError{mathhub}
1473      {Library file missing; cannot input #1.tex\MessageBreak%
1474      Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
1475      do not exist}%
1476    {Check whether the file name is correct}}}}
1477 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1478 \let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}
```

3.11 omdoc/omgroup

```
1479 \newcount\section@level
1480
1481 \section@level=2
1482 \ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{}
1483 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{}
1484 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{}
1485 \ifdefstring{\omdoc@sty@topsect}{chapter}{\section@level=1}{}
```

\omgroup@nonum convenience macro: \omgroup@nonum{⟨level⟩}{⟨title⟩} makes an unnumbered sec-
tioning with title ⟨title⟩ at level ⟨level⟩.

```
1486 \newcommand\omgroup@nonum[2]{%
1487 \ifx\hyper@anchor\@undefined\else\phantomsection\fi%
1488 \addcontentsline{toc}{#1}{#2}\@nameuse{#1}*{#2}}
```

\omgroup@num convenience macro: \omgroup@nonum{⟨level⟩}{⟨title⟩} makes numbered sectioning
with title ⟨title⟩ at level ⟨level⟩. We have to check the `short` key was given in the
`omgroup` environment and – if it is use it. But how to do that depends on whether
the `rdfmeta` package has been loaded. In the end we call \sref@label@id to
enable crossreferencing.

```
1489 \newcommand\omgroup@num[2]{%
1490 \edef\@@ID{\sref@id}
1491 \ifx\omgroup@short\@empty% no short title
```

```
1492 \@nameuse{#1}{#2}%
1493 \else% we have a short title
1494 \@ifundefined{rdfmeta@sectioning}%
1495   {\@nameuse{#1}[\omgroup@short]{#2}}%
1496   {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
1497 \fi%
1498 \sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@@ID}
```

omgroup

```
1499 \def\@true{true}
1500 \def\@false{false}
1501 \srefaddidkey{omgroup}
1502 \addmetakey{omgroup}{date}
1503 \addmetakey{omgroup}{creators}
1504 \addmetakey{omgroup}{contributors}
1505 \addmetakey{omgroup}{srccite}
1506 \addmetakey{omgroup}{type}
1507 \addmetakey*{omgroup}{short}
1508 \addmetakey*{omgroup}{display}
1509 \addmetakey[false]{omgroup}{loadmodules}[true]
```

we define a switch for numbering lines and a hook for the beginning of groups:

\at@begin@omgroup   The \at@begin@omgroup macro allows customization. It is run at the beginning of the omgroup, i.e. after the section heading.

```
1510 \newif\if@mainmatter\@mainmattertrue
1511 \newcommand\at@begin@omgroup[3][]{}
```

Then we define a helper macro that takes care of the sectioning magic. It comes with its own key/value interface for customization.

```
1512 \addmetakey{omdoc@sect}{name}
1513 \addmetakey[false]{omdoc@sect}{clear}[true]
1514 \addmetakey{omdoc@sect}{ref}
1515 \addmetakey[false]{omdoc@sect}{num}[true]
1516 \newcommand\omdoc@sectioning[3][]{\metasetkeys{omdoc@sect}{#1}%
1517 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
1518 \if@mainmatter% numbering not overridden by frontmatter, etc.
1519 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
1520 \def\current@section@level{\omdoc@sect@name}%
1521 \else\omgroup@nonum{#2}{#3}%
1522 \fi}% if@mainmatter
```

and another one, if redefines the \addtocontentsline macro of LATEX to import the respective macros. It takes as an argument a list of module names.

```
1523 \newcommand\omgroup@redefine@addtocontents[1]{%
1524 %\edef\@@import{#1}%
1525 %\@for\@I:=\@@import\do{%
1526 %\edef\@path{\csname module@\@I  @path\endcsname}%
1527 %\@ifundefined{tf@toc}\relax%
1528 %    {\protected@write\tf@toc{}{\string\@requiremodules{\@path}}}}
1529 %\ifx\hyper@anchor\@undefined% hyperref.sty loaded?
```

```
1530 %\def\addcontentsline##1##2##3{%
1531 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{#1}{##3}}{\thepage}}}
1532 %\else% hyperref.sty not loaded
1533 %\def\addcontentsline##1##2##3{%
1534 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{#1}{##3}}{\thepage}{\@cu
1535 %\fi
1536 }% hypreref.sty loaded?
```

now the `omgroup` environment itself. This takes care of the table of contents
via the helper macro above and then selects the appropriate sectioning com-
mand from `article.cls`. It also registeres the current level of omgroups in the
`\omgroup@level` counter.

