

# stex-master.sty: $\text{\TeX}$ 2.0\*

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

December 17, 2020

## **Abstract**

TODO

---

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

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>User commands</b>	<b>3</b>
<b>3</b>	<b>Implementation</b>	<b>3</b>
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
<b>4</b>	<b>Things to deprecate</b>	<b>54</b>

# 1 Introduction

TODO

## 2 User commands

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

## 3 Implementation

```
1 <*package>
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

A conditional for LaTeXML:
15 \ifcname if@latexml\endcname\else
16 \expandafter\newif\cname if@latexml\endcname\@latexmlfalse
17 \fi
```

```

18 \RequirePackage{xspace}
19 \if@latexml\else\RequirePackage{standalone}\RequirePackage{metakeys}\fi

```

### 3.1 sTeX base

The sTeX logo:

```

20 \protected\def\stex{%
21   \@ifundefined{texorpdfstring}%
22   {\let\texorpdfstring\@firstoftwo}%
23   }%
24   \texorpdfstring{\raisebox{-.5ex}{S}\kern-.5ex\TeX}{sTeX}\xspace%
25 }
26 \def\sTeX{\stex}

```

### 3.2 Paths and URIs

```

27 \if@latexml\else
28 \RequirePackage{xstring}
29 \RequirePackage{etoolbox}
30 \fi

```

`\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`.

```

31 \newrobustcmd\defpath[3][]{%
32   \expandafter\newcommand\csname #2\endcsname[1]{#3/#1}%
33 }%
34 \let\namespace\defpath

```

#### 3.2.1 Path Canonicalization

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

```

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

```

```

45 \def\pathsuris@resetcatcodes{%
46   \catcode'\#\pathsuris@oldcatcode@hash\relax%
47   \catcode'\/\pathsuris@oldcatcode@slash\relax%
48   \catcode'\:\pathsuris@oldcatcode@colon\relax%
49   \catcode'\?\pathsuris@oldcatcode@qm\relax%
50 }

```

We define some macros for later comparison.

```

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

```

\@cpath Canonicalizes (file) paths:

```

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

```

```

91     }{}%
92 }
93
94 \def\@cpath@loop{%
95     \IfSubStr\pathsuris@cpath@temp\@Slash{%
96         \StrCut\pathsuris@cpath@temp\@Slash\pathsuris@cpath@temp@a\pathsuris@cpath@temp%
97         \ifx\pathsuris@cpath@temp@a\@ToTop%
98             \ifx\@CanPath\@empty%
99                 \edef\@CanPath{\@ToTop}%
100             \else%
101                 \edef\@CanPath{\@CanPath\@Slash\@ToTop}%
102             \fi%
103             \@cpath@loop%
104         \else%
105         \ifx\pathsuris@cpath@temp@a\@Dot%
106             \@cpath@loop%
107         \else%
108         \IfBeginWith\pathsuris@cpath@temp\@ToTop{%
109             \StrBehind{\pathsuris@cpath@temp}{\@ToTop}[\pathsuris@cpath@temp]%
110             \IfBeginWith\pathsuris@cpath@temp\@Slash{%
111                 \edef\pathsuris@cpath@temp{\@CanPath\pathsuris@cpath@temp}%
112             }{%
113                 \ifx\@CanPath\@empty\else%
114                     \edef\pathsuris@cpath@temp{\@CanPath\@Slash\pathsuris@cpath@temp}%
115                 \fi%
116             }%
117             \def\@CanPath{}%
118             \@cpath@loop%
119         }{%
120             \ifx\@CanPath\@empty%
121                 \edef\@CanPath{\pathsuris@cpath@temp@a}%
122             \else%
123                 \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp@a}%
124             \fi%
125             \@cpath@loop%
126         }%
127     \fi\fi%
128 }{
129     \ifx\@CanPath\@empty%
130         \edef\@CanPath{\pathsuris@cpath@temp}%
131     \else%
132         \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp}%
133     \fi%
134 }%
135 }

```

**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.

```

136 \newcommand\cpath[1]{%
137   \cpath{#1}%
138   \@CanPath%
139 }
```

`\path@filename`

```

140 \def\path@filename#1#2{%
141   \edef\filename@oldpath{#1}%
142   \StrCount\filename@oldpath\@Slash[\filename@lastslash]%
143   \ifnum\filename@lastslash>0%
144     \StrBehind[\filename@lastslash]\filename@oldpath\@Slash[\filename@oldpath]%
145     \edef#2{\filename@oldpath}%
146   \else%
147     \edef#2{\filename@oldpath}%
148   \fi%
149 }
```

**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:

```

150 \newif\if@iswindows@\@iswindows@false
151 \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:

```

152 \newif\if@windowstopath@inpath@
153 \def\windows@to@path#1{
```

```

154 \@windowstopath@inpath@false
155 \def\windows@temp{}
156 \edef\windows@path{#1}
157 \ifx\windows@path\@empty\else
158 \expandafter\windows@path@loop\windows@path\windows@path@end
159 \fi
160 \let#1\windows@temp
161 }
162 \def\windows@path@loop#1#2\windows@path@end{
163 \def\windows@temp@b{#2}
164 \ifx\windows@temp@b\@empty
165 \def\windows@continue{}
166 \else
167 \def\windows@continue{\windows@path@loop#2\windows@path@end}
168 \fi
169 \if@windowstopath@inpath@
170 \ifx#1\@BackSlash
171 \edef\windows@temp{\windows@temp\@Slash}
172 \else
173 \edef\windows@temp{\windows@temp#1}
174 \fi
175 \else
176 \ifx#1:
177 \edef\windows@temp{\@Slash\windows@temp}
178 \@windowstopath@inpath@true
179 \else
180 \edef\windows@temp{\windows@temp#1}
181 \fi
182 \fi
183 \windows@continue
184 }

```

#### 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:

```

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

```



```

198     \def\windows@continue{}
199   \else
200     \def\windows@continue{\path@windows@loop#2\windows@path@end}
201   \fi
202   \if@windowstopath@inpath@
203     \ifx#1/
204       \edef\windows@temp{\windows@temp\@BackSlash}
205     \else
206       \edef\windows@temp{\windows@temp#1}
207     \fi
208   \else
209     \ifx#1/
210       \edef\windows@temp{\windows@temp:\@BackSlash}
211       \@windowstopath@inpath@true
212     \else
213       \edef\windows@temp{\windows@temp#1}
214     \fi
215   \fi
216   \windows@continue
217 }

```

**Test:**

Input: /C/foø/bar.baz

Output: C:\foø\bar.baz

### 3.2.3 Auxiliary methods

`\trimstring` Removes initial and trailing spaces from a string:

```

218 \def\trimstring#1{%
219   \edef\pathsuris@trim@temp{#1}%
220   \IfBeginWith\pathsuris@trim@temp\@Space{%
221     \StrGobbleLeft\pathsuris@trim@temp1[#1]%
222     \trimstring{#1}%
223   }{%
224     \IfEndWith\pathsuris@trim@temp\@Space{%
225       \StrGobbleRight\pathsuris@trim@temp1[#1]%
226       \trimstring{#1}%
227     }{%
228       \edef#1{\pathsuris@trim@temp}%
229     }%
230   }%
231 }

```

**Test:**

»bla blubb«

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

```

232 %\if@latexml\else
233 \def\kpsewhich#1#2{\begingroup%
234   \edef\kpsewhich@cmd{"|kpsewhich #2"}%

```

```

235 \everyeof{\noexpand}%
236 \catcode'\=12%
237 \edef#1{\@@input\kpsewhich@cmd\@Space}%
238 \trimstring#1%
239 \if@iswindows@\windows@to@path#1\fi%
240 \xdef#1{\expandafter\detokenize\expandafter{#1}}%
241 \endgroup}
242 %\fi

```

**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:

```

243 \edef\pwd@cmd{\if@iswindows@ -expand-var \@Percent CD\@Percent\else -var-value PWD\fi}
244 \kpsewhich\stex@maindir\pwd@cmd
245 \edef\stex@mainfile{\stex@maindir\@Slash\jobname}
246 \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:

```

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

```

```

274 \global\let\stex@currfile\stex@currfile%
275 \global\let\stex@currpath\stex@currpath%
276 \global\let\stex@currfilename\stex@currfilename%
277 \fi%
278 }

```

`\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:

```

279 \def\stexinput#1{%
280 \stexiffileexists{#1}{%
281 \stex@currfile@push\stex@temp@path%
282 \input{\stex@currfile}%
283 \stex@currfile@pop%
284 }%
285 {%
286 \PackageError{stex}{File does not exist (#1): \stex@temp@path}{}%
287 }%
288 }
289 \def\stexiffileexists#1#2#3{%
290 \edef\stex@temp@path{#1}%
291 \if@iswindows@\path@to@windows\stex@temp@path\fi%
292 \IfFileExists\stex@temp@path{#2}{#3}%
293 }
294 \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:

```

295 \kpsewhich\mathhub@path{--var-value MATHHUB}
296 \if@iswindows@\windows@to@path\mathhub@path\fi
297 \ifx\mathhub@path\@empty%
298 \PackageWarning{stex}{MATHHUB system variable not found or wrongly set}{%
299 \defpath{MathHub}{%
300 \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.

```

301 \def\findmanifest#1{
302 \cpath{#1}
303 \ifx\@CanPath\@Slash
304 \def\manifest@mf{}
305 \else\ifx\@CanPath\@empty
306 \def\manifest@mf{}

