

stex-master.sty: \TeX 2.0*

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

December 1, 2020

Abstract

TODO

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

Contents

1	Introduction	3
2	User commands	3
3	Implementation	3
3.1	sTeX base	4
3.2	Paths and URIs	4
3.3	Modules	15
3.4	Inheritance	19
3.5	Symbols/Notations/Verbalizations	28
3.6	Term References	40
3.7	sref	42
3.8	smultiling	45
3.9	smglom	45
3.10	mathhub	46
3.11	omdoc/omgroup	46
3.12	omtext	49
4	Things to deprecate	54

1 Introduction

TODO

2 User commands

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

3 Implementation

```
1 \langle *package \rangle
2 \edef\old@newlinechar{\the\newlinechar}
3 \newlinechar=-1
4 % TODO
5 \newif\if@modules@html@\@modules@html@true
6 \DeclareOption{omdocmode}{\@modules@html@false}
7 % Modules:
8 \newif\ifmod@show\mod@showfalse
9 \DeclareOption{showmods}{\mod@showtrue}
10 % sref:
11 \newif\ifextrefs\extrefsfalse
12 \DeclareOption{extrefs}{\extrefstrue}
13 %
14 \ProcessOptions

15 \RequirePackage{standalone}
16 \RequirePackage{xspace}
17 \RequirePackage{metakeys}
```

3.1 sTeX base

The sTeX logo:

```

18 \protected\def\stex{%
19   \@ifundefined{texorpdfstring}%
20   {\let\texorpdfstring\@firstoftwo}%
21   }%
22   \texorpdfstring{\raisebox{-.5ex}{S\kern-.5ex\TeX}}{sTeX}\xspace%
23 }
24 \def\sTeX{\stex}

```

and a conditional for LaTeXML:

```

25 \newif\if@latexml\@latexmlfalse

```

3.2 Paths and URIs

```

26 \RequirePackage{xstring}
27 \RequirePackage{etoolbox}

```

`\defpath` `\defpath[optional argument]{macro name}{base path}` defines a new macro which can take another path to form 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/smgglom/sets}

```

will generate `/path/to/localmh/MathHub/source/smgglom/sets`.

```

28 \newrobustcmd\defpath[3][]{%
29   \expandafter\newcommand\csname #2\endcsname[1]{#3/#1}%
30 }%
31 \let\namespace\defpath

```

3.2.1 Path Canonicalization

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

```

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

```

```

44 \catcode'\pathsuris@oldcatcode@slash\relax%
45 \catcode'\pathsuris@oldcatcode@colon\relax%
46 \catcode'\pathsuris@oldcatcode@qm\relax%
47 }

```

We define some macros for later comparison.

```

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

```

\@cpath Canonicalizes (file) paths:

```

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

```

```

90
91 \def\@cpath@loop{%
92   \IfSubStr\pathsuris@cpath@temp\@Slash{%
93     \StrCut\pathsuris@cpath@temp\@Slash\pathsuris@cpath@temp@a\pathsuris@cpath@temp%
94     \ifx\pathsuris@cpath@temp@a\@ToTop%
95       \ifx\@CanPath\@empty%
96         \edef\@CanPath{\@ToTop}%
97       \else%
98         \edef\@CanPath{\@CanPath\@Slash\@ToTop}%
99       \fi%
100     \cpath@loop%
101   \else%
102     \ifx\pathsuris@cpath@temp@a\@Dot%
103       \cpath@loop%
104     \else%
105     \IfBeginWith\pathsuris@cpath@temp\@ToTop{%
106       \StrBehind{\pathsuris@cpath@temp}{\@ToTop}[\pathsuris@cpath@temp]%
107       \IfBeginWith\pathsuris@cpath@temp\@Slash{%
108         \edef\pathsuris@cpath@temp{\@CanPath\pathsuris@cpath@temp}%
109       }{%
110         \ifx\@CanPath\@empty\else%
111           \edef\pathsuris@cpath@temp{\@CanPath\@Slash\pathsuris@cpath@temp}
112         \fi%
113       }%
114       \def\@CanPath{}%
115       \cpath@loop%
116     }{%
117       \ifx\@CanPath\@empty%
118         \edef\@CanPath{\pathsuris@cpath@temp@a}%
119       \else%
120         \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp@a}%
121       \fi%
122       \cpath@loop
123     }%
124   \fi\fi%
125 }{
126   \ifx\@CanPath\@empty%
127     \edef\@CanPath{\pathsuris@cpath@temp}%
128   \else%
129     \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp}%
130   \fi%
131 }%
132 }

```

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.

```

133 \newcommand\cpath[1]{%
134   \cpath{#1}%
135   \@CanPath%
136 }
```

`\path@filename`

```

137 \def\path@filename#1#2{%
138   \edef\filename@oldpath{#1}%
139   \StrCount\filename@oldpath\@Slash[\filename@lastslash]%
140   \ifnum\filename@lastslash>0%
141     \StrBehind[\filename@lastslash]\filename@oldpath\@Slash[\filename@oldpath]%
142     \edef#2{\filename@oldpath}%
143   \else%
144     \edef#2{\filename@oldpath}%
145   \fi%
146 }
```

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:

```

147 \newif\if@iswindows@\@iswindows@false
148 \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:

```

149 \newif\if@windowstopath@inpath@
150 \def\windows@to@path#1{
```

```

151 \@windowstopath@inpath@false
152 \def\windows@temp{}
153 \edef\windows@path{#1}
154 \ifx\windows@path\@empty\else
155 \expandafter\windows@path@loop\windows@path\windows@path@end
156 \fi
157 \let#1\windows@temp
158 }
159 \def\windows@path@loop#1#2\windows@path@end{
160 \def\windows@temp@b{#2}
161 \ifx\windows@temp@b\@empty
162 \def\windows@continue{}
163 \else
164 \def\windows@continue{\windows@path@loop#2\windows@path@end}
165 \fi
166 \if@windowstopath@inpath@
167 \ifx#1\@BackSlash
168 \edef\windows@temp{\windows@temp\@Slash}
169 \else
170 \edef\windows@temp{\windows@temp#1}
171 \fi
172 \else
173 \ifx#1:
174 \edef\windows@temp{\@Slash\windows@temp}
175 \@windowstopath@inpath@true
176 \else
177 \edef\windows@temp{\windows@temp#1}
178 \fi
179 \fi
180 \windows@continue
181 }

```

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:

```

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

```



```

195     \def\windows@continue{}
196   \else
197     \def\windows@continue{\path@windows@loop#2\windows@path@end}
198   \fi
199   \if@windowstopath@inpath@
200     \ifx#1/
201       \edef\windows@temp{\windows@temp\@BackSlash}
202     \else
203       \edef\windows@temp{\windows@temp#1}
204     \fi
205   \else
206     \ifx#1/
207       \edef\windows@temp{\windows@temp:\@BackSlash}
208       \@windowstopath@inpath@true
209     \else
210       \edef\windows@temp{\windows@temp#1}
211     \fi
212   \fi
213   \windows@continue
214 }

```

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:

```

215 \def\trimstring#1{%
216   \edef\pathsuris@trim@temp{#1}%
217   \IfBeginWith\pathsuris@trim@temp\@Space{%
218     \StrGobbleLeft\pathsuris@trim@temp1[#1]%
219     \trimstring{#1}%
220   }{%
221     \IfEndWith\pathsuris@trim@temp\@Space{%
222       \StrGobbleRight\pathsuris@trim@temp1[#1]%
223       \trimstring{#1}%
224     }{%
225       \edef#1{\pathsuris@trim@temp}%
226     }%
227   }%
228 }

```

Test:

»bla blubb«

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

```

229 \def\kpsewhich#1#2{\begingroup%
230   \edef\kpsewhich@cmd{"|kpsewhich #2"}%
231   \everyeof{\noexpand}%

```

```

232 \catcode'\=12%
233 \edef#1{\@input\kpsewhich@cmd\@Space}%
234 \trimstring#1%
235 \if@iswindows@ \windows@to@path#1\fi%
236 \xdef#1{\expandafter\detokenize\expandafter{#1}}%
237 \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:

```

238 \edef\pwd@cmd{\if@iswindows@ -expand-var \percent CD\percent\else -var-value PWD\fi}
239 \kpsewhich\stex@maindir\pwd@cmd
240 \edef\stex@mainfile{\stex@maindir\@Slash\jobname}
241 \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:

```

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

```

```

271 \global\let\stex@currfilename\stex@currfilename%
272 \fi%
273 }

```

\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:

```

274 \def\stexinput#1{%
275 \stexiffileexists{#1}{%
276 \stex@currfile@push\stex@temp@path%
277 \input{\stex@currfile}%
278 \stex@currfile@pop%
279 }%
280 {%
281 \PackageError{stex}{File does not exist (#1): \stex@temp@path}{}%
282 }%
283 }
284 \def\stexiffileexists#1#2#3{%
285 \edef\stex@temp@path{#1}%
286 \if@iswindows@ \path@to@windows\stex@temp@path\fi%
287 \IfFileExists\stex@temp@path{#2}{#3}%
288 }
289 \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:

```

290 \kpsewhich\mathhub@path{--var-value MATHHUB}
291 \if@iswindows@\windows@to@path\mathhub@path\fi
292 \ifx\mathhub@path\@empty%
293 \PackageWarning{stex}{MATHHUB system variable not found or wrongly set}{%
294 \defpath{MathHub}{%
295 \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.

```

296 \def\findmanifest#1{
297 \cpath{#1}
298 \ifx\@CanPath\@Slash
299 \def\manifest@mf{}
300 \else\ifx\@CanPath\@empty
301 \def\manifest@mf{}
302 \else
303 \edef\@findmanifest@path{\@CanPath/MANIFEST.MF}

```

```

304 \if@iswindows@path@to@windows\@findmanifest@path\fi
305 \IfFileExists{\@findmanifest@path}{
306   \%message{MANIFEST.MF found at \@findmanifest@path}
307   \edef\manifest@mf{\@findmanifest@path}
308   \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
309 }{
310 \edef\@findmanifest@path{\@CanPath/META-INF/MANIFEST.MF}
311 \if@iswindows@path@to@windows\@findmanifest@path\fi
312 \IfFileExists{\@findmanifest@path}{
313   \%message{MANIFEST.MF found at \@findmanifest@path}
314   \edef\manifest@mf{\@findmanifest@path}
315   \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
316 }{
317 \edef\@findmanifest@path{\@CanPath/meta-inf/MANIFEST.MF}
318 \if@iswindows@path@to@windows\@findmanifest@path\fi
319 \IfFileExists{\@findmanifest@path}{
320   \%message{MANIFEST.MF found at \@findmanifest@path}
321   \edef\manifest@mf{\@findmanifest@path}
322   \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
323 }{
324   \findmanifest{\@CanPath/..}
325 }}}
326 \fi\fi
327 }

```

Test:

</home/jazzpirate/work/MathHub/smgglom/mv/META-INF/MANIFEST.MF>

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

```

328 \def\split@manifest@key{
329   \IfSubStr{\manifest@line}{\@Colon}{
330     \StrBefore{\manifest@line}{\@Colon}[\manifest@key]
331     \StrBehind{\manifest@line}{\@Colon}[\manifest@line]
332     \trimstring\manifest@line
333     \trimstring\manifest@key
334   }{
335     \def\manifest@key{}
336   }
337 }