```
1537 \newcount\omgroup@level
1538 \newenvironment{omgroup}[2][]% keys, title
1539 {\metasetkeys{omgroup}{#1}\sref@target%
1540 \advance\omgroup@level by 1\relax%
```

If the `loadmodules` key is set on `\begin{omgroup}`, we redefine the `\addcontetsline`
macro that determines how the sectioning commands below construct the entries
for the table of contents.

```
1541 \ifx\omgroup@loadmodules\@true%
1542 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
1543 {\@ifundefined{module@\module@id @path}{\used@modules}\module@id}}\fi%
```

now we only need to construct the right sectioning depending on the value of
`\section@level`.

```
1544 \advance\section@level by 1\relax%
1545 \ifcase\section@level%
1546 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
1547 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
1548 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
1549 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
1550 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%
1551 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{#2}%
1552 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{#2}
1553 \fi% \ifcase
1554 \at@begin@omgroup[#1]\section@level{#2}}% for customization
1555 {\advance\section@level by -1\advance\omgroup@level by -1}
```

and finally, we localize the sections

```
1556 \newcommand\omdoc@part@kw{Part}
1557 \newcommand\omdoc@chapter@kw{Chapter}
1558 \newcommand\omdoc@section@kw{Section}
1559 \newcommand\omdoc@subsection@kw{Subsection}
1560 \newcommand\omdoc@subsubsection@kw{Subsubsection}
1561 \newcommand\omdoc@paragraph@kw{paragraph}
1562 \newcommand\omdoc@subparagraph@kw{subparagraph}
```

`\setSGvar`   set a global variable

```
1563 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}
```

\useSGvar   use a global variable

```
1564 \newrobustcmd\useSGvar[1]{%
1565   \@ifundefined{sTeX@Gvar@#1}
1566   {\PackageError{omdoc}
1567     {The sTeX Global variable #1 is undefined}
1568     {set it with \protect\setSGvar}}
1569 \@nameuse{sTeX@Gvar@#1}}
```

blindomgroup

```
1570 \newcommand\at@begin@blindomgroup[1]{}
1571 \newenvironment{blindomgroup}
1572 {\advance\section@level by 1\at@begin@blindomgroup\setion@level}
1573 {\advance\section@level by -1}
```

3.12   omtext

4   Mathematical Text

We define the actions that are undertaken, when the keys are encountered. The
first set just records metadata; this is very simple via the \addmetakey infrastruc-
ture [Koh20]. Note that we allow math in the title field, so we do not declare it
to be Semiverbatim (indeed not at all, which allows it by default).

```
1574 \srefaddidkey{omtext}
1575 \addmetakey[]{omtext}{functions}
1576 \addmetakey*{omtext}{display}
1577 \addmetakey{omtext}{for}
1578 \addmetakey{omtext}{from}
1579 \addmetakey{omtext}{type}
1580 \addmetakey*{omtext}{title}
1581 \addmetakey*{omtext}{start}
1582 \addmetakey{omtext}{theory}
1583 \addmetakey{omtext}{continues}
1584 \addmetakey{omtext}{verbalizes}
1585 \addmetakey{omtext}{subject}
```

\st@flow   We define this macro, so that we can test whether the display key has the value
flow

```
1586 \def\st@flow{flow}
```

We define a switch that allows us to see whether we are inside an omtext
environment or a statement. It will be used to give better error messages for
inline statements.

```
1587 \newif\if@in@omtext\@in@omtextfalse
```

omtext   The omtext environment can have a title, which is used in a similar way. We
redefine the \lec macro so the trailing \par does not get into the way.

```
1588 \def\omtext@pre@skip{\smallskip}
```

47

```
1589 \def\omtext@post@skip{}
1590 \newenvironment{omtext}[1][]{\@in@omtexttrue%
1591   \bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
1592   \def\lec##1{\@lec{##1}}%
1593   \omtext@pre@skip\par\noindent%
1594   \ifx\omtext@title\@empty%
1595     \ifx\omtext@start\@empty\else%
1596       \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
1597     \fi% end omtext@start empty
1598   \else\stDMemph{\omtext@title}:\enspace%
1599     \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
1600   \fi% end omtext@title empty
1601   \ignorespacesandpars}
1602 {\egroup\omtext@post@skip\@in@omtextfalse\ignorespacesandpars}
```

5 Phrase-level Markup

\phrase   For the moment, we do disregard the most of the keys