```

```

307 \else
308   \edef\@findmanifest@path{\@CanPath/MANIFEST.MF}
309   \if@iswindows@ \path@to@windows \@findmanifest@path \fi
310   \IfFileExists{\@findmanifest@path}{
311     \%message{MANIFEST.MF found at \@findmanifest@path}
312     \edef\manifest@mf{\@findmanifest@path}
313     \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
314   }{
315     \edef\@findmanifest@path{\@CanPath/META-INF/MANIFEST.MF}
316     \if@iswindows@ \path@to@windows \@findmanifest@path \fi
317     \IfFileExists{\@findmanifest@path}{
318       \%message{MANIFEST.MF found at \@findmanifest@path}
319       \edef\manifest@mf{\@findmanifest@path}
320       \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
321     }{
322       \edef\@findmanifest@path{\@CanPath/meta-inf/MANIFEST.MF}
323       \if@iswindows@ \path@to@windows \@findmanifest@path \fi
324       \IfFileExists{\@findmanifest@path}{
325         \%message{MANIFEST.MF found at \@findmanifest@path}
326         \edef\manifest@mf{\@findmanifest@path}
327         \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
328       }{
329         \findmanifest{\@CanPath/..}
330       }
331     \fi\fi
332 }

```

#### Test:

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

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

```

333 \def\split@manifest@key{
334   \IfSubStr{\manifest@line}{\@Colon}{
335     \StrBefore{\manifest@line}{\@Colon}[\manifest@key]
336     \StrBehind{\manifest@line}{\@Colon}[\manifest@line]
337     \trimstring\manifest@line
338     \trimstring\manifest@key
339   }{
340     \def\manifest@key{}
341   }
342 }

```

the next helper function iterates over lines in MANIFEST.MF

```

343 \def\parse@manifest@loop{
344   \ifeof\@manifest
345   \else
346     \read\@manifest to \manifest@line\relax
347     \edef\manifest@line{\expandafter\detokenize\expandafter{\manifest@line}}
348     \split@manifest@key
349     % id

```

```

350 \IfStrEq\manifest@key{\detokenize{id}}{
351 \xdef\manifest@mf@id{\manifest@line}
352 }{
353 % narration-base
354 \IfStrEq\manifest@key{\detokenize{narration-base}}{
355 \xdef\manifest@mf@narr{\manifest@line}
356 }{
357 % namespace
358 \IfStrEq\manifest@key{\detokenize{source-base}}{
359 \xdef\manifest@mf@ns{\manifest@line}
360 }{
361 \IfStrEq\manifest@key{\detokenize{ns}}{
362 \xdef\manifest@mf@ns{\manifest@line}
363 }{
364 % dependencies
365 \IfStrEq\manifest@key{\detokenize{dependencies}}{
366 \xdef\manifest@mf@deps{\manifest@line}
367 }{
368 }}}}
369 \parse@manifest@loop
370 \fi
371 }

```

`\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.

```

372 \newread\@manifest
373 \def\parsemanifest#1#2{%
374 \gdef\temp@archive@dir{}%
375 \findmanifest{#2}%
376 \begingroup%
377 \gdef\manifest@mf@id{}%
378 \gdef\manifest@mf@narr{}%
379 \gdef\manifest@mf@ns{}%
380 \gdef\manifest@mf@deps{}%
381 \openin\@manifest\manifest@mf%
382 \parse@manifest@loop%
383 \closein\@manifest%
384 \endgroup%
385 \if@iswindows@ \windows@to@path\manifest@mf\fi%
386 \cslet{#1id}\manifest@mf@id%
387 \cslet{#1narr}\manifest@mf@narr%
388 \cslet{#1ns}\manifest@mf@ns%
389 \cslet{#1deps}\manifest@mf@deps%
390 \ifcvoid\manifest@mf@id{-}%
391 \cslet{#1dir}\temp@archive@dir%
392 }%
393 }

```

**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.

```

394 \def\setcurrentreposinfo#1{%
395   \edef\mh@currentrepos{#1}%
396   \ifx\mh@currentrepos\empty%
397     \edef\currentrepos@dir{\@Dot}%
398     \def\currentrepos@narr{%
399       \def\currentrepos@ns{%
400         \def\currentrepos@id{%
401           \def\currentrepos@deps{%
402             \else%
403             \ifcsdef{mathhub@dir@\mh@currentrepos}{%
404               \@inmhrepostrue
405               \edef\mh@currentrepos{#1}%
406               \expandafter\let\expandafter\currentrepos@dir\csname mathhub@dir@#1\endcsname%
407               \expandafter\let\expandafter\currentrepos@narr\csname mathhub@narr@#1\endcsname%
408               \expandafter\let\expandafter\currentrepos@ns\csname mathhub@ns@#1\endcsname%
409               \expandafter\let\expandafter\currentrepos@deps\csname mathhub@deps@#1\endcsname%
410             }{%
411               \parsemanifest{currentrepos@}\MathHub{#1}}%
412             \@setcurrentreposinfo%
413             \ifcvoid{currentrepos@dir}\PackageError{stex}{No archive with %
414               name #1 found!}{make sure that #1 is directly in your MATHHUB folder %
415               and contains a MANIFEST.MF, either directly in #1 or in a meta-inf %
416               subfolder.}}{\@inmhrepostrue}%
417           }%
418         \fi%
419       }
420     }
421   \def\@setcurrentreposinfo{%
422     \edef\mh@currentrepos{\currentrepos@id}%
423     \ifcvoid{currentrepos@dir}{%
424       \csxdef{mathhub@dir@\currentrepos@id}{\currentrepos@dir}%
425       \csxdef{mathhub@narr@\currentrepos@id}{\currentrepos@narr}%
426       \csxdef{mathhub@ns@\currentrepos@id}{\currentrepos@ns}%
427       \csxdef{mathhub@deps@\currentrepos@id}{\currentrepos@deps}%
428     }%
429   }

```

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

```

430 \newif\if@inmhrepos\@inmhreposfalse
431 \ifcvoid{stex@maindir}{%
432   \parsemanifest{currentrepos@}\stex@maindir
433   \@setcurrentreposinfo

```

```

434 \ifcvoid{currentrepos@dir}{\PackageWarning{stex}{Not currently in a MathHub repository}}{}{}%
435 \message{Current repository: \mh@currentrepos}
436 }
437 }

```

### 3.3 Modules

```

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

```

Aux:

```

439 \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
440 \def\ignorespacesandparsafterend#1\ignorespaces\fi{#1\fi\ignorespacesandpars}
441 \def\ignorespacesandpars{\ifhmode\unskip\fi\@ifnextchar\par{\expandafter\ignorespacesandpars\g

```

Options for the module-environment:

```

442 \addmetakey*{module}{title}
443 \addmetakey*{module}{name}
444 \addmetakey*{module}{creators}
445 \addmetakey*{module}{contributors}
446 \addmetakey*{module}{srccite}
447 \addmetakey*{module}{ns}
448 \addmetakey*{module}{narr}

```

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

```

449 \ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%
450 \newrobustcmd\module@heading{%
451 \stepcounter{module}%
452 \ifmod@show%
453 \noindent{\textbf{Module} \thesection.\thetitle [\module@name]}%
454 \sref@label@id{Module \thesection.\thetitle [\module@name]}%
455 \ifx\module@title\@empty :\quad\else\quad(\module@title)\hfill\\\fi%
456 \fi%
457 }%

```

**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.

```

458 \newenvironment{module}[1][]{%
459 \begin{@module}[#1]%
460 \module@heading% make the headings
461 \ignorespacesandpars\parsemodule@maybesetcodes}%
462 \end{@module}%
463 \ignorespacesafterend%
464 }%
465 \ifmod@show\surroundwithmdframed{module@om@common}\fi%

```

Some auxiliary methods:

```

466 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}
467 \def\addto@thismodule#1{%
468   \@ifundefined{this@module}{}{%
469     \expandafter\g@addto@macro@safe\this@module{#1}%
470   }%
471 }
472 \def\addto@thismodulex#1{%
473   \@ifundefined{this@module}{}{%
474     \edef\addto@thismodule@exp{#1}%
475     \expandafter\expandafter\expandafter\g@addto@macro@safe%
476     \expandafter\this@module\expandafter{\addto@thismodule@exp}%
477 }}

```

**@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`.

```

478 \newif\ifarchive@ns@empty@\archive@ns@empty@false
479 \def\set@default@ns{%
480   \edef\@module@ns@temp{\stex@currpath}%
481   \if@iswindows@\windows@to@path\@module@ns@temp\fi%
482   \archive@ns@empty@false%
483   \ifcvoid{mh@currentrepos}\@archive@ns@empty@true}%
484   {\expandafter\ifx\cname mathhub@ns@\mh@currentrepos\endcsname\@empty\archive@ns@empty@true\fi}%
485 }%
486 \ifarchive@ns@empty%
487   \edef\@module@ns@tempuri{file\@Colon\@Slash\@Slash\@module@ns@temp}%
488 \else%
489   \edef\@module@filepath@temppath{\@module@ns@temp}%
490   \edef\@module@ns@tempuri{\cname mathhub@ns@\mh@currentrepos\endcsname}%
491   \edef\@module@archivedirpath{\cname mathhub@dir@\mh@currentrepos\endcsname\@Slash source}%
492   \edef\@module@archivedirpath{\expandafter\detokenize\expandafter{\@module@archivedirpath}}%
493   \IfBeginWith\@module@filepath@temppath\@module@archivedirpath{%
494     \StrLen\@module@archivedirpath[\ns@temp@length]%
495     \StrGobbleLeft\@module@filepath@temppath[\ns@temp@length]\@module@filepath@temprest}%
496   \edef\@module@ns@tempuri{\@module@ns@tempuri\@module@filepath@temprest}%
497   }{}%
498 \fi%
499 \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]}%
500 \setkeys{module}{ns=\@module@ns@tempuri}%
501 }