```

the next helper function iterates over lines in MANIFEST.MF

```

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

```

```

347     }{
348     % narration-base
349     \IfStrEq\manifest@key{\detokenize{narration-base}}{
350         \xdef\manifest@mf@narr{\manifest@line}
351     }{
352     % namespace
353     \IfStrEq\manifest@key{\detokenize{source-base}}{
354         \xdef\manifest@mf@ns{\manifest@line}
355     }{
356     \IfStrEq\manifest@key{\detokenize{ns}}{
357         \xdef\manifest@mf@ns{\manifest@line}
358     }{
359     % dependencies
360     \IfStrEq\manifest@key{\detokenize{dependencies}}{
361         \xdef\manifest@mf@deps{\manifest@line}
362     }{
363     }}}}
364     \parse@manifest@loop
365 \fi
366 }

```

`\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.

```

367 \newread\@manifest
368 \def\parsemanifest#1#2{%
369     \gdef\temp@archive@dir{}%
370     \findmanifest{#2}%
371     \begingroup%
372     \gdef\manifest@mf@id{}%
373     \gdef\manifest@mf@narr{}%
374     \gdef\manifest@mf@ns{}%
375     \gdef\manifest@mf@deps{}%
376     \openin\@manifest\manifest@mf%
377     \parse@manifest@loop%
378     \closein\@manifest%
379     \endgroup%
380     \if@iswindows@ \windows@to@path\manifest@mf\fi%
381     \cslet{#1id}\manifest@mf@id%
382     \cslet{#1narr}\manifest@mf@narr%
383     \cslet{#1ns}\manifest@mf@ns%
384     \cslet{#1deps}\manifest@mf@deps%
385     \ifcvoid\manifest@mf@id{-}%
386     \cslet{#1dir}\temp@archive@dir%
387 }%
388 }

```

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.

```

389 \def\setcurrentreposinfo#1{%
390   \edef\mh@currentrepos{#1}%
391   \ifx\mh@currentrepos\@empty%
392     \edef\currentrepos@dir{\@Dot}%
393     \def\currentrepos@narr{}%
394     \def\currentrepos@ns{}%
395     \def\currentrepos@id{}%
396     \def\currentrepos@deps{}%
397   \else%
398     \ifcsdef{mathhub@dir@\mh@currentrepos}{%
399       \@inmhrepostrue
400       \edef\mh@currentrepos{#1}%
401       \expandafter\let\expandafter\currentrepos@dir\csname mathhub@dir@#1\endcsname%
402       \expandafter\let\expandafter\currentrepos@narr\csname mathhub@narr@#1\endcsname%
403       \expandafter\let\expandafter\currentrepos@ns\csname mathhub@ns@#1\endcsname%
404       \expandafter\let\expandafter\currentrepos@deps\csname mathhub@deps@#1\endcsname%
405     }{%
406       \parsemanifest{currentrepos@}{\MathHub{#1}}%
407       \@setcurrentreposinfo%
408       \ifcsvoid{currentrepos@dir}{\PackageError{stex}{No archive with %
409         name #1 found!}{make sure that #1 is directly in your MATHHUB folder %
410         and contains a MANIFEST.MF, either directly in #1 or in a meta-inf %
411         subfolder.}}{\@inmhrepostrue}%
412     }%
413   \fi%
414 }
415
416 \def\@setcurrentreposinfo{%
417   \edef\mh@currentrepos{\currentrepos@id}%
418   \ifcsvoid{currentrepos@dir}{%
419     \csxdef{mathhub@dir@\currentrepos@id}{\currentrepos@dir}%
420     \csxdef{mathhub@narr@\currentrepos@id}{\currentrepos@narr}%
421     \csxdef{mathhub@ns@\currentrepos@id}{\currentrepos@ns}%
422     \csxdef{mathhub@deps@\currentrepos@id}{\currentrepos@deps}%
423   }%
424 }

```

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

```

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

```

```

431 }
432 }

```

3.3 Modules

```

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

```

Aux:

```

434 \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>

```

435 \def\ignorespacesandparsafterend#1\ignorespaces\fi{#1\fi\ignorespacesandpars}

```

```

436 \def\ignorespacesandpars{\ifhmode\unskip\fi\@ifnextchar\par{\expandafter\ignorespacesandpars\@g

```

Options for the module-environment:

```

437 \addmetakey*{module}{title}
438 \addmetakey*{module}{name}
439 \addmetakey*{module}{creators}
440 \addmetakey*{module}{contributors}
441 \addmetakey*{module}{srccite}
442 \addmetakey*{module}{ns}
443 \addmetakey*{module}{narr}

```

module@heading We make a convenience macro for the module heading. This can be customized.

```

444 \ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%
445 \newrobustcmd\module@heading{%
446   \stepcounter{module}%
447   \ifmod@show%
448   \noindent{\textbf{Module} \thesection.\thetitle [\module@name}}%
449   \sref@label{id{Module \thesection.\thetitle [\module@name}}}%
450   \ifx\module@title\empty : \quad\else\quad(\module@title)\hfill\\%
451   \fi%
452 }%

```

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.

```

453 \newenvironment{module}[1][{}]{%
454   \begin{@module}[#1]%
455   \module@heading% make the headings
456   \ignorespacesandpars\parsemodule@maybesetcodes}{%
457   \end{@module}%
458   \ignorespacesafterend%
459 }%
460 \ifmod@show\surroundwithmdframed{module@om@common}\fi%

```

Some auxiliary methods:

```

461 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}
462 \def\addto@thismodule#1{%

```

```

463 \@ifundefined{this@module}{}{%
464   \expandafter\g@addto@macro@safe\this@module{#1}%
465 }%
466 }
467 \def\addto@thismodule#1{%
468 \ifundefined{this@module}{}{%
469   \edef\addto@thismodule@exp{#1}%
470   \expandafter\expandafter\expandafter\g@addto@macro@safe%
471   \expandafter\this@module\expandafter{\addto@thismodule@exp}%
472 }}

```

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

To compute the $\langle uri \rangle$ 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`.

```

473 \newif\ifarchive@ns@empty@\archive@ns@empty@false
474 \def\set@default@ns{%
475   \edef\@module@ns@temp{\stex@currpath}%
476   \if@iswindows@\windows@to@path\@module@ns@temp\fi%
477   \archive@ns@empty@false%
478   \ifcvoid{mh@currentrepos}{\archive@ns@empty@true}%
479   {\expandafter\ifx\cname mathhub@ns@\mh@currentrepos\endcsname\@empty\archive@ns@empty@true\fi%
480   }%
481   \ifarchive@ns@empty%
482     \edef\@module@ns@tempuri{file\@Colon\@Slash\@Slash\@module@ns@temp}%
483   \else%
484     \edef\@module@filepath@temppath{\@module@ns@temp}%
485     \edef\@module@ns@tempuri{\cname mathhub@ns@\mh@currentrepos\endcsname}%
486     \edef\@module@archivedirpath{\cname mathhub@dir@\mh@currentrepos\endcsname\@Slash source}%
487     \edef\@module@archivedirpath{\expandafter\detokenize\expandafter{\@module@archivedirpath}}%
488     \IfBeginWith\@module@filepath@temppath\@module@archivedirpath{%
489       \StrLen\@module@archivedirpath[\ns@temp@length]%
490       \StrGobbleLeft\@module@filepath@temppath\@module@filepath@temprest}%
491     \edef\@module@ns@tempuri{\@module@ns@tempuri\@module@filepath@temprest}%
492   }{}%
493 \fi%
494 \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]}%
495 \setkeys{module}{ns=\@module@ns@tempuri}%
496 }

```

Test:

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

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

```

497 \def\set@next@moduleid{%
498   \unless\ifcsname namespace@\module@ns @unnamedmodules\endcsname%
499     \csgdef{namespace@\module@ns @unnamedmodules}{0}%
500   \fi%
501   \edef\namespace@currnum{\csname namespace@\module@ns @unnamedmodules\endcsname}%
502   \edef\module@temp@setidname{\noexpand\setkeys{module}{name=module\namespace@currnum}}%
503   \module@temp@setidname%
504   \csxdef{namespace@\module@ns @unnamedmodules}{\the\numexpr\namespace@currnum+1}%
505 }

```

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 unexpanded 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 directly included in this module
- `\ $\langle uri \rangle$` that expands to `\invoke@module{ $\langle uri \rangle$ }` (see below).
- `\stex@module@ $\langle name \rangle$` that expands to `\ $\langle uri \rangle$` , if unambiguous, otherwise to `ambiguous`.

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.

```

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

```

```

514 \let\module@id\module@name% % TODO deprecate
515 \edef\module@uri{\module@ns\@QuestionMark\module@name}%
516 \csgdef\module@names@\module@uri}{}%
517 \csgdef\module@imports@\module@uri}{}%
518 \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
519 \ifcsvoid{stex@module@\module@name}{
520   \expandafter\global\expandafter\let\csname stex@module@\module@name\expandafter\endcsname\c
521 }{
522   \expandafter\edef\csname stex@module@\module@name\endcsname{\detokenize{ambiguous}}
523 }
524 \edef\this@module{%
525   \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
526 }%
527 \csdef{module@defs@\module@uri}{}%
528 \ifcsvoid{mh@currentrepos}{}{%
529   \@inmhrepostrue%
530   \addto@thismodule{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\en
531     {\noexpand\mh@currentrepos}}%
532   \addto@thismodule{\noexpand\setcurrentreposinfo{\mh@currentrepos}}%
533 }%
534 }{%
535   \if@inmhrepos%
536   \@inmhreposfalse%
537   \addto@thismodule{\noexpand\setcurrentreposinfo{\expandafter\noexpand\csname mh@old@repos@\m
538   \fi%
539 }%

```

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 $\langle uri \rangle$ and $\backslash\text{stex@module@}\langle id \rangle$, that ultimately expand to $\backslash\text{@invoke@module}\{\langle uri \rangle\}$. Currently, the only functionality is $\backslash\text{@invoke@module}\{\langle uri \rangle\}\backslash\text{@URI}$, which expands to the full uri of a module (i.e. via $\backslash\text{stex@module@}\langle id \rangle\backslash\text{@URI}$). In the future, this macro can be extended with additional functionality, e.g. accessing symbols in a macro for overloaded (macro-)names.

```
540 \def\@URI{uri}
541 \def\@invoke@module#1#2{%
542   \ifx\@URI#2%
543     #1%
544   \else%
545     % TODO something else
546     #2%
547   \fi%
548 }
```

3.4 Inheritance

3.4.1 Selective Inclusion

The next great goal is to establish the $\backslash\text{requiremodules}$ macro, which reads an \LaTeX 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 $\backslash\text{requiremodules}$.

$\backslash\text{parsemodule@allow*}$ The first step is setting up a functionality for registering \LaTeX macros and environments as part of a module signature.

```
549 \newif\if@smsmode\@smsmodefalse
550 \def\parsemodule@escapechar@allowed{true}
551 \def\parsemodule@allow#1{
552   \expandafter\let\csname parsemodule@allowedmacro@#1\endcsname\parsemodule@escapechar@allowed
553 }
554 \def\parsemodule@allowenv#1{
555   \expandafter\let\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed
556 }
557 \def\parsemodule@escapechar@beginstring{begin}
558 \def\parsemodule@escapechar@endstring{end}
```

and now we use that to actually register all the \LaTeX functionality as relevant for **sms** mode.