```
1603 \srefaddidkey{phrase}
1604 \addmetakey{phrase}{style}
1605 \addmetakey{phrase}{class}
1606 \addmetakey{phrase}{index}
1607 \addmetakey{phrase}{verbalizes}
1608 \addmetakey{phrase}{type}
1609 \addmetakey{phrase}{only}
1610 \newcommand\phrase[2][]{\metasetkeys{phrase}{#1}%
1611 \ifx\prhase@only\@empty\only<\phrase@only>{#2}\else #2\fi}
```

\coref*

```
1612 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}
1613 \newcommand\corefs[2]{#1\textsubscript{#2}}
1614 \newcommand\coreft[2]{#1\textsuperscript{#2}}
```

\n*lex

```
1615 \newcommand\nlex[1]{\green{\sl{#1}}}
1616 \newcommand\nlcex[1]{*\green{\sl{#1}}}
```

sinlinequote

```
1617 \def\@sinlinequote#1{``{\sl{#1}}''}
1618 \def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
1619 \newcommand\sinlinequote[2][]
1620 {\def\@opt{#1}\ifx\@opt\@empty\@sinlinequote{#2}\else\@@sinlinequote\@opt{#2}\fi}
```

6 Declarations (under development)

The declaration macros are still under development (i.e. the macros) are still
under development and may change at any time. Currently they are completely

empty.

```
1621 \newcommand\vdec[2][]{#2}
1622 \newcommand\vrest[2][]{#2}
1623 \newcommand\vcond[2][]{#2}
```

\strucdec [1]

```
1624 \newcommand\strucdec[2][]{#2}
```

\impdec [2]

```
1625 \newcommand\impdec[2][]{#2}
```

7   Block-Level Markup

sblockquote

```
1626 \def\begin@sblockquote{\begin{quote}\sl}
1627 \def\end@sblockquote{\end{quote}}
1628 \def\begin@@sblockquote#1{\begin@sblockquote}
1629 \def\end@@sblockquote#1{\def\@@lec##1{\textrm{##1}}\@lec{#1}\end@sblockquote}
1630 \newenvironment{sblockquote}[1][]
1631   {\def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
1632   {\ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}
```

sboxquote

```
1633 \newenvironment{sboxquote}[1][]
1634 {\def\@@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
1635 {\@lec{\textrm\@@src}\end{mdframed}}
```

The line end comment macro makes sure that it will not be forced on the next line unless necessary.

\lec   The actual appearance of the line end comment is determined by the \@@lec macro, which can be customized in the document class. The basic one here is provided so that it is not missing.

```
1636 \providecommand{\@@lec}[1]{(#1)}
1637 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@@lec{#1}}
1638 \def\lec#1{\@lec{#1}\par}
```

8   Index Markup

\omdoc@index*   These are the main internal indexing commands – dividing them into four macros is awful, but I did not get list processing running. It makes sure that the modules necessary for interpreting the math in the index entries are loaded. If the loadmodules key is given, we import the module we are in otherwise all the currently imported modules. We do not have to require the module files, since the

---

[1]EdNote: document above
[2]EdNote: document above

index is a the end of the document. If the `at` key is given, then we use that for sorting in the index.

```
1639 \addmetakey{omdoc@index}{at}
1640 \addmetakey[false]{omdoc@index}{loadmodules}[true]
1641 \newcommand\omdoc@indexi[2][]{\ifindex%
1642 \metasetkeys{omdoc@index}{#1}%
1643 \@bsphack\begingroup\@sanitize%
1644 \protected@write\@indexfile{}{\string\indexentry%
1645 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1646 \ifx\omdoc@index@loadmodules\@true%
1647 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
1648 \else #2\fi% loadmodules
1649 }{\thepage}}%
1650 \endgroup\@esphack\fi}%ifindex
1651 \newcommand\omdoc@indexii[3][]{\ifindex%
1652 \metasetkeys{omdoc@index}{#1}%
1653 \@bsphack\begingroup\@sanitize%
1654 \protected@write\@indexfile{}{\string\indexentry%
1655 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1656 \ifx\omdoc@index@loadmodules\@true%
1657 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1658 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
1659 \else #2!#3\fi% loadmodules
1660 }{\thepage}}%
1661 \endgroup\@esphack\fi}%ifindex
1662 \newcommand\omdoc@indexiii[4][]{\ifindex%
1663 \metasetkeys{omdoc@index}{#1}%
1664 \@bsphack\begingroup\@sanitize%
1665 \protected@write\@indexfile{}{\string\indexentry%
1666 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1667 \ifx\omdoc@index@loadmodules\@true%
1668 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1669 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1670 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1671 \else #2!#3!#4\fi% loadmodules
1672 }{\thepage}}%
1673 \endgroup\@esphack\fi}%ifindex
1674 \newcommand\omdoc@indexiv[5][]{\ifindex%
1675 \metasetkeys{omdoc@index}{#1}%
1676 \@bsphack\begingroup\@sanitize%
1677 \protected@write\@indexfile{}{\string\indexentry%
1678 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1679 \ifx\omdoc@index@loadmodules\@true%
1680 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1681 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1682 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1683 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
1684 \else #2!#3!#4!#5\fi% loadmodules
1685 }{\thepage}}%
```