```



**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.

```

502 \def\set@next@moduleid{%
503   \unless\ifcsname namespace@\module@ns @unnamedmodules\endcsname%
504     \csgdef{namespace@\module@ns @unnamedmodules}{0}%
505   \fi%
506   \edef\namespace@currnum{\csname namespace@\module@ns @unnamedmodules\endcsname}%
507   \edef\module@temp@setidname{\noexpand\setkeys{module}{name=module\namespace@currnum}}%
508   \module@temp@setidname%
509   \csxdef{namespace@\module@ns @unnamedmodules}{\the\numexpr\namespace@currnum+1}%
510 }

```

**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.

```

511 \newenvironment{@module}[1][{}]{%
512   \metasetkeys{module}{#1}%
513   \ifcsvoid{module@name}{\let\module@name\module@id}{}}% % TODO deprecate
514   \ifx\module@ns\empty\set@default@ns\fi%
515   \ifx\module@narr\empty%
516     \setkeys{module}{narr=\module@ns}%

```

```

517 \fi%
518 \ifcvoid{module@name}{\set@next@moduleid}{}%
519 \let\module@id\module@name% % TODO deprecate
520 \edef\module@uri{\module@ns\@QuestionMark\module@name}%
521 \csgdef{module@names@\module@uri}{}%
522 \csgdef{module@imports@\module@uri}{}%
523 \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
524 \ifcvoid{stex@module@\module@name}{
525   \expandafter\global\expandafter\let\csname stex@module@\module@name\expandafter\endcsname\c
526 }{
527   \expandafter\edef\csname stex@module@\module@name\endcsname{\detokenize{ambiguous}}
528 }
529 \edef\this@module{%
530   \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
531 }%
532 \csdef{module@defs@\module@uri}{}%
533 \ifcvoid{mh@currentrepos}{}{%
534   \@inmhrepostrue%
535   \addto@thismodule{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\endcsname{
536     {\noexpand\mh@currentrepos}}%
537   \addto@thismodule{\noexpand\setcurrentreposinfo{\mh@currentrepos}}%
538 }%
539 }{%
540   \if@inmhrepos%
541     \@inmhreposfalse%
542     \addto@thismodule{\noexpand\setcurrentreposinfo{\expandafter\noexpand\csname mh@old@repos@\module@uri\endcsname{
543       {\noexpand\mh@currentrepos}}%
544 }%

```

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

```
545 \def\@URI{uri}
546 \def\@invoke@module#1#2{%
547   \ifx\@URI#2%
548     #1%
549   \else%
550     % TODO something else
551     #2%
552   \fi%
553 }
```

## 3.4 Inheritance

### 3.4.1 Selective Inclusion

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

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

```
554 \newif\if@smsmode\@smsmodefalse
555 \def\parsemodule@escapechar@allowed{true}
556 \def\parsemodule@allow#1{
557   \expandafter\let\csname parsemodule@allowedmacro@#1\endcsname\parsemodule@escapechar@allowed
558 }
559 \def\parsemodule@allowenv#1{
560   \expandafter\let\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed
561 }
562 \def\parsemodule@escapechar@beginstring{begin}
563 \def\parsemodule@escapechar@endstring{end}
```

and now we use that to actually register all the  $\text{\LaTeX}$  functionality as relevant for `sms` mode.

```
564 \parsemodule@allow{symdef}
565 \parsemodule@allow{abbrdef}
```

```

566 \parsemodule@allow{importmodule}
567 \parsemodule@allowenv{module}
568 \parsemodule@allow{importmhmodule}
569 \parsemodule@allow{gimport}
570 \parsemodule@allowenv{modsig}
571 \parsemodule@allowenv{mhmodsig}
572 \parsemodule@allowenv{mhmodnl}
573 \parsemodule@allowenv{modnl}
574 \parsemodule@allow{symvariant}
575 \parsemodule@allow{symi}
576 \parsemodule@allow{symii}
577 \parsemodule@allow{symiii}
578 \parsemodule@allow{symiv}
579 \parsemodule@allow{notation}
580 \parsemodule@allow{verbalization}
581 \parsemodule@allow{symdecl}
582
583 % to deprecate:
584
585 \parsemodule@allow{defi}
586 \parsemodule@allow{defii}
587 \parsemodule@allow{defiii}
588 \parsemodule@allow{defiv}
589 \parsemodule@allow{adefi}
590 \parsemodule@allow{adefii}
591 \parsemodule@allow{adefiii}
592 \parsemodule@allow{adefiv}
593 \parsemodule@allow{defis}
594 \parsemodule@allow{defiis}
595 \parsemodule@allow{defiiis}
596 \parsemodule@allow{defivs}
597 \parsemodule@allow{Defi}
598 \parsemodule@allow{Defii}
599 \parsemodule@allow{Defiii}
600 \parsemodule@allow{Defiv}
601 \parsemodule@allow{Defis}
602 \parsemodule@allow{Defiis}
603 \parsemodule@allow{Defiiis}
604 \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*).

```

605 \catcode'\.=0

```

```

606 .catcode'\=13
607 .def.@active@slash{\}
608 .catcode'<=1
609 .catcode'>=2
610 .catcode'_{=12
611 .catcode'._=12
612 .def.@open@brace{<{>
613 .def.@close@brace{>}
614 .catcode'\=0
615 \catcode'\.=12
616 \catcode'\{=1
617 \catcode'\}=2
618 \catcode'\<=12
619 \catcode'\>=12

```

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

`\set@parsemodule@catcodes`

```

620 \def\set@parsemodule@catcodes{%
621     \global\catcode'\=13%
622     \global\catcode'\#=12%
623     \global\catcode'\{=12%
624     \global\catcode'\}=12%
625     \global\catcode'\$=12%$
626     \global\catcode'\^=12%
627     \global\catcode'\_ =12%
628     \global\catcode'\&=12%
629     \expandafter\let\@active@slash\parsemodule@escapechar%
630 }

```

`\reset@parsemodule@catcodes`

```

631 \def\reset@parsemodule@catcodes{%
632     \global\catcode'\=0%
633     \global\catcode'\#=6%
634     \global\catcode'\{=1%
635     \global\catcode'\}=2%
636     \global\catcode'\$=3%$
637     \global\catcode'\^=7%
638     \global\catcode'\_ =8%
639     \global\catcode'\&=4%
640 }

```

`\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.

```

641 \def\parsemodule@maybesetcodes{%
642     \if@smsmode\set@parsemodule@catcodes\fi%
643 }

```

`\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:

```

644
645 \def\parsemodule@escapechar{%
646     \def\parsemodule@escape@currcls{}%
647     \parsemodule@escape@parse@nextchar%
648 }%

```

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`.

```

649 \long\def\parsemodule@escape@parse@nextchar@#1{%
650     \ifcat a#1\relax%
651         \edef\parsemodule@escape@currcls{\parsemodule@escape@currcls#1}%
652         \let\parsemodule@do@next\parsemodule@escape@parse@nextchar%
653     \else%
654         \def\parsemodule@last@char{#1}%
655         \ifx\parsemodule@escape@currcls\@empty%
656             \def\parsemodule@do@next{}%
657         \else%
658             \def\parsemodule@do@next{\parsemodule@escapechar@checkcls}%
659         \fi%
660     \fi%
661     \parsemodule@do@next%
662 }

```

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.

```

663 \def\parsemodule@escapechar@checkcls{%
664     \ifx\parsemodule@escape@currcls\parsemodule@escapechar@beginstring%
665         \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@last@char}%
666     \else%
667         \ifx\parsemodule@escape@currcls\parsemodule@escapechar@endstring%

```

```

668         \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@last@char}%
669     \else%
670         \expandafter\ifx\csname parsemodule@allowedmacro@\parsemodule@escape@curr\endcsname\relax%
671             \parsemodule@escapechar@allowed%
672         \ifx\parsemodule@last@char\@open@brace%
673             \expandafter\let\expandafter\parsemodule@do@next@ii\csname\parsemodule@escape@curr\endcsname%
674             \edef\parsemodule@do@next{\noexpand\parsemodule@converttoproperbraces\@open@brace}%
675         \else%
676             \reset@parsemodule@catcodes%
677             \edef\parsemodule@do@next{\expandafter\noexpand\csname\parsemodule@escape@curr\endcsname}%
678         \fi%
679     \else\def\parsemodule@do@next{\relax\parsemodule@last@char}\fi%
680 \fi%
681 \fi%
682 \parsemodule@do@next%
683 }

```

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.

```

684 \expandafter\expandafter\expandafter\def%
685 \expandafter\expandafter\expandafter\parsemodule@converttoproperbraces%
686 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
687     \reset@parsemodule@catcodes%
688     \parsemodule@do@next@ii{#1}%
689 }

```

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.