```
559 \parsemodule@allow{symdef}
560 \parsemodule@allow{abbrdef}
561 \parsemodule@allow{importmodule}
```

```

562 \parsemodule@allowenv{module}
563 \parsemodule@allow{importmhmodule}
564 \parsemodule@allow{gimport}
565 \parsemodule@allowenv{modsig}
566 \parsemodule@allowenv{mhmodsig}
567 \parsemodule@allowenv{mhmodnl}
568 \parsemodule@allowenv{modnl}
569 \parsemodule@allow{symvariant}
570 \parsemodule@allow{symi}
571 \parsemodule@allow{symii}
572 \parsemodule@allow{symiii}
573 \parsemodule@allow{symiv}
574 \parsemodule@allow{notation}
575 \parsemodule@allow{verbalization}
576 \parsemodule@allow{symdecl}
577
578 % to deprecate:
579
580 \parsemodule@allow{defi}
581 \parsemodule@allow{defii}
582 \parsemodule@allow{defiii}
583 \parsemodule@allow{defiv}
584 \parsemodule@allow{adefi}
585 \parsemodule@allow{adefii}
586 \parsemodule@allow{adefiii}
587 \parsemodule@allow{adefiv}
588 \parsemodule@allow{defis}
589 \parsemodule@allow{defiis}
590 \parsemodule@allow{defiiis}
591 \parsemodule@allow{defivs}
592 \parsemodule@allow{Defi}
593 \parsemodule@allow{Defii}
594 \parsemodule@allow{Defiii}
595 \parsemodule@allow{Defiv}
596 \parsemodule@allow{Defis}
597 \parsemodule@allow{Defiis}
598 \parsemodule@allow{Defiiis}
599 \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*).

```

600 \catcode'\.=0
601 .catcode'\=13

```

```

602 .def.@active@slash{\}
603 .catcode'.<=1
604 .catcode'.>=2
605 .catcode'\'{=12
606 .catcode'\'}=12
607 .def.@open@brace{<{>
608 .def.@close@brace{>}
609 .catcode'\'.=0
610 \catcode'\'.=12
611 \catcode'\'{=1
612 \catcode'\'}=2
613 \catcode'\'<=12
614 \catcode'\'>=12

```

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

`\set@parsemodule@catcodes`

```

615 \def\set@parsemodule@catcodes{%
616     \global\catcode'\=13%
617     \global\catcode'\#=12%
618     \global\catcode'\{=12%
619     \global\catcode'\}=12%
620     \global\catcode'\$=12%$
621     \global\catcode'\^=12%
622     \global\catcode'\_ =12%
623     \global\catcode'\&=12%
624     \expandafter\let\@active@slash\parsemodule@escapechar%
625 }

```

`\reset@parsemodule@catcodes`

```

626 \def\reset@parsemodule@catcodes{%
627     \global\catcode'\=0%
628     \global\catcode'\#=6%
629     \global\catcode'\{=1%
630     \global\catcode'\}=2%
631     \global\catcode'\$=3%$
632     \global\catcode'\^=7%
633     \global\catcode'\_ =8%
634     \global\catcode'\&=4%
635 }

```

`\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.

```

636 \def\parsemodule@maybesetcodes{%
637     \if@smsmode\set@parsemodule@catcodes\fi%
638 }

```

`\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@currcls` 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:

```
639
640 \def\parsemodule@escapechar{%
641     \def\parsemodule@escape@currcls{}}%
642     \parsemodule@escape@parse@nextchar{%
643 }%
```

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

```
644 \long\def\parsemodule@escape@parse@nextchar@#1{%
645     \ifcat a#1\relax%
646         \edef\parsemodule@escape@currcls{\parsemodule@escape@currcls#1}%
647         \let\parsemodule@do@next\parsemodule@escape@parse@nextchar@%
648     \else%
649         \def\parsemodule@last@char{#1}%
650         \def\parsemodule@do@next{\parsemodule@escapechar@checkcls}%
651     \fi%
652     \parsemodule@do@next%
653 }
```

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@checkendenv` respectively, to check whether the environment being opened/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.

```
654 \def\parsemodule@escapechar@checkcls{%
655     \ifx\parsemodule@escape@currcls\parsemodule@escapechar@beginstring%
656         \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@last@char}%
657     \else%
658         \ifx\parsemodule@escape@currcls\parsemodule@escapechar@endstring%
659             \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@last@char}%
660         \else%
661             \expandafter\ifx\csname parsemodule@allowedmacro@\parsemodule@escape@currcls\endcsname\relax
662                 \parsemodule@escapechar@allowed%
663             \else
664                 \parsemodule@do@next%
665             \fi
666         \fi
667     \fi
668 }
```

```

663         \ifx\parsemodule@last@char\@open@brace%
664             \expandafter\let\expandafter\parsemodule@do@next@ii\csname\parsemodule@escape@c
665             \edef\parsemodule@do@next{\noexpand\parsemodule@converttoproperbraces\@open@bra
666         \else%
667             \reset@parsemodule@catcodes%
668             \edef\parsemodule@do@next{\expandafter\noexpand\csname\parsemodule@escape@curr
669         \fi%
670     \else\def\parsemodule@do@next{\relax\parsemodule@last@char}\fi%
671 \fi%
672 \fi%
673 \parsemodule@do@next%
674 }

```

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.

```

675 \expandafter\expandafter\expandafter\def%
676 \expandafter\expandafter\expandafter\parsemodule@converttoproperbraces%
677 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
678     \reset@parsemodule@catcodes%
679     \parsemodule@do@next@ii{#1}%
680 }

```

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.

```

681 \expandafter\expandafter\expandafter\def%
682 \expandafter\expandafter\expandafter\parsemodule@escapechar@checkbeginenv%
683 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
684     \expandafter\ifx\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed%
685         \reset@parsemodule@catcodes%
686         \def\parsemodule@do@next{\begin{#1}}%
687     \else%
688         \def\parsemodule@do@next{#1}%
689     \fi%
690     \parsemodule@do@next%
691 }
692 \expandafter\expandafter\expandafter\def%
693 \expandafter\expandafter\expandafter\parsemodule@escapechar@checkendenv%
694 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
695     \expandafter\ifx\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed%
696         %\reset@parsemodule@catcodes%
697         \def\parsemodule@do@next{\end{#1}}%
698     \else%
699         \def\parsemodule@do@next{#1}%

```

```

700     \fi%
701     \parsemodule@do@next%
702 }

```

`\@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.

```

703 \newrobustcmd\@requiremodules[1]{%
704   \if@tempswa\requiremodules{#1}\fi%
705 }%

```

`\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.

```

706   \newrobustcmd\requiremodules[1]{%
707     \mod@showfalse%
708     \edef\mod@path{#1}%
709     \edef\mod@path{\expandafter\detokenize\expandafter{\mod@path}}%
710     \requiremodules@smsmode{#1}%
711   }%

```

`\requiremodules@smsmode` this reads `TeX` 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.

```

712   \newbox\modules@import@tempbox
713   \def\requiremodules@smsmode#1{%
714     \setbox\modules@import@tempbox\vbox{%
715       \@smsmodetrue%
716       \set@parsemodule@catcodes%
717       \hbadness=100000\relax%
718       \hfuzz=10000pt\relax%
719       \vbadness=100000\relax%
720       \vfuzz=10000pt\relax%
721       \stexinput{#1.tex}%
722       \reset@parsemodule@catcodes%
723     }%
724     \parsemodule@maybesetcodes%
725   }

```

Test:

parsing `F00/testmodule.tex`

macro:->`\@invoke@module {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master/F00?testmodule}`

3.4.2 importmodule

`\importmodule@bookkeeping`

```

726 \newif\if@importmodule@switchrepos\@importmodule@switchreposfalse

```



```

727 \def\importmodule@bookkeeping#1#2#3{%
728   \@importmodule@switchreposfalse%
729   \metasetkeys{importmodule}{#1}%
730   \ifcsvoid{importmodule@mhrepos}{%
731     \ifcsvoid{currentrepos@dir}{%
732       \let\importmodule@dir\stex@maindir%
733     }{%
734       \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
735     }%
736   }{%
737     \@importmodule@switchrepostrue%
738     \expandafter\let\csname importmodule@oldrepos@#2\endcsname\mh@currentrepos%
739     \setcurrentreposinfo\importmodule@mhrepos%
740     \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
741   }%
742   \StrCut{#2}\@QuestionMark\importmodule@subdir\importmodule@modulename%
743   \ifx\importmodule@modulename\@empty%
744     \let\importmodule@modulename\importmodule@subdir%
745     \let\importmodule@subdir\@empty%
746   \else%
747     \ifx\importmodule@subdir\@empty\else%
748       \edef\importmodule@dir{\importmodule@dir\@Slash\importmodule@subdir}%
749     \fi%
750   \fi%
751   #3%
752   \if@importmodule@switchrepos%
753     \expandafter\setcurrentreposinfo\csname importmodule@oldrepos@#2\endcsname%
754   \fi%
755   \ignorespacesandpars%
756 }

```

\importmodule

```

757 %\srefaddidkey{importmodule}
758 \addmetakey{importmodule}{mhrepos}
759 \newcommand\importmodule[2][\@importmodule[#1]{#2}{export}}{
760 \newcommand\@importmodule[3][\@importmodule@bookkeeping{#1}{#2}{%
761   \importmodule@dir\importmodule@modulename{#3}%
762 }%
763 }%
764 }

```

\@importmodule $\langle filepath \rangle$ [$\langle mod \rangle$] [$\langle export? \rangle$] loads $\langle filepath \rangle.tex$ and activates the module $\langle mod \rangle$. If $\langle export? \rangle$ is **export**, then it also re-exports the \symdefs from $\langle mod \rangle$.