```
1686 \endgroup\@esphack\fi}%ifindex
```

Now, we make two interface macros that make use of this:

`\*indi*`

```
1687 \newcommand\aindi[3][]{{#2}\omdoc@indexi[#1]{#3}}
1688 \newcommand\indi[2][]{{#2}\omdoc@indexi[#1]{#2}}
1689 \newcommand\indis[2][]{{#2}\omdoc@indexi[#1]{#2s}}
1690 \newcommand\Indi[2][]{{\captitalize{#2}}\omdoc@indexi[#1]{#2}}
1691 \newcommand\Indis[2][]{{\capitalize{#2}}\omdoc@indexi[#1]{#2s}}
1692
1693 \newcommand\@indii[3][]{\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
1694 \newcommand\aindii[4][]{#2\@indii[#1]{#3}{#4}}
1695 \newcommand\indii[3][]{{#2 #3}\@indii[#1]{#2}{#3}}
1696 \newcommand\indiis[3][]{{#2 #3s}\@indii[#1]{#2}{#3}}
1697 \newcommand\Indii[3][]{{\captitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1698 \newcommand\Indiis[3][]{{\capitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1699
1700 \newcommand\@indiii[4][]{\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexii[#1]{#3}{#2 (#4)}}
1701 \newcommand\aindiii[5][]{{#2}\@indiii[#1]{#3}{#4}{#5}}
1702 \newcommand\indiii[4][]{{#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
1703 \newcommand\indiiis[4][]{{#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}
1704 \newcommand\Indiii[4][]{\captitalize{#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
1705 \newcommand\Indiiis[4][]{\capitalize{#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}
1706
1707 \newcommand\@indiv[5][]{\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
1708 \newcommand\aindiv[6][]{#2\@indiv[#1]{#3}{#4}{#5}{#6}}
1709 \newcommand\indiv[5][]{{#2 #3 #4 #5}\@indiv[#1]{#2}{#3}{#4}{#5}}
1710 \newcommand\indivs[5][]{{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
1711 \newcommand\Indiv[5][]{\capitalize{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
1712 \newcommand\Indivs[5][]{\capitalize{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
```

9 Miscellaneous

Some shortcuts that use math symbols but are not mathematical at all; in particular, they should not be translated by LaTeXML.

```
1713 \newcommand\hateq{\ensuremath{\widehat=}\xspace}
1714 \newcommand\hatequiv{\ensuremath{\widehat\equiv}\xspace}
1715 \@ifundefined{ergo}%
1716 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1717 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1718 \newcommand{\reflect@squig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%
1719 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
1720 \newcommand\notergo{\ensuremath{\not\leadsto}}
1721 \newcommand\notogre{\ensuremath{\not\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
```

10 Deprecated Functionality

In this section we centralize old interfaces that are only partially supported any more.

def

```
1722 \newcommand\indextoo[2][]{\indi[#1]{#2}%
1723 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}}
1724 \newcommand\indexalt[2][]{\aindi[#1]{#2}%
1725 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead}
1726 \newcommand\twintoo[3][]{\indii[#1]{#2}{#3}%
1727 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}
1728 \newcommand\twinalt[3][]{\aindii[#1]{#2}{#3}%
1729 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead}
1730 \newcommand\atwintoo[4][]{\indiii[#1]{#2}{#3}{#4}%
1731 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead
1732 \newcommand\atwinalt[4][]{\aindii[#1]{#2}{#3}{#4}%
1733 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instea
1734 ⟨/package⟩
```

\my*graphics