```

690 \expandafter\expandafter\expandafter\def%
691 \expandafter\expandafter\expandafter\parsemodule@escapechar@checkbeginenv%
692 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
693     \expandafter\ifx\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed%
694         \reset@parsemodule@catcodes%
695         \def\parsemodule@do@next{\begin{#1}}%
696     \else%
697         \def\parsemodule@do@next{#1}%
698     \fi%
699     \parsemodule@do@next%
700 }
701 \expandafter\expandafter\expandafter\def%
702 \expandafter\expandafter\expandafter\parsemodule@escapechar@checkendenv%
703 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
704     \expandafter\ifx\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed%

```

```

705      %\reset@parsemodule@catcodes%
706      \def\parsemodule@do@next{\end{#1}}%
707    \else%
708      \def\parsemodule@do@next{#1}%
709    \fi%
710    \parsemodule@do@next%
711 }

```

`\@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.

```

712 \newrobustcmd\@requiremodules[1]{%
713   \if@tempswa\requiremodules{#1}\fi%
714 }%

```

`\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.

```

715 \newrobustcmd\requiremodules[1]{%
716   \mod@showfalse%
717   \edef\mod@path{#1}%
718   \edef\mod@path{\expandafter\detokenize\expandafter{\mod@path}}%
719   \requiremodules@smsmode{#1}%
720 }%

```

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

```

721 \newbox\modules@import@tempbox
722 \def\requiremodules@smsmode#1{%
723   \setbox\modules@import@tempbox\vbox{%
724     \@smsmodetrue%
725     \set@parsemodule@catcodes%
726     \hbadness=100000\relax%
727     \hfuzz=10000pt\relax%
728     \vbadness=100000\relax%
729     \vfuzz=10000pt\relax%
730     \stexinput{#1.tex}%
731     \reset@parsemodule@catcodes%
732   }%
733   \parsemodule@maybesetcodes%
734 }

```

#### Test:

parsing `FOO/testmodule.tex`

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



### 3.4.2 importmodule

`\importmodule@bookkeeping`

```

735 \newif\if@importmodule@switchrepos\@importmodule@switchreposfalse
736 \def\importmodule@bookkeeping#1#2#3{%
737   \@importmodule@switchreposfalse%
738   \metasetkeys{importmodule}{#1}%
739   \ifcvoid{importmodule@mhrepos}{%
740     \ifcvoid{currentrepos@dir}{%
741       \let\importmodule@dir\stex@maindir%
742     }{%
743       \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
744     }%
745   }{%
746     \@importmodule@switchrepostrue%
747     \expandafter\let\csname importmodule@oldrepos@#2\endcsname\mh@currentrepos%
748     \setcurrentreposinfo\importmodule@mhrepos%
749     \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
750   }%
751   \StrCut{#2}\@QuestionMark\importmodule@subdir\importmodule@modulename%
752   \ifx\importmodule@modulename\empty%
753     \let\importmodule@modulename\importmodule@subdir%
754     \let\importmodule@subdir\empty%
755   \else%
756     \ifx\importmodule@subdir\empty\else%
757       \edef\importmodule@dir{\importmodule@dir\@Slash\importmodule@subdir}%
758     \fi%
759   \fi%
760   #3%
761   \if@importmodule@switchrepos%
762     \expandafter\setcurrentreposinfo\csname importmodule@oldrepos@#2\endcsname%
763   \fi%
764   \ignorespacesandpars%
765 }

```

`\importmodule`

```

766 %\srefaddidkey{importmodule}
767 \addmetakey{importmodule}{mhrepos}
768 \newcommand\importmodule[2][\@importmodule{#1}{#2}{export}]
769 \newcommand\@importmodule[3][\@importmodule@bookkeeping{#1}{#2}{%
770   \importmodule@bookkeeping{#1}{#2}{%
771     \@importmodule[\importmodule@dir]\importmodule@modulename{#3}%
772   }%
773 }

```

`\@importmodule` `\@importmodule[\<filepath>]{\<mod>}{\<export?>}` loads `\<filepath>.tex` and activates the module `\<mod>`. If `\<export?>` is `export`, then it also re-exports the `\symdefs` from `\<mod>`.

First `\load` will store the base file name with full path, then check if `\module@<mod>@path` is defined. If this macro is defined, a module of this name

has already been loaded, so we check whether the paths coincide, if they do, all is fine and we do nothing otherwise we give a suitable error. If this macro is undefined we load the path by `\requiremodules`.

```

774 \newcommand\@importmodule[3][\]{%
775 {%
776   \edef\@load{#1}%
777   \edef\@importmodule@name{#2}
778   \if@smsmode\else\ifcsvoid{stex@module@\@importmodule@name}{% TODO check this
779     \stexiffileexists\@load{\requiremodules\@load}{%
780       \requiremodules{\@load\@Slash\@importmodule@name}%
781     }%
782   }\fi%
783   \ifx\@load\@empty\else%
784     {% TODO
785       \edef\@path{csname module@#2@path\endcsname}%
786       \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do nothing
787       {\PackageError{stex}{% else signal an error
788         {Module Name Clash\MessageBreak%
789           A module with name #2 was already loaded under the path "\@path"\MessageBreak%
790           The imported path "\@load" is probably a different module with the\MessageBreak%
791           same name; this is dangerous -- not importing}%
792         {Check whether the Module name is correct}%
793       }%
794     }%
795   \fi%
796   \global\let\@importmodule@load\@load%
797 }%
798 \edef\@export{#3}\def\@@export{export}%prepare comparison
799 %\ifx\@export\@@export\export@defs{#2}\fi% export the module
800 \ifx\@export\@@export\addto@thismodulex{%
801   \noexpand\@importmodule[\@importmodule@load]{#2}{noexport}%
802 }%
803 \if@smsmode\else
804 \ifcsvoid{this@module}{\fi%
805   \ifcsvoid{module@imports@\module@uri}{%
806     \csxdef{module@imports@\module@uri}{%
807       \csname stex@module@#2\endcsname\@URI% TODO check this
808     }%
809   }\fi%
810   \csxdef{module@imports@\module@uri}{%
811     \csname stex@module@#2\endcsname\@URI,% TODO check this
812     \csname module@imports@\module@uri\endcsname%
813   }%
814 }%
815 }%
816 \fi\fi%
817 \if@smsmode\else\activate@defs{#2}\fi% activate the module
818 }%

```

**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 \@import@star \@gimport@nostar

```

Default document module:

```

819 \AtBeginDocument{%
820   \set@default@ns%
821   \ifx\module@narr\@empty\setkeys{module}{narr=\module@ns}\fi%
822   \let\module@name\jobname%
823   \let\module@id\module@name % TODO deprecate
824   \edef\module@uri{\module@ns\@QuestionMark\module@name}%
825   \csgdef{module@names@\module@uri}{}%
826   \csgdef{module@imports@\module@uri}{}%
827   \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
828   \expandafter\global\expandafter\let\csname stex@module@\module@name\expandafter\endcsname\csname
829   \edef\this@module{%
830     \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
831   }%
832   \csdef{module@defs@\module@uri}{}%
833   \ifcvoid{mh@currentrepos}{}%
834     \inmhrepostrue%
835     \addto@thismodule{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\endcsname
836       {\noexpand\mh@currentrepos}}%
837     \addto@thismodule{\noexpand\setcurrentreposinfo{\mh@currentrepos}}%
838   }%
839 }

```

`\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.

```

840 \def\activate@defs#1{%
841   \ifcsundef{stex@module@#1}{ % TODO check this
842     \PackageError{stex}{No module with name #1 loaded}{Probably missing an
843       \detokenize{\importmodule} (or variant) somewhere?

```

```

844     }
845   }{%
846     \ifcsundef{module@\csname stex@module@#1\endcsname\@URI @activated}%
847     {\csname module@defs@\csname stex@module@#1\endcsname\@URI\endcsname}{}}%
848     \namedef{module@\csname stex@module@#1\endcsname\@URI @activated}{true}%
849   }%
850 }%

\usemodule \usemodule acts like \importmodule, except that it does not re-export the se-
semantic macros in the modules it loads.
851 \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.
852 \def\inputref@preskip{}
853 \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).
854 \newrobustcmd\inputref[2] [] {%
855   \importmodule@bookkeeping{#1}{#2}{%
856     %\inputreftrue
857     \inputref@preskip%
858     \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
859     \inputref@postskip%
860   }%
861 }%

```

### 3.5 Symbols/Notations/Verbalizations

```

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

\define@in@module calls \edef\#1{#2} and adds the macro definition to \this@module
863 \def\define@in@module#1#2{
864   \expandafter\edef\csname #1\endcsname{#2}%
865   \edef\define@in@module@temp{%
866     \def\expandafter\noexpand\csname#1\endcsname%
867     {#2}%
868   }%
869   \if@symdeflocal\else%