First \@load will store the base file name with full path, then check if \module@ $\langle mod \rangle$ @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.

```

765 \newcommand\@importmodule[3][\{%
766 {%
767   \edef\@load{#1}%
768   \edef\@importmodule@name{#2}
769   \if@smsmode\else\ifcsvoid{stex@module@\@importmodule@name}{% TODO check this
770     \stexiffileexists\@load{\requiremodules\@load}{%
771       \requiremodules{\@load\@Slash\@importmodule@name}%
772     }%
773   }\fi%
774   \ifx\@load\@empty\else%
775     {% TODO
776     %     \edef\@path{csname module@#2@path\endcsname}%
777     %     \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do nothing
778     %     {\PackageError{stex}% else signal an error
779     %       {Module Name Clash\MessageBreak%
780     %         A module with name #2 was already loaded under the path "\@path"\MessageBreak%
781     %         The imported path "\@load" is probably a different module with the\MessageBreak%
782     %         same name; this is dangerous -- not importing}%
783     %     {Check whether the Module name is correct}%
784     %   }%
785   }%
786   \fi%
787   \global\let\@importmodule@load\@load%
788 }%
789 \edef\@export{#3}\def\@@export{export}%prepare comparison
790 %\ifx\@export\@@export\export@defs{#2}\fi% export the module
791 \ifx\@export\@@export\addto@thismodulex{%
792   \noexpand\@importmodule[\@importmodule@load]{#2}{noexport}%
793 }%
794 \if@smsmode\else
795 \ifcsvoid{this@module}{\{%
796   \ifcsvoid{module@imports@\module@uri}{
797     \csxdef{module@imports@\module@uri}{%
798       \csname stex@module@#2\endcsname\@URI% TODO check this
799     }%
800   }\{%
801     \csxdef{module@imports@\module@uri}{%
802       \csname stex@module@#2\endcsname\@URI,% TODO check this
803       \csname module@imports@\module@uri\endcsname%
804     }%
805   }%
806 }%
807 \fi\fi%
808 \if@smsmode\else\activate@defs{#2}\fi% activate the module
809 }%

```

Test:

`\importmodule {testmoduleimporta}:`

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

```
macro:->\@invoke@symbol {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-
master?testmoduleimporta?foo}
```

Test:

```
\importmodule {testmoduleimportb?importb}:
```

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

```
macro:->\@invoke@symbol {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-
master?importb?bar}
```

Test:

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

```
macro:->\@invoke@module {http://mathhub.info/smglob/algebra?idempotent}
```

```
macro:->\@invoke@symbol {http://mathhub.info/smglob/mv?equal?notequal}
```

```
macro:->\@ifstar \@gimport@star \@gimport@nostar
```

Default document module:

```
810 \AtBeginDocument{%
811   \set@default@ns%
812   \ifx\module@narr\@empty\setkeys{module}{narr=\module@ns}\fi%
813   \let\module@name\jobname%
814   \let\module@id\module@name % TODO deprecate
815   \edef\module@uri{\module@ns\@QuestionMark\module@name}%
816   \csgdef{module@names@\module@uri}{}%
817   \csgdef{module@imports@\module@uri}{}%
818   \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
819   \expandafter\global\expandafter\let\csname stex@module@\module@name\expandafter\endcsname\csname
820   \edef\this@module{%
821     \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
822   }%
823   \csdef{module@defs@\module@uri}{}%
824   \ifcvoid{mh@currentrepos}{}%
825     \@inmhreposttrue%
826     \addto@thismodule{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\endcsname
827       {\noexpand\mh@currentrepos}}%
828     \addto@thismodule{\noexpand\setcurrentreposinfo{\mh@currentrepos}}%
829   }%
830 }
```

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

```
831 \def\activate@defs#1{%
832   \ifcsundef{stex@module@#1}{ % TODO check this
833     \PackageError{stex}{No module with name #1 loaded}{Probably missing an
834       \detokenize{\importmodule} (or variant) somewhere?
835     }
836   }{%
837     \ifcsundef{module@\csname stex@module@#1\endcsname\@URI @activated}%

```

```

838      {\csname module@defs@\csname stex@module@#1\endcsname\@URI\endcsname}{}}%
839      \@namedef{module@\csname stex@module@#1\endcsname\@URI @activated}{true}%
840    }%
841  }%

\usemodule \usemodule acts like \importmodule, except that it does not re-export the se-
          mantic macros in the modules it loads.
842 \newcommand\usemodule[2] [] {\@importmodule[#1]{#2}{noexport}}

Test:
Module 3.26[Foo]:
Module 3.27[Bar]:    macro:->\@invoke@symbol {file:///home/jazzpirate/work/Software/ext/sTeX/sty/
master?Foo?foo}
Module 3.28[Baz]:    undefined
macro:->\@invoke@symbol {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-
master?Bar?bar}

\inputref@*skip hooks for spacing customization, they are empty by default.
843 \def\inputref@preskip{}
844 \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).
845 \newrobustcmd\inputref[2] [] {%
846   \importmodule@bookkeeping{#1}{#2}{%
847     %\inputreftrue
848     \inputref@preskip%
849     \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
850     \inputref@postskip%
851   }%
852 }%

```

3.5 Symbols/Notations/Verbalizations

```

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

\define@in@module calls \edef{#1}{#2} and adds the macro definition to \this@module
854 \def\define@in@module#1#2{
855   \expandafter\edef\csname #1\endcsname{#2}%
856   \edef\define@in@module@temp{%
857     \def\expandafter\noexpand\csname#1\endcsname%
858     {#2}%
859   }%
860   \if@symdeflocal\else%
861     \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
862     \expandafter\endcsname\expandafter{\define@in@module@temp}%
863   \fi%
864 }

```

```

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

865 \addmetakey{symdecl}{name}%
866 \addmetakey{symdecl}{verbalization}%
867
868 % constructs a symbol name and a verbalization by splitting at exclamation
869 % points - e.g. \symdecl{symmetric!group} leads to name=symmetric-group
870 % and verbalization "symmetric group".
871 \def\symdecl@constructname#1{%
872   \def\symdecl@name{}%
873   \def\symdecl@verbalization{}%
874   \edef\symdecl@tempname{#1}%
875   \symdecl@constructname@loop%
876 }
877
878 \def\symdecl@constructname@loop{%
879   \ifx\symdecl@tempname\@empty\else%
880     \StrCut\symdecl@tempname!\symdecl@tempfirst\symdecl@tempname%
881     \ifx\symdecl@name\@empty%
882       \let\symdecl@name\symdecl@tempfirst%
883       \let\symdecl@verbalization\symdecl@tempfirst%
884       \symdecl@constructname@loop%
885     \else%
886       \edef\symdecl@name{\symdecl@name-\symdecl@tempfirst}%
887       \edef\symdecl@verbalization{\symdecl@verbalization\@Space\symdecl@tempfirst}%
888       \symdecl@constructname@loop%
889     \fi%
890   \fi%
891 }
892
893 \newcommand\symdecl[2][]{%
894   \ifcsdef{this@module}{%
895     \metasetkeys{symdecl}{#1}%
896     \ifcsvoid{symdecl@name}{%
897       \ifcsvoid{symdecl@verbalization}{%
898         \symdecl@constructname{#2}%
899       }{%
900         \edef\symdecl@name{#2}%
901       }%
902     }{%
903       \ifcsvoid{symdecl@verbalization}{\edef\symdecl@verbalization{#2}}{%
904       }%
905       \edef\symdecl@uri{\module@uri\@QuestionMark\symdecl@name}%
906       \ifcsvoid{stex@symbol@\symdecl@name}{
907         \expandafter\edef\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}
908       }{
909         \expandafter\def\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}
910       }

```

```

911 \edef\symdecl@symbolmacro{
912   \noexpand\ifcsvoid{stex@symbol@\symdecl@name}{
913     \expandafter\edef\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\symd
914   }{
915     \expandafter\def\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\detok
916   }
917 }
918 \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
919 \expandafter\endcsname\expandafter{\symdecl@symbolmacro}%
920 \ifcsvoid{\symdecl@uri}{
921   \ifcsvoid{module@names@\module@uri}{%
922     \csxdef{module@names@\module@uri}{\symdecl@name}%
923   }{%
924     \csxdef{module@names@\module@uri}{\symdecl@name,%
925       \csname module@names@\module@uri\endcsname}%
926   }%
927 }{%
928 % not compatible with circular dependencies, e.g. test/omdoc/07-modules/smsstest.tex
929 \PackageWarning{stex}{symbol already defined: \symdecl@uri}{%
930   You need to pick a fresh name for your symbol%
931 }%
932 }%
933 \define@in@module\symdecl@uri{\noexpand\@invoke@symbol{\symdecl@uri}}%
934 \define@in@module{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
935 \global\expandafter\let\csname\symdecl@uri\@Fragment verb\@Fragment\endcsname\symdecl@verba
936 }{%
937 \PackageError{stex}{\detokenize{\symdecl} not in a module}{You need to be in a module%
938 in order to declare a new symbol}
939 }%
940 \if@insymdef@ \else \parsemodule@maybesetcodes \fi%
941 }

```

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:

```

942 \edef\stex@ambiguous{\detokenize{ambiguous}}
943 \edef\stex@macrostring{\detokenize{macro:->\@invoke@symbol}}
944 \def\modules@getURIfromName#1{%
945   \def\notation@uri{}%
946   \edef\modules@getURI@name{#1}%
947   \ifcsvoid{\modules@getURI@name}{
948     \edef\modules@temp@meaning{
949   }{

```

```

950 \edef\modules@temp@meaning{\expandafter\meaning\csname\modules@getURI@name\endcsname}
951 }
952 \IfBeginWith\modules@temp@meaning\stex@macrostring{
953 % is a \@invoke@symbol macro
954 \StrPosition\modules@temp@meaning\@close@brace[\stex@tempnum]
955 \StrMid\modules@temp@meaning{26}{\the\numexpr\stex@tempnum-1\@Space}{\notation@uri}
956 }{
957 % Check whether full URI or module?symbol or just name
958 \StrCount\modules@getURI@name\@QuestionMark[\isuri@number]
959 \ifnum\isuri@number=2
960 \edef\notation@uri{\modules@getURI@name}
961 \else
962 \ifnum\isuri@number=1
963 % module?name
964 \StrCut\modules@getURI@name\@QuestionMark\isuri@mod\isuri@name
965 \ifcsvoid{stex@module@\isuri@mod}{
966 \PackageError{stex}{No module with name \isuri@mod\@Space loaded}{
967 }{
968 \expandafter\ifx\csname stex@module@\isuri@mod\endcsname\stex@ambiguous
969 \PackageError{stex}{Module name \isuri@mod\@Space is ambiguous}{
970 \else
971 \edef\notation@uri{\csname stex@module@\isuri@mod\endcsname\@URI\@QuestionMark\isuri@mod}
972 \fi
973 }
974 \else
975 %name
976 \ifcsvoid{stex@symbol@\modules@getURI@name}{
977 \PackageError{stex}{No symbol with name \modules@getURI@name\@Space known}{
978 }{
979 \ifcsvoid{\module@uri\@QuestionMark\modules@getURI@name}{
980 \expandafter\ifx\csname stex@symbol@\modules@getURI@name\endcsname\stex@ambiguous
981 % Symbol name ambiguous and not in current module
982 \PackageError{stex}{Symbol name, URI or macroname \detokenize{#1} found!}{
983 \else
984 % Symbol not in current module, but unambiguous
985 \edef\notation@uri{\csname stex@symbol@\modules@getURI@name\endcsname}
986 \fi
987 }{ % Symbol in current module
988 \edef\notation@uri{\module@uri\@QuestionMark\modules@getURI@name}
989 }
990 }
991 \fi
992 \fi
993 }
994 }

```

`\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}{...}`

```

995 \newif\if@inverbalization\@inverbalizationfalse
996 % parses the first two arguments:
997 \providerobustcmd\notation[2][0]{%
998   \edef\notation@first{#1}%
999   \edef\notation@second{#2}%
1000   \notation@%
1001 }
1002
1003 \providerobustcmd\verbalization{%
1004   \@inverbalizationtrue%
1005   \notation%
1006 }
1007
1008 % parses the last two arguments
1009 \newcommand\notation@[2][0]{%
1010   \edef\notation@donext{\noexpand\notation@@[\notation@first]%
1011     {\notation@second}[#1]}%
1012   \notation@donext{#2}%
1013 }
1014
1015 % parses the notation arguments and wraps them in
1016 % \notation@assoc and \notation@argprec for flexary arguments and precedences
1017 \def\notation@@[#1]#2[#3]#4{%
1018   \modules@getURIfromName{#2}%
1019   \notation@parse@params{#1}{#3}
1020   \let\notation@curr@todo@args\notation@curr@args%
1021   \def\notation@temp@notation{%
1022     \StrLen\notation@curr@args[\notation@temp@arity]%
1023     \expandafter\renewcommand\expandafter\notation@temp@notation%
1024       \expandafter[\notation@temp@arity]{#4}%
1025   % precedence
1026   \IfSubStr\notation@curr@prec;{%
1027     \StrCut\notation@curr@prec;\notation@curr@prec\notation@curr@prec%
1028     \ifx\notation@curr@prec\empty\def\notation@curr@prec{0}\fi%
1029   }{%
1030     \ifx\notation@curr@prec\empty%
1031       \ifnum\notation@temp@arity=0\relax%
1032         \edef\notation@curr@prec{\infprec}%
1033       \else%
1034         \def\notation@curr@prec{0}%
1035       \fi%
1036     \else%
1037       \edef\notation@curr@prec{\notation@curr@prec}%
1038       \def\notation@curr@prec{}%
1039     \fi%
1040   }%
1041   % arguments
1042   \def\notation@curr@extargs{}
1043   \def\notation@nextarg@index{1}%
1044   \notation@do@args%