```
1735 \newcommand\mygraphics[2][]{\includegraphics[#1]{#2}%
1736   \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics
1737 \newcommand\mycgraphics[2][]{\begin{center}\mygraphics[#1]{#2}\end{center}%
1738   \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics
1739 \newcommand\mybgraphics[2][]{\fbox{\mygraphics[#1]{#2}}%
1740   \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics
1741 \newcommand\mycbgraphics[2][]{\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}%
1742   \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphi
```

11 Things to deprecate

Module options:

```
1743 \addmetakey*{module}{id} % TODO: deprecate properly
1744 \addmetakey*{module}{load}
1745 \addmetakey*{module}{path}
1746 \addmetakey*{module}{dir}
1747 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
1748 \addmetakey*{module}{noalign}[true]
1749
1750 \newif\if@insymdef@\@insymdef@false
```

symdef:keys The optional argument local specifies the scope of the function to be defined. If local is not present as an optional argument then \symdef assumes the scope of the function is global and it will include it in the pool of macros of the current module. Otherwise, if local is present then the function will be defined only locally and it will not be added to the current module (i.e. we cannot inherit a local function). Note, the optional key local does not need a value: we write

\symdef[local]{somefunction}[0]{some expansion}. The other keys are not used in the LATEX part.

```
1751 %\srefaddidkey{symdef}% what does this do?
1752 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
1753 \define@key{symdef}{noverb}[all]{}%
1754 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
1755 \define@key{symdef}{specializes}{}%
1756 \addmetakey*{symdef}{noalign}[true]
1757 \define@key{symdef}{primary}[true]{}%
1758 \define@key{symdef}{assocarg}{}%
1759 \define@key{symdef}{bvars}{}%
1760 \define@key{symdef}{bargs}{}%
1761 \addmetakey{symdef}{lang}%
1762 \addmetakey{symdef}{prec}%
1763 \addmetakey{symdef}{arity}%
1764 \addmetakey{symdef}{variant}%
1765 \addmetakey{symdef}{ns}%
1766 \addmetakey{symdef}{args}%
1767 \addmetakey{symdef}{name}%
1768 \addmetakey*{symdef}{title}%
1769 \addmetakey*{symdef}{description}%
1770 \addmetakey{symdef}{subject}%
1771 \addmetakey*{symdef}{display}%
1772 \addmetakey*{symdef}{gfc}%
```

EdN:3   3

\symdef   The the \symdef, and \@symdef macros just handle optional arguments.

```
1773 \def\symdef{\@ifnextchar[{\@symdef}{\@symdef[]}}%
1774 \def\@symdef[#1]#2{\@ifnextchar[{\@@symdef[#1]{#2}}{\@@symdef[#1]{#2}[0]}}%
```

\@@symdef   now comes the real meat: the \@@symdef macro does two things, it adds the macro definition to the macro definition pool of the current module and also provides it.

```
1775 \def\@@symdef[#1]#2[#3]{%
1776   \@insymdef@true%
1777   \metasetkeys{symdef}{#1}%
1778   \edef\symdef@tmp@optpars{\ifcsvoid{symdef@name}{[]}{[name=\symdef@name]}}%
1779   \expandafter\symdecl\symdef@tmp@optpars{#2}%
1780   \@insymdef@false%
1781   \notation[#1]{#2}[#3]%
1782 }% mod@show
1783 \def\symdef@type{Symbol}%
1784 \providecommand{\stDMemph}[1]{\textbf{#1}}
```

\symvariant   \symvariant{⟨sym⟩}[⟨args⟩]{⟨var⟩}{⟨cseq⟩} just extends the internal macro \modules@⟨sym⟩@pres@ defined by \symdef{⟨sym⟩}[⟨args⟩]{...} with a variant \modules@⟨sym⟩@pres@⟨var⟩ which expands to ⟨cseq⟩. Recall that this is called by the macro \⟨sym⟩[⟨var⟩] induced by the \symdef.

---
[3]EDNOTE: MK@MK: we need to document the binder keys above.

```
1785 \def\symvariant#1{%
1786   \@ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}%
1787   }%
1788 \def\@symvariant#1[#2]#3#4{%
1789   \notation[#3]{#1}[#2]{#4}%
1790 \ignorespacesandpars}%
```

\abbrdef   The \abbrdef macro is a variant of \symdef that does the same on the LATEX
           level.

```
1791 \let\abbrdef\symdef%
```

\@sym*   has a starred form for primary symbols. The key/value interface has no effect on
         the LATEX side. We read the to check whether only allowed ones are used.