```

```

870     \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
871     \expandafter\endcsname\expandafter{\define@in@module@temp}%
872     \fi%
873 }

\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.

874 \addmetakey{symdecl}{name}%
875 \addmetakey{symdecl}{verbalization}%
876
877 % constructs a symbol name and a verbalization by splitting at exclamation
878 % points - e.g. \symdecl{symmetric!group} leads to name=symmetric-group
879 % and verbalization "symmetric group".
880 \def\symdecl@constructname#1{%
881     \def\symdecl@name{}%
882     \def\symdecl@verbalization{}%
883     \edef\symdecl@tempname{#1}%
884     \symdecl@constructname@loop%
885 }
886
887 \def\symdecl@constructname@loop{%
888     \ifx\symdecl@tempname\@empty\else%
889         \StrCut\symdecl@tempname!\symdecl@tempfirst\symdecl@tempname%
890         \ifx\symdecl@name\@empty%
891             \let\symdecl@name\symdecl@tempfirst%
892             \let\symdecl@verbalization\symdecl@tempfirst%
893             \symdecl@constructname@loop%
894         \else%
895             \edef\symdecl@name{\symdecl@name-\symdecl@tempfirst}%
896             \edef\symdecl@verbalization{\symdecl@verbalization\@Space\symdecl@tempfirst}%
897             \symdecl@constructname@loop%
898         \fi%
899     \fi%
900 }
901
902 \newcommand\symdecl[2][]{%
903     \ifcsdef{this@module}%
904         \metasetkeys{symdecl}{#1}%
905         \ifcsvoid{symdecl@name}{%
906             \ifcsvoid{symdecl@verbalization}{%
907                 \symdecl@constructname{#2}%
908             }{%
909                 \edef\symdecl@name{#2}%
910             }%
911         }{%
912             \ifcsvoid{symdecl@verbalization}{\edef\symdecl@verbalization{#2}}{%
913             }%
914             \edef\symdecl@uri{\module@uri\@QuestionMark\symdecl@name}%

```

```

915 \ifcsvoid{stex@symbol@\symdecl@name}{
916   \expandafter\edef\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}
917 }{
918   \expandafter\def\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}
919 }
920 \edef\symdecl@symbolmacro{
921   \noexpand\ifcsvoid{stex@symbol@\symdecl@name}{
922     \expandafter\edef\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}
923   }{
924     \expandafter\def\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}
925   }
926 }
927 \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
928 \expandafter\endcsname\expandafter{\symdecl@symbolmacro}%
929 \ifcsvoid{\symdecl@uri}{
930   \ifcsvoid{module@names@\module@uri}{%
931     \csxdef{module@names@\module@uri}{\symdecl@name}%
932   }{%
933     \csxdef{module@names@\module@uri}{\symdecl@name,%
934       \csname module@names@\module@uri\endcsname}%
935   }%
936 }{%
937   % not compatible with circular dependencies, e.g. test/omdoc/07-modules/sms testa.tex
938   \PackageWarning{stex}{symbol already defined: \symdecl@uri}{%
939     You need to pick a fresh name for your symbol%
940   }%
941 }%
942 \define@in@module\symdecl@uri{\noexpand\@invoke@symbol{\symdecl@uri}}%
943 \define@in@module{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
944 \global\expandafter\let\csname\symdecl@uri\@Fragment verb\@Fragment\endcsname\symdecl@verb
945 }{%
946   \PackageError{stex}{\detokenize{\symdecl} not in a module}{You need to be in a module%
947     in order to declare a new symbol}
948 }%
949 \ifinsymdef@\else\parsemodule@maybesetcodes\fi%
950 }

```

**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:

```

951 \edef\stex@ambiguous{\detokenize{ambiguous}}
952 \edef\stex@macrostring{\detokenize{macro:->\@invoke@symbol}}
953 \def\modules@getURIfromName#1{%

```

```

954 \def\notation@uri{}%
955 \edef\modules@getURI@name{#1}%
956 \ifcsvoid{\modules@getURI@name}{
957   \edef\modules@temp@meaning{
958 }{
959   \edef\modules@temp@meaning{\expandafter\meaning\csname\modules@getURI@name\endcsname}
960 }
961 \IfBeginWith\modules@temp@meaning\stex@macrostring{
962   % is a \@invoke@symbol macro
963   \StrPosition\modules@temp@meaning\@close@brace[\stex@tempnum]
964   \StrMid\modules@temp@meaning{26}{\the\numexpr\stex@tempnum-1\@Space}{\notation@uri}
965 }{
966   % Check whether full URI or module?symbol or just name
967   \StrCount\modules@getURI@name\@QuestionMark[\isuri@number]
968   \ifnum\isuri@number=2
969     \edef\notation@uri{\modules@getURI@name}
970   \else
971     \ifnum\isuri@number=1
972       % module?name
973       \StrCut\modules@getURI@name\@QuestionMark\isuri@mod\isuri@name
974       \ifcsvoid{stex@module@\isuri@mod}{
975         \PackageError{stex}{No module with name \isuri@mod\@Space loaded}{}
976       }{
977         \expandafter\ifx\csname stex@module@\isuri@mod\endcsname\stex@ambiguous
978         \PackageError{stex}{Module name \isuri@mod\@Space is ambiguous}{}
979       }
980       \edef\notation@uri{\csname stex@module@\isuri@mod\endcsname\@URI\@QuestionMark\isuri@mod}
981     \fi
982   }
983   \else
984     %name
985     \ifcsvoid{stex@symbol@\modules@getURI@name}{
986       \PackageError{stex}{No symbol with name \modules@getURI@name\@Space known}{}
987     }{
988       \ifcsvoid{\module@uri\@QuestionMark\modules@getURI@name}{
989         \expandafter\ifx\csname stex@symbol@\modules@getURI@name\endcsname\stex@ambiguous
990         % Symbol name ambiguous and not in current module
991         \PackageError{stex}{Symbol name, URI or macroname \detokenize{#1} found!}{}%
992       }
993       \else
994         % Symbol not in current module, but unambiguous
995         \edef\notation@uri{\csname stex@symbol@\modules@getURI@name\endcsname}
996       \fi
997     }{ % Symbol in current module
998       \edef\notation@uri{\module@uri\@QuestionMark\modules@getURI@name}
999     }
1000   \fi
1001 }
1002 }
1003 }

```

```

\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}{...}

1004 \newif@if@inverbalization\@inverbalizationfalse
1005 % parses the first two arguments:
1006 \providerobustcmd\notation[2][ ]{%
1007   \edef\notation@first{#1}%
1008   \edef\notation@second{#2}%
1009   \notation@%
1010 }
1011
1012 \providerobustcmd\verbalization{%
1013   \@inverbalizationtrue%
1014   \notation%
1015 }
1016
1017 % parses the last two arguments
1018 \newcommand\notation@[2][0]{%
1019   \edef\notation@donext{\noexpand\notation@[ \notation@first]%
1020     {\notation@second}[#1]}%
1021   \notation@donext{#2}%
1022 }
1023
1024 % parses the notation arguments and wraps them in
1025 % \notation@assoc and \notation@argprec for flexary arguments and precedences
1026 \def\notation@[#1]#2[#3]#4{%
1027   \modules@getURIfromName{#2}%
1028   \notation@parse@params{#1}{#3}
1029   \let\notation@curr@todo@args\notation@curr@args%
1030   \def\notation@temp@notation{%
1031     \StrLen\notation@curr@args[\notation@temp@arity]%
1032     \expandafter\renewcommand\expandafter\notation@temp@notation%
1033       \expandafter[\notation@temp@arity]{#4}%
1034     % precedence
1035     \IfSubStr\notation@curr@prec;{%
1036       \StrCut\notation@curr@prec;\notation@curr@prec\notation@curr@prec%
1037       \ifx\notation@curr@prec\@empty\def\notation@curr@prec{0}\fi%
1038     }{%
1039       \ifx\notation@curr@prec\@empty%
1040         \ifnum\notation@temp@arity=0\relax%
1041           \edef\notation@curr@prec{\infprec}%
1042         \else%
1043           \def\notation@curr@prec{0}%
1044         \fi%
1045       \else%
1046         \edef\notation@curr@prec{\notation@curr@prec}%
1047         \def\notation@curr@prec{}%
1048       \fi%
1049     }%

```



```

1050 % arguments
1051 \def\notation@curr@extargs{}
1052 \def\notation@nextarg@index{1}%
1053 \notation@do@args%
1054 }
1055
1056 % parses additional notation components for (associative) arguments
1057 \def\notation@do@args{%
1058   \def\notation@nextarg@temp{}%
1059   \ifx\notation@curr@todo@args\@empty%
1060     \notation@after%
1061   \else%
1062     % argument precedence
1063     \IfSubStr\notation@curr@prec{s}{x}{%
1064       \StrCut\notation@curr@prec{s}{x}\notation@curr@argprec\notation@curr@prec%
1065     }{%
1066       \edef\notation@curr@argprec{\notation@curr@prec}%
1067       \def\notation@curr@prec{}%
1068     }%
1069     \ifx\notation@curr@argprec\@empty%
1070       \let\notation@curr@argprec\notation@curr@prec%
1071     \fi%
1072     \StrChar\notation@curr@todo@args1[\notation@argchar]%
1073     \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1074     \expandafter\ifx\notation@argchar i%
1075       % normal argument
1076       \edef\notation@nextarg@temp{\noexpand\notation@argprec{\notation@curr@argprec}{#####\
1077       \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }
1078       \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
1079       \expandafter{\notation@nextarg@temp}%
1080       \expandafter\expandafter\expandafter\notation@do@args%
1081     \else%
1082       % associative argument
1083       \expandafter\expandafter\expandafter\notation@parse@assocarg%
1084     \fi%
1085   \fi%
1086 }
1087
1088 \def\notation@parse@assocarg#1{%
1089   \edef\notation@nextarg@temp{\noexpand\notation@argprec{\notation@curr@argprec}{\noexpand\not
1090   \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }%
1091   \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
1092   \expandafter{\notation@nextarg@temp}%
1093   \notation@do@args%
1094 }
1095
1096 \protected\def\safe@newcommand#1{%
1097   \ifdefined#1\expandafter\renewcommand\else\expandafter\newcommand\fi#1%
1098 }
1099