```



```

1045 }
1046
1047 % parses additional notation components for (associative) arguments
1048 \def\notation@do@args{%
1049   \def\notation@nextarg@temp{}%
1050   \ifx\notation@curr@todo@args\@empty%
1051     \notation@after%
1052   \else%
1053     % argument precedence
1054     \IfSubStr\notation@curr@prec{x}{%
1055       \StrCut\notation@curr@prec{x}\notation@curr@argprec\notation@curr@prec%
1056     }{%
1057       \edef\notation@curr@argprec{\notation@curr@prec}%
1058       \def\notation@curr@prec{}%
1059     }%
1060     \ifx\notation@curr@argprec\@empty%
1061       \let\notation@curr@argprec\notation@curr@prec%
1062     \fi%
1063     \StrChar\notation@curr@todo@args1[\notation@argchar]%
1064     \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1065     \expandafter\ifx\notation@argchar i%
1066       % normal argument
1067       \edef\notation@nextarg@temp{{\noexpand\notation@argprec{\notation@curr@argprec}}{#####\
1068       \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }
1069       \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
1070       \expandafter{\notation@nextarg@temp}%
1071       \expandafter\expandafter\expandafter\notation@do@args%
1072     \else%
1073       % associative argument
1074       \expandafter\expandafter\expandafter\notation@parse@assocarg%
1075     \fi%
1076   \fi%
1077 }
1078
1079 \def\notation@parse@assocarg#1{%
1080   \edef\notation@nextarg@temp{{\noexpand\notation@argprec{\notation@curr@argprec}}{\noexpand\not
1081   \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }%
1082   \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
1083   \expandafter{\notation@nextarg@temp}%
1084   \notation@do@args%
1085 }
1086
1087 \protected\def\safe@newcommand#1{%
1088   \ifdefined#1\expandafter\renewcommand\else\expandafter\newcommand\fi#1%
1089 }
1090
1091 % finally creates the actual macros
1092 \def\notation@after{
1093   \let\ex\expandafter%
1094   \ex\ex\ex\def\ex\ex\ex\notation@temp\notation\ex\ex\ex%

```

```

1095     {\ex\notation@temp@notation\notation@curr@extargs}%
1096 \edef\notation@temp@notation{\noexpand\notation@symprec{\notation@curr@prec}{\ex\unexpanded\ex}}
1097 \def\notation@temp@fragment{}%
1098 \ifx\notation@curr@arity\@empty\else%
1099     \edef\notation@temp@fragment{arity=\notation@curr@arity}
1100 \fi%
1101 \ifx\notation@curr@lang\@empty\else%
1102     \ifx\notation@temp@fragment\@empty%
1103         \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1104     \else%
1105         \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1106     \fi%
1107 \fi%
1108 \ifx\notation@curr@variant\@empty\else%
1109     \ifx\notation@temp@fragment\@empty%
1110         \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1111     \else%
1112         \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@variant}%
1113     \fi%
1114 \fi%
1115 \if@inverbalization\@inverbalizationfalse\verbalization@final%
1116 \else\notation@final\fi%
1117 \parsemodule@maybesetcodes%
1118 }
1119
1120 \def\notation@final{%
1121     \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1122     \ifcsvoid{\notation@csname}{%
1123         \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1124             \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@temp@arity\ex]%
1125         \ex{\notation@temp@notation}%
1126         \edef\symdecl@temps{%
1127             \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@temp@arity%
1128             }%
1129         \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1130         \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@notation}%
1131         }-%
1132         \PackageWarning{stex}{notation already defined: \notation@csname}{%
1133             Choose a different set of notation options (variant,lang,arity)%
1134         }%
1135     }%
1136 }
1137
1138 \def\verbalization@final{%
1139     \edef\notation@csname{\notation@uri\@Fragment verb\@Fragment\notation@temp@fragment}%
1140     \ifcsvoid{\notation@csname}{%
1141         \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1142             \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@temp@arity\ex]%
1143         \ex{\notation@temp@notation}%
1144         \edef\symdecl@temps{%

```

```

1145     \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@temp@ari
1146   }%
1147   \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1148   \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1149 }{%
1150   \PackageWarning{stex}{verbalization already defined: \notation@csname}{%
1151     Choose a different set of verbalization options (variant,lang,arity)%
1152   }%
1153 }%
1154 }
1155
1156 % parses optional parameters
1157 \def\notation@parse@params#1#2{%
1158   \def\notation@curr@prec{%
1159     \def\notation@curr@args{%
1160       \def\notation@curr@variant{%
1161         \def\notation@curr@arity{%
1162           \def\notation@curr@provided@arity{#2}
1163           \def\notation@curr@lang{%
1164             \def\notation@options@temp{#1}
1165             \notation@parse@params%
1166             \ifx\notation@curr@args\@empty%
1167               \ifx\notation@curr@provided@arity\@empty%
1168                 \notation@num@to@ia\notation@curr@arity%
1169               \else%
1170                 \notation@num@to@ia\notation@curr@provided@arity%
1171               \fi%
1172             \fi%
1173           }
1174           \def\notation@parse@params@{%
1175             \IfSubStr\notation@options@temp,{%
1176               \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
1177               \notation@parse@param%
1178               \notation@parse@params@%
1179             }{\ifx\notation@options@temp\@empty\else%
1180               \let\notation@option@temp\notation@options@temp%
1181               \notation@parse@param%
1182             \fi}%
1183           }
1184
1185 %parses an individual optional argument/key-value-pair
1186 \def\notation@parse@param{%
1187   \trimstring\notation@option@temp%
1188   \ifx\notation@option@temp\@empty\else%
1189     \IfSubStr\notation@option@temp={%
1190       \StrCut\notation@option@temp=\notation@key\notation@value%
1191       \trimstring\notation@key%
1192       \trimstring\notation@value%
1193       \IfStrEq\notation@key{prec}{%
1194         \edef\notation@curr@prec{\notation@value}%

```

```

1195     }{%
1196     \IfStrEq\notation@key{args}{%
1197         \edef\notation@curr@args{\notation@value}%
1198     }{%
1199     \IfStrEq\notation@key{lang}{%
1200         \edef\notation@curr@lang{\notation@value}%
1201     }{%
1202     \IfStrEq\notation@key{variant}{%
1203         \edef\notation@curr@variant{\notation@value}%
1204     }{%
1205     \IfStrEq\notation@key{arity}{%
1206         \edef\notation@curr@arity{\notation@value}%
1207     }{%
1208         }}}}%
1209     }{%
1210         \edef\notation@curr@variant{\notation@option@temp}%
1211     }%
1212 \fi%
1213 }
1214
1215 % converts an integer to a string of 'i's, e.g. 3 => iii,
1216 % and stores the result in \notation@curr@args
1217 \def\notation@num@to@ia#1{%
1218     \IfInteger{#1}{
1219         \notation@num@to@ia@#1%
1220     }{%
1221         %
1222     }%
1223 }
1224 \def\notation@num@to@ia@#1{%
1225     \ifnum#1>0%
1226         \edef\notation@curr@args{\notation@curr@args i}%
1227         \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
1228     \fi%
1229 }

```

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

```

1230 \def\notation@assoc#1#2{% function, argv
1231     \let\@tmpop=\relax% do not print the function the first time round
1232     \@for\@I:=#2\do{\@tmpop% print the function
1233         % write the i-th argument with locally updated precedence
1234         \@I%
1235     \def\@tmpop{#1}%
1236     }%
1237 }%
1238
1239 \def\notation@lparen{()
1240 \def\notation@rparen{)}
1241 \def\infprec{1000000}

```

```

1242 \def\neginfprec{-\infprec}
1243
1244 \newcount\notation@downprec
1245 \notation@downprec=\neginfprec
1246
1247 % patching displaymode
1248 \newif@if@displaymode\@displaymodefalse
1249 \expandafter\everydisplay\expandafter{\the\everydisplay\@displaymodetrue}
1250 \let\old@displaystyle\displaystyle
1251 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1252
1253 \def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1254   \def\notation@innertmp{#1}%
1255   \let\ex\expandafter%
1256   \if@displaymode%
1257     \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1258     \ex\notation@resetbrackets\ex\notation@innertmp%
1259     \ex\right\notation@rparen%
1260   \else%
1261     \ex\ex\ex\notation@lparen%
1262     \ex\notation@resetbrackets\ex\notation@innertmp%
1263     \notation@rparen%
1264   \fi%
1265 }
1266
1267 \def\withbrackets#1#2#3{%
1268   \edef\notation@lparen{#1}%
1269   \edef\notation@rparen{#2}%
1270   #3%
1271   \notation@resetbrackets%
1272 }
1273
1274 \def\notation@resetbrackets{%
1275   \def\notation@lparen{()%
1276   \def\notation@rparen{)%}
1277 }
1278
1279 \def\notation@symprec#1#2{%
1280   \ifnum#1>\notation@downprec\relax%
1281     \notation@resetbrackets#2%
1282   \else%
1283     \ifnum\notation@downprec=\infprec\relax%
1284       \notation@resetbrackets#2%
1285     \else
1286       \if@inarray@
1287         \notation@resetbrackets#2
1288       \else\dobrackets{#2}\fi%
1289     \fi\fi%
1290 }
1291

```

```

1292 \newif\if@inpparray@\@inpparray@false
1293
1294 \def\notation@argprec#1#2{%
1295   \def\notation@innertmp{#2}
1296   \edef\notation@downprec@temp{\number#1}%
1297   \notation@downprec=\expandafter\notation@downprec@temp%
1298   \expandafter\relax\expandafter\notation@innertmp%
1299   \expandafter\notation@downprec\expandafter=\number\notation@downprec\relax%
1300 }

```

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

```

1301 \protected\def\@invoke@symbol#1{%
1302   \def\@invoke@symbol@first{#1}%
1303   \symbol@args%
1304 }

```

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

```

1305 \newcommand\symbol@args[1][]{%
1306   \notation@parse@params{#1}{}%
1307   \def\notation@temp@fragment{}%
1308   \ifx\notation@curr@arity\@empty\else%
1309     \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1310   \fi%
1311   \ifx\notation@curr@lang\@empty\else%
1312     \ifx\notation@temp@fragment\@empty%
1313       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1314     \else%
1315       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1316     \fi%
1317   \fi%
1318   \ifx\notation@curr@variant\@empty\else%
1319     \ifx\notation@temp@fragment\@empty%
1320       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1321     \else%
1322       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1323     \fi%
1324   \fi%
1325   %
1326   \ifmmode\def\invoke@symbol@next{\@invoke@symbol@math\@invoke@symbol@first\notation@temp@fragme
1327   \else\def\invoke@symbol@next{\@invoke@symbol@text\@invoke@symbol@first\notation@temp@fragment
1328   \invoke@symbol@next%
1329 }

```

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

```

1330 \def\@invoke@symbol@math#1#2{%
1331   \csname #1\@Fragment#2\endcsname%
1332 }

```

TODO:

```
1333 \def\@invoke@symbol@text#1#2{%
1334   \@termref{#1}{\csname #1\@Fragment verb\@Fragment#2\endcsname}%
1335 }
```

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

```
1336 \def\setstexlang#1{%
1337   \def\stex@lang{#1}%
1338 }%
1339 \setstexlang{en}
1340 \def\setstexvariant#1#2{%
1341   % TODO
1342 }
1343 \def\setstexvariants#1{%
1344   \def\stex@variants{#1}%
1345 }
```

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 }
```

```
$\times {\frac {vara }{varb } , \plus {\frac {vara }{\frac {vara }{varb } } , \times {\frac {varc }{ , \plus {\frac {vard }{ , \frac {vare }{ }}} } } }$:
```

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

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

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

`\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`

```
1346 \newif\ifhref\hreffalse%
1347 \AtBeginDocument{%
1348   \ifpackageloaded{hyperref}{%
1349     \hreftrue%
1350   }{%
1351     \hreffalse%
1352   }%
1353 }
```

`\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.