```
1792 \newif\if@importing\@importingfalse
1793 \define@key{symi}{noverb}[all]{}%
1794 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
1795 \define@key{symi}{specializes}{}%
1796 \define@key{symi}{gfc}{}%
1797 \define@key{symi}{noalign}[true]{}%
1798 \newcommand\symi{\@ifstar\@symi@star\@symi}
1799 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
1800   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespacesa
1801 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
1802   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignor
1803 \newcommand\symii{\@ifstar\@symii@star\@symii}
1804 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
1805   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespa
1806 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
1807   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\i
1808 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
1809 \newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%
1810   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignores
1811 \newcommand\@symiii@star[4][]{\metasetkeys{symi}{#1}%
1812   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi
1813 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
1814 \newcommand\@symiv[5][]{\metasetkeys{symi}{#1}%
1815   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi\igno
1816 \newcommand\@symiv@star[5][]{\metasetkeys{symi}{#1}%
1817   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4-#5}
```

\importmhmodule   The \importmhmodule[⟨key=value list⟩]{module} saves the current value of
                  \mh@currentrepos in a local macro \mh@@repos, resets \mh@currentrepos to
                  the new value if one is given in the optional argument, and after importing resets
                  \mh@currentrepos to the old value in \mh@@repos. We do all the \ifx compar-
                  ison with an \expandafter, since the values may be passed on from other key
                  bindings. Parameters will be passed to \importmodule.

```
1818 %\srefaddidkey{importmhmodule}%
1819 \addmetakey{importmhmodule}{mhrepos}%
1820 \addmetakey{importmhmodule}{path}%
```

```
1821 \addmetakey{importmhmodule}{ext}% why does this exist?
1822 \addmetakey{importmhmodule}{dir}%
1823 \addmetakey[false]{importmhmodule}{conservative}[true]%
1824 \newcommand\importmhmodule[2][]{%
1825   \parsemodule@maybesetcodes
1826   \metasetkeys{importmhmodule}{#1}%
1827   \ifx\importmhmodule@dir\@empty%
1828     \edef\@path{\importmhmodule@path}%
1829   \else\edef\@path{\importmhmodule@dir/#2}\fi%
1830   \ifx\@path\@empty% if module name is not set
1831     \@importmodule[]{#2}{export}%
1832   \else%
1833     \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
1834     \ifx\importmhmodule@mhrepos\@empty% if in the same repos
1835       \relax% no need to change mh@currentrepos, i.e, current directory.
1836     \else%
1837       \setcurrentreposinfo\importmhmodule@mhrepos% change it.
1838       \addto@thismodulex{\noexpand\setcurrentreposinfo{\importmhmodule@mhrepos}}%
1839     \fi%
1840     \@importmodule[\MathHub{\mh@currentrepos/source/\@path}]{#2}{export}%
1841     \setcurrentreposinfo\mh@@repos% after importing, reset to old value
1842     \addto@thismodulex{\noexpand\setcurrentreposinfo{\mh@@repos}}%
1843   \fi%
1844   \ignorespacesandpars%
1845 }
```

\usemhmodule

```
1846 \addmetakey{importmhmodule}{load}
1847 \addmetakey{importmhmodule}{id}
1848 \addmetakey{importmhmodule}{dir}
1849 \addmetakey{importmhmodule}{mhrepos}
1850
1851 \addmetakey{importmodule}{load}
1852 \addmetakey{importmodule}{id}
1853
1854 \newcommand\usemhmodule[2][]{%
1855 \metasetkeys{importmhmodule}{#1}%
1856 \ifx\importmhmodule@dir\@empty%
1857 \edef\@path{\importmhmodule@path}%
1858 \else\edef\@path{\importmhmodule@dir/#2}\fi%
1859 \ifx\@path\@empty%
1860 \usemodule[id=\importmhmodule@id]{#2}%
1861 \else%
1862 \edef\mh@@repos{\mh@currentrepos}%
1863 \ifx\importmhmodule@mhrepos\@empty%
1864 \else\setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
1865 \usemodule{\@path\@QuestionMark#2}%
1866 %\usemodule[load=\MathHub{\mh@currentrepos/source/\@path},
1867 %                          id=\importmhmodule@id]{#2}%
1868 \setcurrentreposinfo\mh@@repos%
```

```
1869 \fi%
1870 \ignorespacesandpars}
```

\mhinputref

```
1871 \newcommand\mhinputref[2][]{%
1872   \edef\mhinputref@first{#1}%
1873   \ifx\mhinputref@first\@empty%
1874     \inputref{#2}%
1875   \else%
1876     \inputref[mhrepos=\mhinputref@first]{#2}%
1877   \fi%
1878 }
```