```

```

1100 % finally creates the actual macros
1101 \def\notation@after{
1102   \let\ex\expandafter%
1103   \ex\ex\ex\def\ex\ex\ex\notation@temp@notation\ex\ex\ex%
1104     {\ex\notation@temp@notation\notation@curr@extargs}%
1105   \edef\notation@temp@notation{\noexpand\notation@symprec{\notation@curr@prec}{\ex\unexpanded\ex%
1106     \def\notation@temp@fragment{}}%
1107   \ifx\notation@curr@arity\@empty\else%
1108     \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1109   \fi%
1110   \ifx\notation@curr@lang\@empty\else%
1111     \ifx\notation@temp@fragment\@empty%
1112       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1113     \else%
1114       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1115     \fi%
1116   \fi%
1117   \ifx\notation@curr@variant\@empty\else%
1118     \ifx\notation@temp@fragment\@empty%
1119       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1120     \else%
1121       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1122     \fi%
1123   \fi%
1124   \if@inverbalization\@inverbalizationfalse\verbalization@final%
1125   \else\notation@final\fi%
1126   \parsemodule@maybesetcodes%
1127 }
1128
1129 \def\notation@final{%
1130   \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1131   \ifcsvoid{\notation@csname}{%
1132     \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1133       \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@temp@arity\ex]%
1134     \ex{\notation@temp@notation}%
1135     \edef\symdecl@temps{%
1136       \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@temp@ari
1137     }%
1138     \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1139     \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1140   }{%
1141     \PackageWarning{stex}{notation already defined: \notation@csname}{%
1142       Choose a different set of notation options (variant,lang,arity)%
1143     }%
1144   }%
1145 }
1146
1147 \def\verbalization@final{%
1148   \edef\notation@csname{\notation@uri\@Fragment verb\@Fragment\notation@temp@fragment}%
1149   \ifcsvoid{\notation@csname}{%

```

```

1150 \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1151 \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@temp@arity\ex]%
1152 \ex{\notation@temp@notation}%
1153 \edef\symdecl@temps{%
1154 \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@temp@ari
1155 }%
1156 \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1157 \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1158 }-%
1159 \PackageWarning{stex}{verbalization already defined: \notation@csname}%
1160 Choose a different set of verbalization options (variant,lang,arity)%
1161 }%
1162 }%
1163 }
1164
1165 % parses optional parameters
1166 \def\notation@parse@params#1#2{%
1167 \def\notation@curr@prec{%
1168 \def\notation@curr@args{%
1169 \def\notation@curr@variant{%
1170 \def\notation@curr@arity{%
1171 \def\notation@curr@provided@arity#{#2}
1172 \def\notation@curr@lang{%
1173 \def\notation@options@temp#{#1}
1174 \notation@parse@params%
1175 \ifx\notation@curr@args\@empty%
1176 \ifx\notation@curr@provided@arity\@empty%
1177 \notation@num@to@ia\notation@curr@arity%
1178 \else%
1179 \notation@num@to@ia\notation@curr@provided@arity%
1180 \fi%
1181 \fi%
1182 }
1183 \def\notation@parse@params@{%
1184 \IfSubStr\notation@options@temp,{%
1185 \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
1186 \notation@parse@param%
1187 \notation@parse@params@%
1188 }{\ifx\notation@options@temp\@empty\else%
1189 \let\notation@option@temp\notation@options@temp%
1190 \notation@parse@param%
1191 \fi}%
1192 }
1193
1194 %parses an individual optional argument/key-value-pair
1195 \def\notation@parse@param{%
1196 \trimstring\notation@option@temp%
1197 \ifx\notation@option@temp\@empty\else%
1198 \IfSubStr\notation@option@temp=%{%
1199 \StrCut\notation@option@temp=\notation@key\notation@value%

```

```

1200 \trimstring\notation@key%
1201 \trimstring\notation@value%
1202 \IfStrEq\notation@key{prec}{%
1203 \edef\notation@curr@prec{\notation@value}%
1204 }{%
1205 \IfStrEq\notation@key{args}{%
1206 \edef\notation@curr@args{\notation@value}%
1207 }{%
1208 \IfStrEq\notation@key{lang}{%
1209 \edef\notation@curr@lang{\notation@value}%
1210 }{%
1211 \IfStrEq\notation@key{variant}{%
1212 \edef\notation@curr@variant{\notation@value}%
1213 }{%
1214 \IfStrEq\notation@key{arity}{%
1215 \edef\notation@curr@arity{\notation@value}%
1216 }{%
1217 }}}}%
1218 }{%
1219 \edef\notation@curr@variant{\notation@option@temp}%
1220 }%
1221 \fi%
1222 }
1223
1224 % converts an integer to a string of 'i's, e.g. 3 => iii,
1225 % and stores the result in \notation@curr@args
1226 \def\notation@num@to@ia#1{%
1227 \IfInteger{#1}{
1228 \notation@num@to@ia@#1%
1229 }{%
1230 %
1231 }%
1232 }
1233 \def\notation@num@to@ia@#1{%
1234 \ifnum#1>0%
1235 \edef\notation@curr@args{\notation@curr@args i}%
1236 \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
1237 \fi%
1238 }

```

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

```

1239 \def\notation@assoc#1#2{% function, argv
1240 \let\@tmpop=\relax% do not print the function the first time round
1241 \@for\@I:=#2\do{\@tmpop% print the function
1242 % write the i-th argument with locally updated precedence
1243 \@I%
1244 \def\@tmpop{#1}%
1245 }%
1246 }%

```

```

1247
1248 \def\notation@lparen{()
1249 \def\notation@rparen{)}
1250 \def\infprec{1000000}
1251 \def\neginfprec{-\infprec}
1252
1253 \newcount\notation@downprec
1254 \notation@downprec=\neginfprec
1255
1256 % patching displaymode
1257 \newif\if@displaymode\@displaymodefalse
1258 \expandafter\everydisplay\expandafter{\the\everydisplay\@displaymodetrue}
1259 \let\old@displaystyle\displaystyle
1260 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1261
1262 \def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1263   \def\notation@innertmp{#1}%
1264   \let\ex\expandafter%
1265   \if@displaymode%
1266     \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1267     \ex\notation@resetbrackets\ex\notation@innertmp%
1268     \ex\right\notation@rparen%
1269   \else%
1270     \ex\ex\ex\notation@lparen%
1271     \ex\notation@resetbrackets\ex\notation@innertmp%
1272     \notation@rparen%
1273   \fi%
1274 }
1275
1276 \def\withbrackets#1#2#3{%
1277   \edef\notation@lparen{#1}%
1278   \edef\notation@rparen{#2}%
1279   #3%
1280   \notation@resetbrackets%
1281 }
1282
1283 \def\notation@resetbrackets{%
1284   \def\notation@lparen{()%
1285   \def\notation@rparen{)%}
1286 }
1287
1288 \def\notation@symprec#1#2{%
1289   \ifnum#1>\notation@downprec\relax%
1290     \notation@resetbrackets#2%
1291   \else%
1292     \ifnum\notation@downprec=\infprec\relax%
1293       \notation@resetbrackets#2%
1294     \else
1295       \if@inarray@
1296         \notation@resetbrackets#2

```

```

1297         \else\dobrackets{#2}\fi%
1298     \fi\fi%
1299 }
1300
1301 \newif\if@inpparray@\@inpparray@false
1302
1303 \def\notation@argprec#1#2{%
1304     \def\notation@innertmp{#2}
1305     \edef\notation@downprec@temp{\number#1}%
1306     \notation@downprec=\expandafter\notation@downprec@temp%
1307     \expandafter\relax\expandafter\notation@innertmp%
1308     \expandafter\notation@downprec\expandafter=\number\notation@downprec\relax%
1309 }

```

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

```

1310 \protected\def\@invoke@symbol#1{%
1311     \def\@invoke@symbol@first{#1}%
1312     \symbol@args%
1313 }

```

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

```

1314 \newcommand\symbol@args[1][]{%
1315     \notation@parse@params{#1}{}%
1316     \def\notation@temp@fragment{}%
1317     \ifx\notation@curr@arity\@empty\else%
1318         \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1319     \fi%
1320     \ifx\notation@curr@lang\@empty\else%
1321         \ifx\notation@temp@fragment\@empty%
1322             \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1323         \else%
1324             \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1325         \fi%
1326     \fi%
1327     \ifx\notation@curr@variant\@empty\else%
1328         \ifx\notation@temp@fragment\@empty%
1329             \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1330         \else%
1331             \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1332         \fi%
1333     \fi%
1334     %
1335     \ifmmode\def\invoke@symbol@next{\@invoke@symbol@math\@invoke@symbol@first\notation@temp@fragm
1336     \else\def\invoke@symbol@next{\@invoke@symbol@text\@invoke@symbol@first\notation@temp@fragment
1337     \invoke@symbol@next%
1338 }

```

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

a LaTeXML binding:

```
1339 \def\@invoke@symbol@math#1#2{%
1340   \csname #1\@Fragment#2\endcsname%
1341 }
```