```
1354 \def\termref@maketarget#1#2{%
1355   % #1: symbol URI
1356   % #2: text
1357   \ifhref%
1358     \hypertarget{sref@#1@target}{#2}%
1359   \fi%
1360   \expandafter\edef\csname sref@#1\endcsname##1{%
1361     \ifhref\noexpand\hyperlink{sref@#1@target}{##1}\fi%
1362   }%
1363 }
```

`\@termref`

```
1364 \def\@termref#1#2{%
1365   % #1: symbol URI
1366   % #2: text
1367   \ifcvoid{#1}{%
1368     \StrCut[2]{#1}\@QuestionMark\termref@mod\termref@name%
1369     \ifcvoid{\termref@mod}{%
1370       \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
1371     }{%
1372       \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
```



```

1373         contains no symbol with name \termref@name.%
1374     }{%
1375 }%
1376 }{%
1377     \ifcsvoid{sref@#1}{%
1378         #2% TODO: No reference point exists!
1379     }{%
1380         \csname sref@#1\endcsname{#2}%
1381     }%
1382 }%
1383 }

\tref
1384
1385 \def\@capitalize#1{\uppercase{#1}}%
1386 \newrobustcmd\capitalize[1]{\expandafter\@capitalize #1}%
1387
1388 \newcommand\tref[2][]{%
1389     \edef\tref@name{#1}%
1390     \ifx\tref@name\@empty
1391         \symdecl@constructname{#2}%
1392         \edef\tref@name{\symdecl@name}%
1393     \else%
1394         \edef\symdecl@verbalization{#2}%
1395     \fi%
1396     \expandafter\modules@getURIfromName\expandafter{\tref@name}%
1397     \expandafter\@termref\expandafter{\notation@uri}{\symdecl@verbalization}%
1398 }
1399 \def\trefs#1{%
1400     \modules@getURIfromName{#1}%
1401     \expandafter\@termref\expandafter{\notation@uri}{\csname\notation@uri\@Fragment verb\@Fragment
1402 }
1403 \def\Tref#1{%
1404     \modules@getURIfromName{#1}%
1405     \expandafter\@termref\expandafter{\notation@uri}{\expandafter\capitalize\csname\notation@uri\
1406 }
1407 \def\Trefs#1{%
1408     \modules@getURIfromName{#1}%
1409     \expandafter\@termref\expandafter{\notation@uri}{\expandafter\capitalize\csname\notation@uri\
1410 }

Test:
foo bar
foo-bar
finite group

\defi
1411 \addmetakey{defi}{name}
1412 \def\@definiendum#1#2{%
1413     \defemph{\termref@maketarget{#1}{#2}}%

```

```

1414 \parsemodule@maybesetcodes%
1415 }
1416
1417 \newcommand\defi[2] []{%
1418 \metasetkeys{defi}{#1}%
1419 \ifx\defi@name\@empty%
1420 \symdecl@constructname{#2}%
1421 \let\defi@name\symdecl@name%
1422 \let\defi@verbalization\symdecl@verbalization%
1423 \else%
1424 \edef\defi@verbalization{#2}%
1425 \fi%
1426 \ifcsvoid{\module@uri\@QuestionMark\defi@name}{%
1427 \symdecl\defi@name%
1428 }{\edef\symdecl@uri{\module@uri\@QuestionMark\defi@name}}%
1429 \@definiendum\symdecl@uri\defi@verbalization%
1430 }
1431 \def\Defi#1{%
1432 \symdecl{#1}%
1433 \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization}%
1434 }
1435 \def\defis#1{%
1436 \symdecl{#1}%
1437 \@definiendum\symdecl@uri{\symdecl@verbalization s}%
1438 }
1439 \def\Defis#1{%
1440 \symdecl{#1}%
1441 \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization s}%
1442 }

```

Test:

a simple group
simple group

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
1443 \newif\ifhhref\hreffalse%
1444 \AtBeginDocument{%
1445 \@ifpackageloaded{hyperref}{%
1446 \hreftrue%
1447 }{%
1448 \hreffalse%
1449 }%
1450 }%
1451 \newcommand\sref@href@ifh[2]{%

```

```

1452 \ifhref%
1453   \href{#1}{#2}%
1454 \else%
1455   #2%
1456 \fi%
1457 }%
1458 \newcommand\sref@hlink@ifh[2]{%
1459   \ifhref%
1460     \hyperlink{#1}{#2}%
1461   \else%
1462     #2%
1463   \fi%
1464 }%
1465 \newcommand\sref@target@ifh[2]{%
1466   \ifhref%
1467     \hypertarget{#1}{#2}%
1468   \else%
1469     #2%
1470   \fi%
1471 }%

```

Then we provide some macros for \LaTeX -specific crossreferencing

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

```

1472 \def\sref@target{%
1473   \ifx\sref@id\empty%
1474     \relax%
1475   \else%
1476     \edef\@target{\sref@ifcsundef{sref@part}}{\sref@part @}\sref@id @target}%
1477     \sref@target@ifh\@target{}%
1478   \fi%
1479 }%

```

`\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.

```

1480 \addmetakey\srefaddidkey{prefix}
1481 \newcommand\srefaddidkey[2][]{%
1482   \metasetkeys{srefaddidkey}{#1}%
1483   \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
1484   \metakeys@ext@clear@keys{#2}{id}{}%
1485   \metakeys@ext@showkeys{#2}{id}%
1486   \define@key{#2}{id}{%
1487     \edef\sref@id{\srefaddidkey@prefix ##1}%
1488     \expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%

```

```

1489     \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1490   }%
1491 }%

```

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

```

1492 \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.

```

1493 \ifextrefs%
1494   \newwrite\refs@file%
1495 \else%
1496   \def\refs@file{\@auxout}%
1497 \fi%

```

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

```

1498 \newcommand\sref@def[3]{%
1499   \protected@write\refs@file{}{\string\sref@def{#1}{#2}{#3}}%
1500 }%

```

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

```

1501 \newcommand\sref@label[2]{%
1502   \sref@def{\ifcsundef{sref@part}}{\sref@part @}{#2}{page}{\thepage}%
1503   \sref@def{\ifcsundef{sref@part}}{\sref@part @}{#2}{label}{#1}%
1504 }%

```

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

```

1505 \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.

```

1506 \def\sref@id{} % make sure that defined
1507 \newcommand\sref@label@id[1]{%
1508   \ifx\sref@id\@empty%
1509     \relax%
1510   \else%
1511     \sref@label{#1}{\sref@id}%
1512   \fi%
1513 }%

```

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

```

1514 \newcommand\sref@label@id@arg[2]{%
1515   \def\@id{#2}
1516   \ifx\@id\@empty%
1517     \relax%

```

```

1518 \else%
1519 \sref@label{#1}{\@@id}%
1520 \fi%
1521 }%

```

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.

```

1522 \newenvironment{modsig}[2] [] {\def\@test{#1}%
1523 \ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1524 \expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
1525 \ignorespacesandpars}
1526 {\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`.

```

1527 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1528 \newrobustcmd\@gimport@star[2] [] {\def\@test{#1}%
1529 \edef\mh@@repos{\mh@currentrepos}%
1530 \ifx\@test\@empty%
1531 \importmhmodule[conservative,mhrepos=\mh@@repos,path=#2]{#2}%
1532 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1533 \setcurrentreposinfo{\mh@@repos}%
1534 \ignorespacesandpars\parsemodule@maybesetcodes}
1535 \newrobustcmd\@gimport@nostar[2] [] {\def\@test{#1}%
1536 \edef\mh@@repos{\mh@currentrepos}%
1537 \ifx\@test\@empty%
1538 \importmhmodule[mhrepos=\mh@@repos,path=#2]{#2}%
1539 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1540 \setcurrentreposinfo{\mh@@repos}%
1541 \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 practice 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.

```

1542 \def\modules@@first#1/#2;{#1}
1543 \newcommand\libinput[1]{%
1544 \ifcsvoid{mh@currentrepos}{%
1545   \PackageError{mathhub}{current MathHub repository not found}{}%
1546   {}
1547 \edef\@mh@group{\expandafter\modules@@first\mh@currentrepos;}
1548 \let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
1549 \def\mh@inffile{\MathHub{\@mh@group/meta-inf/lib/#1}}
1550 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1551 \IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
1552 \IfFileExists\mh@inffile{\IfFileExists\mh@libfile}{%
1553   {\PackageError{mathhub}
1554     {Library file missing; cannot input #1.tex\MessageBreak%
1555     Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
1556     do not exist}%
1557   {Check whether the file name is correct}}}%
1558 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1559 \let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}

```

3.11 omdoc/omgroup

```

1560 \newcount\section@level
1561
1562 \section@level=2
1563 \ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{}
1564 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{}
1565 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{}
1566 \ifdefstring{\omdoc@sty@topsect}{chapter}{\section@level=1}{}

```

`\omgroup@nonum` convenience macro: `\omgroup@nonum{<level>}{<title>}` makes an unnumbered sectioning with title *<title>* at level *<level>*.

```

1567 \newcommand\omgroup@nonum[2]{%
1568 \ifx\hyper@anchor\undefined\else\phantomsection\fi%
1569 \addcontentsline{toc}{#1}{#2}\@nameuse{#1}*{#2}}

```

`\omgroup@num` convenience macro: `\omgroup@num{<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 `rdfmata` package has been loaded. In the end we call `\sref@label@id` to enable crossreferencing.

```

1570 \newcommand\omgroup@num[2]{%
1571 \edef\@ID{\sref@id}
1572 \ifx\omgroup@short\empty% no short title

```

```

1573 \@nameuse{#1}{#2}%
1574 \else% we have a short title
1575 \@ifundefined{rdfmeta@sectioning}%
1576   {\@nameuse{#1}[\omgroup@short]{#2}}%
1577   {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
1578 \fi%
1579 \sref@label{id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@@ID}

```

omgroup

```

1580 \def\@true{true}
1581 \def\@false{false}
1582 \srefaddidkey{omgroup}
1583 \addmetakey{omgroup}{date}
1584 \addmetakey{omgroup}{creators}
1585 \addmetakey{omgroup}{contributors}
1586 \addmetakey{omgroup}{srccite}
1587 \addmetakey{omgroup}{type}
1588 \addmetakey*{omgroup}{short}
1589 \addmetakey*{omgroup}{display}
1590 \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 omgrou, i.e. after the section heading.