TODO:

```
1342 \def\@invoke@symbol@text#1#2{%
1343   \@termref{#1}\csname #1\@Fragment verb\@Fragment#2\endcsname}%
1344 }
```

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

```
1345 \def\setstexlang#1{%
1346   \def\stex@lang{#1}%
1347 }%
1348 \setstexlang{en}
1349 \def\setstexvariant#1#2{%
1350   % TODO
1351 }
1352 \def\setstexvariants#1{%
1353   \def\stex@variants{#1}%
1354 }
```

**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 }`

`$\times {\frac {\mathrm {a} }{\mathrm {b} }},\mathrm {+} {\frac {\mathrm {a} }{\mathrm {b} }}{\frac {\mathrm {a} }{\mathrm {b} }},\times {\mathrm {c} },\mathrm {+} {\mathrm {d} },\mathrm {e} }$:`  

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

`\[\times {\frac {\mathrm {a} }{\mathrm {b} }},\mathrm {+} {\frac {\mathrm {a} }{\mathrm {b} }}{\frac {\mathrm {a} }{\mathrm {b} }},\times {\mathrm {c} },\mathrm {+} {\mathrm {d} },\mathrm {e} }\]`:

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

`\symdecl {foo!bar}`

`\foo !bar: foo bar`

`\symdecl [verbalization={finite group}]{finitigroup}`

`\verbalization [variant=oforder]{finitigroup}[1]{finite group of order ##1}`

`\finitigroup [oforder]{n$}: finite group of order n`

### 3.6 Term References

`\ifhref`

```
1355 \newif\ifhref\hreffalse%
1356 \AtBeginDocument{%
1357   \ifpackageloaded{hyperref}{%
1358     \hreftrue%
1359   }{%
1360     \hreffalse%
1361   }%
1362 }
```

`\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.

```
1363 \newbox\stex@targetbox
1364 \def\termref@maketarget#1#2{%
1365   % #1: symbol URI
1366   % #2: text
1367   \message{^^JHere: #1 <> #2^^J}%
1368   \ifhref\if@smsmode\else%
1369     \hypertarget{sref@#1@target}{#2}%
1370   \fi\fi%
1371   \message{^^JHere!^^J}%
1372   \expandafter\edef\csname sref@#1\endcsname##1{%
1373     \ifhref\if@smsmode\else\noexpand\hyperlink{sref@#1@target}{##1}\fi\fi%
1374   }%
1375 }
```

`\@termref`

```
1376 \def\@termref#1#2{%
```



```

1377 % #1: symbol URI
1378 % #2: text
1379 \ifcsvoid{#1}{%
1380   \StrCut[2]{#1}\@QuestionMark\termref@mod\termref@name%
1381   \ifcsvoid{\termref@mod}{%
1382     \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
1383   }{%
1384     \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
1385       contains no symbol with name \termref@name.%
1386     }{%
1387   }%
1388 }{%
1389   \ifcsvoid{sref@#1}{%
1390     #2% TODO: No reference point exists!
1391   }{%
1392     \csname sref@#1\endcsname{#2}%
1393   }%
1394 }%
1395 }

\tref

1396
1397 \def\@capitalize#1{\uppercase{#1}}%
1398 \newrobustcmd\capitalize[1]{\expandafter\@capitalize #1}%
1399
1400 \newcommand\tref[2][]{%
1401   \edef\tref@name{#1}%
1402   \ifx\tref@name\empty
1403     \symdecl@constructname{#2}%
1404     \edef\tref@name{\symdecl@name}%
1405   \else%
1406     \edef\symdecl@verbalization{#2}%
1407   \fi%
1408   \expandafter\modules@getURIfromName\expandafter{\tref@name}%
1409   \expandafter\@termref\expandafter{\notation@uri}\symdecl@verbalization}%
1410 }
1411 \def\trefs#1{%
1412   \modules@getURIfromName{#1}%
1413   \expandafter\@termref\expandafter{\notation@uri}\csname\notation@uri\@Fragment verb\@Fragment
1414 }
1415 \def\Tref#1{%
1416   \modules@getURIfromName{#1}%
1417   \expandafter\@termref\expandafter{\notation@uri}\expandafter\capitalize\csname\notation@uri\
1418 }
1419 \def\Trefs#1{%
1420   \modules@getURIfromName{#1}%
1421   \expandafter\@termref\expandafter{\notation@uri}\expandafter\capitalize\csname\notation@uri\
1422 }

```

**Test:**

foo bar  
foo-bar  
finite group

```
\defi
1423 \addmetakey{defi}{name}
1424 \def\@definiendum#1#2{%
1425   \parsemodule@maybesetcodes%
1426   \message{^^JHere: #1 | #2^^J}%
1427   \termref@maketarget{#1}{#2}\termref@maketarget{#1}{\defemph{#2}}%
1428 }
1429
1430 \newcommand\defi[2][{}]{%
1431   \metasetkeys{defi}{#1}%
1432   \ifx\defi@name\@empty%
1433     \symdecl@constructname{#2}%
1434     \let\defi@name\symdecl@name%
1435     \let\defi@verbalization\symdecl@verbalization%
1436   \else%
1437     \edef\defi@verbalization{#2}%
1438   \fi%
1439   \ifcvoid{\module@uri\@QuestionMark\defi@name}{%
1440     \symdecl\defi@name%
1441   }{\edef\symdecl@uri{\module@uri\@QuestionMark\defi@name}}%
1442   \@definiendum\symdecl@uri\defi@verbalization%
1443 }
1444 \def\Defi#1{%
1445   \symdecl{#1}%
1446   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization}%
1447 }
1448 \def\defis#1{%
1449   \symdecl{#1}%
1450   \@definiendum\symdecl@uri{\symdecl@verbalization s}%
1451 }
1452 \def\Defis#1{%
1453   \symdecl{#1}%
1454   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization s}%
1455 }
```

**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
```

```

1456 \newif\ifhref\hreffalse%
1457 \AtBeginDocument{%
1458   \ifpackageloaded{hyperref}{%
1459     \hreftrue%
1460   }{%
1461     \hreffalse%
1462   }%
1463 }%
1464 \newcommand\sref@href@ifh[2]{%
1465   \ifhref%
1466     \href{#1}{#2}%
1467   \else%
1468     #2%
1469   \fi%
1470 }%
1471 \newcommand\sref@hlink@ifh[2]{%
1472   \ifhref%
1473     \hyperlink{#1}{#2}%
1474   \else%
1475     #2%
1476   \fi%
1477 }%
1478 \newcommand\sref@target@ifh[2]{%
1479   \ifhref%
1480     \hypertarget{#1}{#2}%
1481   \else%
1482     #2%
1483   \fi%
1484 }%

```

Then we provide some macros for  $\text{\TeX}$ -specific crossreferencing

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

```

1485 \def\sref@target{%
1486   \ifx\sref@id\@empty%
1487     \relax%
1488   \else%
1489     \edef\@target{\sref@ifcsundef{sref@part}}{\sref@part @}\sref@id @target}%
1490     \sref@target@ifh\@target{}%
1491   \fi%
1492 }%

```

**\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.

```

1493 \addmetakey{srefaddidkey}{prefix}
1494 \newcommand\srefaddidkey[2][]{%
1495   \metasetkeys{srefaddidkey}{#1}%
1496   \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
1497   \metakeys@ext@clear@keys{#2}{id}{}%
1498   \metakeys@ext@showkeys{#2}{id}%
1499   \define@key{#2}{id}{%
1500     \edef\sref@id{\srefaddidkey@prefix ##1}%
1501     %\expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
1502     \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1503   }%
1504 }%

\@sref@def This macro stores the value of its last argument in a custom macro for reference.
1505 \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.
1506 \ifextrefs%
1507   \newwrite\refs@file%
1508 \else%
1509   \def\refs@file{\@auxout}%
1510 \fi%

\sref@def This macro writes an \@sref@def command to the current aux file and also exe-
cutes it.
1511 \newcommand\sref@def[3]{%
1512   \protected@write\refs@file{}{\string\sref@def{#1}{#2}{#3}}%
1513 }%

\sref@label The \sref@label macro writes a label definition to the auxfile.
1514 \newcommand\sref@label[2]{%
1515   \sref@def{\ifcsundef{sref@part}{}\sref@part @}{#2}{page}{\thepage}%
1516   \sref@def{\ifcsundef{sref@part}{}\sref@part @}{#2}{label}{#1}%
1517 }%

\sreflabel The \sreflabel macro is a semantic version of \label, it combines the catego-
rization given in the first argument with LATEX's \@currentlabel.
1518 \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.
1519 \def\sref@id{} % make sure that defined
1520 \newcommand\sref@label@id[1]{%
1521   \ifx\sref@id\@empty%
1522     \relax%
1523   \else%
1524     \sref@label{#1}{\sref@id}%
1525   \fi%
1526 }%

```

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

```

1527 \newcommand\sref@label@id@arg[2]{%
1528   \def\@@id{#2}
1529   \ifx\@@id\@empty%
1530     \relax%
1531   \else%
1532     \sref@label{#1}{\@@id}%
1533   \fi%
1534 }%
```

### 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`.