```

1591 \newif\if@mainmatter\@mainmattertrue
1592 \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.

```

1593 \addmetakey{omdoc@sect}{name}
1594 \addmetakey[false]{omdoc@sect}{clear}[true]
1595 \addmetakey{omdoc@sect}{ref}
1596 \addmetakey[false]{omdoc@sect}{num}[true]
1597 \newcommand\omdoc@sectioning[3][]{\metasetkeys{omdoc@sect}{#1}%
1598 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
1599 \if@mainmatter% numbering not overridden by frontmatter, etc.
1600 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
1601 \def\current@section@level{\omdoc@sect@name}%
1602 \else\omgroup@nonum{#2}{#3}%
1603 \fi}% if@mainmatter

```

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

```

1604 \newcommand\omgroup@redefine@addtocontents[1]{%
1605 %\edef\@import{#1}%
1606 %\@for\@I:=\@import\do{%
1607 %\edef\@path{\csname module@\@I @path\endcsname}%
1608 %\@ifundefined{tf@toc}\relax%
1609 %   {\protected@write\tf@toc}{\string\@requiremodules{\@path}}}%
1610 %\ifx\hyper@anchor\@undefined% hyperref.sty loaded?

```

```

1611 %\def\addcontentsline##1##2##3{%
1612 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##3}}{\thepage}}
1613 %\else% hyperref.sty not loaded
1614 %\def\addcontentsline##1##2##3{%
1615 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##3}}{\thepage}}{\c
1616 %\fi
1617 }% 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 command from `article.cls`. It also registers the current level of `omgroups` in the `\omgroup@level` counter.

```

1618 \newcount\omgroup@level
1619 \newenvironment{omgroup}[2][ ]% keys, title
1620 {\metasetkeys{omgroup}{##1}\sref@target%
1621 \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.

```

1622 \ifx\omgroup@loadmodules\@true%
1623 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
1624 {\@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`.

```

1625 \advance\section@level by 1\relax%
1626 \ifcase\section@level%
1627 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{##2}%
1628 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{##2}%
1629 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{##2}%
1630 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{##2}%
1631 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{##2}%
1632 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{##2}%
1633 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{##2}%
1634 \fi% \ifcase
1635 \at@begin@omgroup[##1]\section@level{##2}}% for customization
1636 {\advance\section@level by -1\advance\omgroup@level by -1}

```

and finally, we localize the sections

```

1637 \newcommand\omdoc@part@kw{Part}
1638 \newcommand\omdoc@chapter@kw{Chapter}
1639 \newcommand\omdoc@section@kw{Section}
1640 \newcommand\omdoc@subsection@kw{Subsection}
1641 \newcommand\omdoc@subsubsection@kw{Subsubsection}
1642 \newcommand\omdoc@paragraph@kw{paragraph}
1643 \newcommand\omdoc@subparagraph@kw{subparagraph}

```

`\setSGvar` set a global variable

```

1644 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}

```



```

\useSGvar use a global variable
1645 \newrobustcmd\useSGvar[1]{%
1646   \@ifundefined{sTeX@Gvar@#1}
1647   {\PackageError{omdoc}
1648     {The sTeX Global variable #1 is undefined}
1649     {set it with \protect\setSGvar}}
1650 \@nameuse{sTeX@Gvar@#1}}

```

blindomgroup

```

1651 \newcommand\at@begin@blindomgroup[1]{%
1652 \newenvironment{blindomgroup}
1653 {\advance\section@level by 1\at@begin@blindomgroup\section@level}
1654 {\advance\section@level by -1}}

```

3.12 omtex

3.12.1 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` infrastructure [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).

```

1655 \srefaddidkey{omtext}
1656 \addmetakey[] {omtext} {functions}
1657 \addmetakey* {omtext} {display}
1658 \addmetakey {omtext} {for}
1659 \addmetakey {omtext} {from}
1660 \addmetakey {omtext} {type}
1661 \addmetakey* {omtext} {title}
1662 \addmetakey* {omtext} {start}
1663 \addmetakey {omtext} {theory}
1664 \addmetakey {omtext} {continues}
1665 \addmetakey {omtext} {verbalizes}
1666 \addmetakey {omtext} {subject}

```

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

```

1667 \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.

```

1668 \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.

```

1669 \def\omtext@pre@skip{\smallskip}
1670 \def\omtext@post@skip{}

```

```

1671 \newenvironment{omtext}[1][\@in@omtexttrue%
1672   \bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
1673   \def\lec##1{\@lec{##1}}%
1674   \omtext@pre@skip\par\noindent%
1675   \ifx\omtext@title\@empty%
1676     \ifx\omtext@start\@empty\else%
1677       \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
1678     \fi% end omtext@start empty
1679   \else\stDMemph{\omtext@title}:\enspace%
1680   \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
1681   \fi% end omtext@title empty
1682   \ignorespacesandpars}
1683 {\egroup\omtext@post@skip\@in@omtextfalse\ignorespacesandpars}

```

3.12.2 Phrase-level Markup

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

```

1684 \srefaddidkey{phrase}
1685 \addmetakey{phrase}{style}
1686 \addmetakey{phrase}{class}
1687 \addmetakey{phrase}{index}
1688 \addmetakey{phrase}{verbalizes}
1689 \addmetakey{phrase}{type}
1690 \addmetakey{phrase}{only}
1691 \newcommand\phrase[2][\metasetkeys{phrase}{#1}%
1692 \ifx\prhase@only\@empty\only<\phrase@only>{#2}\else #2\fi}

```

`\coref*`

```

1693 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}
1694 \newcommand\corefs[2]{#1\textsubscript{#2}}
1695 \newcommand\coreft[2]{#1\textsuperscript{#2}}

```

`\n*lex`

```

1696 \newcommand\nlex[1]{\green{\sl{#1}}}
1697 \newcommand\nlcex[1]{*\green{\sl{#1}}}

```

`sinlinequote`

```

1698 \def\@sinlinequote#1{‘‘{\sl{#1}}’’}
1699 \def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
1700 \newcommand\sinlinequote[2][
1701 {\def\@opt{#1}\ifx\@opt\@empty\@sinlinequote{#2}\else\@@sinlinequote\@opt{#2}\fi}

```

3.12.3 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.

```

1702 \newcommand\vdec[2][\{#2}

```

```
1703 \newcommand\vest[2] [] {#2}
1704 \newcommand\vcond[2] [] {#2}
```

EdN:1

```
\strucdec 1
1705 \newcommand\strucdec[2] [] {#2}
```

EdN:2

```
\impdec 2
1706 \newcommand\impdec[2] [] {#2}
```

3.12.4 Block-Level Markup

`sblockquote`

```
1707 \def\begin@sblockquote{\begin{quote}\sl}
1708 \def\end@sblockquote{\end{quote}}
1709 \def\begin@@sblockquote#1{\begin@sblockquote}
1710 \def\end@@sblockquote#1{\def@@lec#1{\textrm{##1}}\@lec{#1}\end@sblockquote}
1711 \newenvironment{sblockquote}[1] []
1712 {\def@opt{#1}\ifx@opt@empty\begin@sblockquote\else\begin@@sblockquote@opt\fi}
1713 {\ifx@opt@empty\end@sblockquote\else\end@@sblockquote@opt\fi}
```

`sboxquote`

```
1714 \newenvironment{sboxquote}[1] []
1715 {\def@@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
1716 {\@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.

```
1717 \providecommand{\@@lec}[1]{(##1)}
1718 \def@lec#1{\strut\hfil\strut\null\nobreak\hfill@lec{#1}}
1719 \def\lec#1{@lec{#1}\par}
```

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

```
1720 \addmetakey{omdoc@index}{at}
1721 \addmetakey[false]{omdoc@index}{loadmodules}[true]
```

¹EdNOTE: document above

²EdNOTE: document above

```

1722 \newcommand\omdoc@indexi[2][\ifindex%
1723 \metasetkeys{omdoc@index}{#1}%
1724 \@bsphack\begingroup\@sanitize%
1725 \protected@write\@indexfile{\string\indexentry%
1726 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1727 \ifx\omdoc@index@loadmodules\@true%
1728 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
1729 \else #2\fi% loadmodules
1730 }\thepage}}%
1731 \endgroup\@esphack\fi}%ifindex
1732 \newcommand\omdoc@indexii[3][\ifindex%
1733 \metasetkeys{omdoc@index}{#1}%
1734 \@bsphack\begingroup\@sanitize%
1735 \protected@write\@indexfile{\string\indexentry%
1736 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1737 \ifx\omdoc@index@loadmodules\@true%
1738 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1739 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
1740 \else #2!#3\fi% loadmodules
1741 }\thepage}}%
1742 \endgroup\@esphack\fi}%ifindex
1743 \newcommand\omdoc@indexiii[4][\ifindex%
1744 \metasetkeys{omdoc@index}{#1}%
1745 \@bsphack\begingroup\@sanitize%
1746 \protected@write\@indexfile{\string\indexentry%
1747 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1748 \ifx\omdoc@index@loadmodules\@true%
1749 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1750 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1751 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1752 \else #2!#3!#4\fi% loadmodules
1753 }\thepage}}%
1754 \endgroup\@esphack\fi}%ifindex
1755 \newcommand\omdoc@indexiv[5][\ifindex%
1756 \metasetkeys{omdoc@index}{#1}%
1757 \@bsphack\begingroup\@sanitize%
1758 \protected@write\@indexfile{\string\indexentry%
1759 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1760 \ifx\omdoc@index@loadmodules\@true%
1761 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1762 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1763 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1764 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
1765 \else #2!#3!#4!#5\fi% loadmodules
1766 }\thepage}}%
1767 \endgroup\@esphack\fi}%ifindex

```

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

indi

```

1768 \newcommand\aindi[3] []{{#2}\omdoc@indexi[#1]{#3}}
1769 \newcommand\indi[2] []{{#2}\omdoc@indexi[#1]{#2}}
1770 \newcommand\indis[2] []{{#2}\omdoc@indexi[#1]{#2s}}
1771 \newcommand\Indi[2] []{{\captitalize{#2}}\omdoc@indexi[#1]{#2}}
1772 \newcommand\Indis[2] []{{\capitalize{#2}}\omdoc@indexi[#1]{#2s}}
1773
1774 \newcommand\@indii[3] []{\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
1775 \newcommand\aindii[4] []{{#2}\@indii[#1]{#3}{#4}}
1776 \newcommand\indii[3] []{{#2 #3}\@indii[#1]{#2}{#3}}
1777 \newcommand\indiis[3] []{{#2 #3s}\@indii[#1]{#2}{#3}}
1778 \newcommand\Indii[3] []{{\captitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1779 \newcommand\Indiis[3] []{{\capitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1780
1781 \newcommand\@indiii[4] []{\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexii[#1]{#3}{#2 (#4)}}
1782 \newcommand\aindiii[5] []{{#2}\@indiii[#1]{#3}{#4}{#5}}
1783 \newcommand\indiii[4] []{{#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
1784 \newcommand\indiis[4] []{{#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}
1785 \newcommand\Indiii[4] []{{\captitalize{#2 #3 #4}}\@indiii[#1]{#2}{#3}{#4}}
1786 \newcommand\Indiis[4] []{{\capitalize{#2 #3 #4s}}\@indiii[#1]{#2}{#3}{#4}}
1787
1788 \newcommand\@indiv[5] []{\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
1789 \newcommand\aindiv[6] []{{#2}\@indiv[#1]{#3}{#4}{#5}{#6}}
1790 \newcommand\indiv[5] []{{#2 #3 #4 #5}\@indiv[#1]{#2}{#3}{#4}{#5}}
1791 \newcommand\indivs[5] []{{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
1792 \newcommand\Indiv[5] []{{\capitalize{#2 #3 #4 #5s}}\@indiv[#1]{#2}{#3}{#4}{#5}}
1793 \newcommand\Indivs[5] []{{\capitalize{#2 #3 #4 #5s}}\@indiv[#1]{#2}{#3}{#4}{#5}}

```

3.12.6 Miscellaneous

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

```

1794 \newcommand\hateq{\ensuremath{\widehat{=}}\xspace}
1795 \newcommand\hatequiv{\ensuremath{\widehat{\equiv}}\xspace}
1796 \@ifundefined{ergo}%
1797 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1798 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1799 \newcommand\reflect@squig[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%
1800 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
1801 \newcommand\notergo{\ensuremath{\not\leadsto}}
1802 \newcommand\notogre{\ensuremath{\not\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%

```

3.12.7 Deprecated Functionality

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

def

```

1803 \newcommand\indextoo[2] []{\indi[#1]{#2}}%
1804 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}

```

```

1805 \newcommand\indexalt[2] [] {\aindi[#1]{#2}%
1806 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead}%
1807 \newcommand\twintoo[3] [] {\indii[#1]{#2}{#3}%
1808 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}%
1809 \newcommand\twinalt[3] [] {\aindii[#1]{#2}{#3}%
1810 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead}%
1811 \newcommand\atwintoo[4] [] {\indiii[#1]{#2}{#3}{#4}%
1812 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead}%
1813 \newcommand\atwinalt[4] [] {\aindii[#1]{#2}{#3}{#4}%
1814 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instead}%
1815 \</package>

```

`\my*graphics`

```

1816 \newcommand\mygraphics[2] [] {\includegraphics[#1]{#2}%
1817 \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics}%
1818 \newcommand\mycgraphics[2] [] {\begin{center}\mygraphics[#1]{#2}\end{center}%
1819 \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics}%
1820 \newcommand\mybgraphics[2] [] {\fbox{\mygraphics[#1]{#2}}%
1821 \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics}%
1822 \newcommand\mycbgraphics[2] [] {\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}%
1823 \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphics}%

```

4 Things to deprecate

Module options:

```

1824 \addmetakey*{module}{id} % TODO: deprecate properly
1825 \addmetakey*{module}{load}
1826 \addmetakey*{module}{path}
1827 \addmetakey*{module}{dir}
1828 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
1829 \addmetakey*{module}{noalign}[true]
1830
1831 \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 L^AT_EX part.

```

1832 %\srefaddidkey{symdef}% what does this do?
1833 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
1834 \define@key{symdef}{noverb}[all]{}%
1835 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
1836 \define@key{symdef}{specializes}{}%
1837 \addmetakey*{symdef}{noalign}[true]

```

```

1838 \define@key{symdef}{primary}[true]{}%
1839 \define@key{symdef}{assocarg}{}%
1840 \define@key{symdef}{bvars}{}%
1841 \define@key{symdef}{bargs}{}%
1842 \addmetakey{symdef}{lang}%
1843 \addmetakey{symdef}{prec}%
1844 \addmetakey{symdef}{arity}%
1845 \addmetakey{symdef}{variant}%
1846 \addmetakey{symdef}{ns}%
1847 \addmetakey{symdef}{args}%
1848 \addmetakey{symdef}{name}%
1849 \addmetakey*{symdef}{title}%
1850 \addmetakey*{symdef}{description}%
1851 \addmetakey{symdef}{subject}%
1852 \addmetakey*{symdef}{display}%
1853 \addmetakey*{symdef}{gfc}%

```

EdN:3

3

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

```

1854 \def\symdef{\ifnextchar[{\@symdef}{\@symdef []}}%
1855 \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.

```

1856 \def\@@symdef[#1]#2[#3]{%
1857   \@insymdef@true%
1858   \metasetkeys{symdef}{#1}%
1859   \edef\symdef@tmp@optpars{\ifcvoid{symdef@name}{[]}{[name=\symdef@name]}}%
1860   \expandafter\symdecl\symdef@tmp@optpars{#2}%
1861   \@insymdef@false%
1862   \notation[#1]{#2}{#3}%
1863 }% mod@show
1864 \def\symdef@type{Symbol}%
1865 \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`.

```

1866 \def\symvariant#1{%
1867   \ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}%
1868   }%
1869 \def\@symvariant#1[#2]#3#4{%
1870   \notation[#3]{#1}{#2}{#4}%
1871 \ignorespacesandpars}%

```

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

`\abbrdef` The `\abbrdef` macro is a variant of `\symdef` that does the same on the L^AT_EX level.

```

1872 \let\abbrdef\symdef%

```

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

```

1873 \newif\if@importing\@importingfalse
1874 \define@key{symi}{noverb}[all]{}%
1875 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
1876 \define@key{symi}{specializes}{}%
1877 \define@key{symi}{gfc}{}%
1878 \define@key{symi}{noalign}[true]{}%
1879 \newcommand\symi{\@ifstar\@symi@star\@symi}
1880 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
1881   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces}
1882 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
1883   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignorespaces}
1884 \newcommand\symii{\@ifstar\@symii@star\@symii}
1885 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
1886   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespaces}
1887 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
1888   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\ignorespaces}
1889 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
1890 \newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%
1891   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignorespaces}
1892 \newcommand\@symiii@star[4][]{\metasetkeys{symi}{#1}%
1893   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi\ignorespaces}
1894 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
1895 \newcommand\@symiv[5][]{\metasetkeys{symi}{#1}%
1896   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces}
1897 \newcommand\@symiv@star[5][]{\metasetkeys{symi}{#1}%
1898   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces}

```

`\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` comparison with an `\expandafter`, since the values may be passed on from other key bindings. Parameters will be passed to `\importmodule`.

```

1899 %\srefaddidkey{importmhmodule}%
1900 \addmetakey{importmhmodule}{mhrepos}%
1901 \addmetakey{importmhmodule}{path}%
1902 \addmetakey{importmhmodule}{ext}% why does this exist?
1903 \addmetakey{importmhmodule}{dir}%
1904 \addmetakey[false]{importmhmodule}{conservative}[true]%
1905 \newcommand\importmhmodule[2][]{%
1906   \parsemodule@maybesetcodes
1907   \metasetkeys{importmhmodule}{#1}%
1908   \ifx\importmhmodule@dir\empty%

```



```

1909 \edef\@path{\importmhmodule@path}%
1910 \else\edef\@path{\importmhmodule@dir/#2}\fi%
1911 \ifx\@path\@empty% if module name is not set
1912 \importmodule[] {#2}{export}%
1913 \else%
1914 \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
1915 \ifx\importmhmodule@mhrepos\@empty% if in the same repos
1916 \relax% no need to change mh@currentrepos, i.e., current directory.
1917 \else%
1918 \setcurrentreposinfo\importmhmodule@mhrepos% change it.
1919 \addto@thismodule\{noexpand\setcurrentreposinfo{\importmhmodule@mhrepos}}%
1920 \fi%
1921 \@importmodule[\MathHub{\mh@currentrepos/source/\@path}] {#2}{export}%
1922 \setcurrentreposinfo\mh@@repos% after importing, reset to old value
1923 \addto@thismodule\{noexpand\setcurrentreposinfo{\mh@@repos}}%
1924 \fi%
1925 \ignorespacesandpars%
1926 }

\usemhmodule

1927 \addmetakey{importmhmodule}{load}
1928 \addmetakey{importmhmodule}{id}
1929 \addmetakey{importmhmodule}{dir}
1930 \addmetakey{importmhmodule}{mhrepos}
1931
1932 \addmetakey{importmodule}{load}
1933 \addmetakey{importmodule}{id}
1934
1935 \newcommand\usemhmodule[2] [] {%
1936 \metasetkeys{importmhmodule}{#1}%
1937 \ifx\importmhmodule@dir\@empty%
1938 \edef\@path{\importmhmodule@path}%
1939 \else\edef\@path{\importmhmodule@dir/#2}\fi%
1940 \ifx\@path\@empty%
1941 \usemodule[id=\importmhmodule@id] {#2}%
1942 \else%
1943 \edef\mh@@repos{\mh@currentrepos}%
1944 \ifx\importmhmodule@mhrepos\@empty%
1945 \else\setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
1946 \usemodule{\@path\@QuestionMark#2}%
1947 %\usemodule[load=\MathHub{\mh@currentrepos/source/\@path},
1948 % id=\importmhmodule@id] {#2}%
1949 \setcurrentreposinfo\mh@@repos%
1950 \fi%
1951 \ignorespacesandpars}

\mhinputref

1952 \newcommand\mhinputref[2] [] {%
1953 \edef\mhinputref@first{#1}%
1954 \ifx\mhinputref@first\@empty%

```

```

1955     \inputref{#2}%
1956   \else%
1957     \inputref[mhrepos=\mhinputref@first]{#2}%
1958   \fi%
1959 }

\trefi*
1960 \newcommand\trefi[2][]{%
1961   \edef\trefi@mod{#1}%
1962   \ifx\trefi@mod@empty\tref{#2}\else\tref{#1\@QuestionMark#2}\fi%
1963 }
1964 \newcommand\trefii[3][]{%
1965   \edef\trefii@mod{#1}%
1966   \ifx\trefii@mod@empty\tref{#2-#3}\else\tref{#1\@QuestionMark#2-#3}\fi%
1967 }

\defi*
1968 \def\defii#1#2{\defi{#1!#2}}
1969 \def\Defii#1#2{\Defi{#1!#2}}
1970 \def\defiis#1#2{\defis{#1!#2}}
1971 \def\Defiis#1#2{\Defis{#1!#2}}
1972 \def\defiii#1#2#3{\defi{#1!#2!#3}}
1973 \def\Defiii#1#2#3{\Defi{#1!#2!#3}}
1974 \def\defiiis#1#2#3{\defis{#1!#2!#3}}
1975 \def\Defiiis#1#2#3{\Defis{#1!#2!#3}}
1976 \def\defiv#1#2#3#4{\defi{#1!#2!#3!#4}}
1977 \def\Defiv#1#2#3#4{\Defi{#1!#2!#3!#4}}
1978 \def\defivs#1#2#3#4{\defis{#1!#2!#3!#4}}
1979 \def\Defivs#1#2#3#4{\Defis{#1!#2!#3!#4}}
1980 \def\adefi#1#2{\defi[name=#2]{#1}}
1981 \def\adefii#1#2#3{\defi[name=#2-#3]{#1}}
1982 \def\adefiii#1#2#3#4{\defi[name=#2-#3-#4]{#1}}
1983 \def\adefiv#1#2#3#4#5{\defi[name=#2-#3-#4-#5]{#1}}

1984 \newlinechar=\old@newlinechar

```