```

1535 \newenvironment{modsig}[2][\def\@test{#1}%
1536 \ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1537 \expandafter\gdef\csname mod@#2\multiling\endcsname{true}%
1538 \ignorespacesandpars}
1539 {\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`.

```

1540 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1541 \newrobustcmd\@gimport@star[2][\def\@test{#1}%
1542 \edef\mh@repos{\mh@currentrepos}%
1543 \ifx\@test\@empty%
1544   \importmhmodule[conservative,mhrepos=\mh@repos,path=#2]{#2}%
1545 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1546 \setcurrentreposinfo{\mh@repos}%
1547 \ignorespacesandpars\parsemodule@maybesetcodes}
1548 \newrobustcmd\@gimport@nostar[2][\def\@test{#1}%
1549 \edef\mh@repos{\mh@currentrepos}%
```

```

1550 \ifx\@test\@empty%
1551 \importmhmodule[mhrepos=\mh@repos,path=#2]{#2}%
1552 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1553 \setcurrentreposinfo{\mh@repos}%
1554 \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@infile` and `\mh@libfile` and restore them at the end.

```

1555 \def\modules@@first#1/#2;{#1}
1556 \newcommand\libinput[1]{%
1557 \ifcsvoid{mh@currentrepos}{%
1558   \PackageError{stex}{current MathHub repository not found}{}}%
1559   {}
1560 \edef\@mh@group{\expandafter\modules@@first\mh@currentrepos;}
1561 \let\orig@infile\mh@infile\let\orig@libfile\mh@libfile
1562 \def\mh@infile{\MathHub{\@mh@group/meta-inf/lib/#1}}
1563 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1564 \IfFileExists\mh@infile{\stexinput\mh@infile}{}%
1565 \IfFileExists\mh@infile{\IfFileExists\mh@libfile}{%
1566   {\PackageError{stex}
1567     {Library file missing; cannot input #1.tex\MessageBreak%
1568       Both \mh@libfile.tex\MessageBreak and \mh@infile.tex\MessageBreak%
1569       do not exist}%
1570     {Check whether the file name is correct}}}%
1571 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1572 \let\mh@infile\orig@infile\let\mh@libfile\orig@libfile}

```

### 3.11 omdoc/omgroup

```

1573 \newcount\section@level
1574
1575 \section@level=2
1576 \ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{%
1577 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{%
1578 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{%
1579 \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>*.

```

1580 \newcommand\omgroup@nonum[2]{%
1581 \ifx\hyper@anchor\@undefined\else\phantomsection\fi%
1582 \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 rdfmeta package has been loaded. In the end we call \sref@label@id to enable crossreferencing.

```

1583 \newcommand\omgroup@num[2]{%
1584 \edef\@@ID{\sref@id}
1585 \ifx\omgroup@short\empty% no short title
1586 \@nameuse{#1}{#2}%
1587 \else% we have a short title
1588 \@ifundefined{rdfmeta@sectioning}%
1589   {\@nameuse{#1}[\omgroup@short]{#2}}%
1590   {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
1591 \fi%
1592 \sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@@ID}

```

omgroup

```

1593 \def\@true{true}
1594 \def\@false{false}
1595 \srefaddidkey{omgroup}
1596 \addmetakey{omgroup}{date}
1597 \addmetakey{omgroup}{creators}
1598 \addmetakey{omgroup}{contributors}
1599 \addmetakey{omgroup}{srccite}
1600 \addmetakey{omgroup}{type}
1601 \addmetakey*{omgroup}{short}
1602 \addmetakey*{omgroup}{display}
1603 \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.

```

1604 \newif\if@mainmatter\@mainmattertrue
1605 \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.

```

1606 \addmetakey{omdoc@sect}{name}
1607 \addmetakey[false]{omdoc@sect}{clear}[true]
1608 \addmetakey{omdoc@sect}{ref}
1609 \addmetakey[false]{omdoc@sect}{num}[true]
1610 \newcommand\omdoc@sectioning[3][]{\metasetkeys{omdoc@sect}{#1}%
1611 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
1612 \if@mainmatter% numbering not overridden by frontmatter, etc.
1613 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
1614 \def\current@section@level{\omdoc@sect@name}%
1615 \else\omgroup@nonum{#2}{#3}%
1616 \fi}% if@mainmatter

```

and another one, if redefines the \addtocontentsline macro of L<sup>A</sup>T<sub>E</sub>X to import the respective macros. It takes as an argument a list of module names.

```

1617 \newcommand\omgroup@redefine@addtocontents[1]{%

```

```

1618 %\edef\@import{#1}%
1619 %\@for\@I:=\@import\do{%
1620 %\edef\@path{\csname module@\@I @path\endcsname}%
1621 %\@ifundefined{tf@toc}\relax%
1622 %    {\protected@write\tf@toc}{\string\@requiremodules{\@path}}}%
1623 %\ifx\hyper@anchor\@undefined% hyperref.sty loaded?
1624 %\def\addcontentsline##1##2##3{%
1625 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}%
1626 %\else% hyperref.sty not loaded
1627 %\def\addcontentsline##1##2##3{%
1628 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}{\@cc}
1629 %\fi
1630 }% hypreref.sty loaded?

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

1631 \newcount\omgroup@level
1632 \newenvironment{omgroup}[2][ ]% keys, title
1633 {\metasetkeys{omgroup}{#1}\sref@target%
1634 \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.

1635 \ifx\omgroup@loadmodules\@true%
1636 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
1637 {\@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.

1638 \advance\section@level by 1\relax%
1639 \ifcase\section@level%
1640 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
1641 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
1642 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
1643 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
1644 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%
1645 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{#2}%
1646 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{#2}%
1647 \fi% \ifcase
1648 \at@beginomgroup[#1]\section@level{#2}}% for customization
1649 {\advance\section@level by -1\advance\omgroup@level by -1}

    and finally, we localize the sections

1650 \newcommand\omdoc@part@kw{Part}
1651 \newcommand\omdoc@chapter@kw{Chapter}
1652 \newcommand\omdoc@section@kw{Section}
1653 \newcommand\omdoc@subsection@kw{Subsection}

```



```

1654 \newcommand\omdoc@subsubsection@kw{Subsubsection}
1655 \newcommand\omdoc@paragraph@kw{paragraph}
1656 \newcommand\omdoc@subparagraph@kw{subparagraph}

\setSGvar set a global variable
1657 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}

\useSGvar use a global variable
1658 \newrobustcmd\useSGvar[1]{%
1659   \ifundefined{sTeX@Gvar@#1}
1660   {\PackageError{omdoc}
1661     {The sTeX Global variable #1 is undefined}
1662     {set it with \protect\setSGvar}}
1663   \@nameuse{sTeX@Gvar@#1}}

blindomgroup
1664 \newcommand\at@begin@blindomgroup[1]{%
1665   \newenvironment{blindomgroup}
1666   {\advance\section@level by 1\at@begin@blindomgroup\setion@level}
1667   {\advance\section@level by -1}}

```

## 3.12 omtext

### 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).

```

1668 \srefaddidkey{omtext}
1669 \addmetakey[]{omtext}{functions}
1670 \addmetakey*{omtext}{display}
1671 \addmetakey{omtext}{for}
1672 \addmetakey{omtext}{from}
1673 \addmetakey{omtext}{type}
1674 \addmetakey*{omtext}{title}
1675 \addmetakey*{omtext}{start}
1676 \addmetakey{omtext}{theory}
1677 \addmetakey{omtext}{continues}
1678 \addmetakey{omtext}{verbalizes}
1679 \addmetakey{omtext}{subject}

\st@flow We define this macro, so that we can test whether the display key has the value
flow
1680 \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.

```

1681 \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.
1682 \def\omtext@pre@skip{\smallskip}
1683 \def\omtext@post@skip{}
1684 \newenvironment{omtext}[1][\@in@omtexttrue%
1685   \bgroup\metasetkeys{omtext}{#1}\sref@label{id{this paragraph}}%
1686   \def\lec##1{\@lec{##1}}}%
1687   \omtext@pre@skip\par\noindent%
1688   \ifx\omtext@title\@empty%
1689     \ifx\omtext@start\@empty\else%
1690       \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
1691     \fi% end omtext@start empty
1692   \else\stDMemph{\omtext@title}:\enspace%
1693     \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
1694   \fi% end omtext@title empty
1695   \ignorespacesandpars}
1696 {\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

```

1697 \srefaddidkey{phrase}
1698 \addmetakey{phrase}{style}
1699 \addmetakey{phrase}{class}
1700 \addmetakey{phrase}{index}
1701 \addmetakey{phrase}{verbalizes}
1702 \addmetakey{phrase}{type}
1703 \addmetakey{phrase}{only}
1704 \newcommand\phrase[2][\metasetkeys{phrase}{#1}%
1705 \ifx\phrase@only\@empty\only<\phrase@only>{#2}\else #2\fi}

```

`\coref*`

```

1706 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}
1707 \newcommand\corefs[2]{#1\textsubscript{#2}}
1708 \newcommand\coreft[2]{#1\textsuperscript{#2}}

```

`\n*lex`

```

1709 \newcommand\nlex[1]{\green{\sl{#1}}}
1710 \newcommand\nlcex[1]{*\green{\sl{#1}}}

```

`sinlinequote`

```

1711 \def\@sinlinequote#1{‘‘{\sl{#1}}}’’}
1712 \def\@sinlinequote#1#2{\@sinlinequote{#2}~#1}
1713 \newcommand\sinlinequote[2][
1714 {\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.

```
1715 \newcommand\vdec[2] [] {#2}
1716 \newcommand\vrest[2] [] {#2}
1717 \newcommand\vcond[2] [] {#2}
```

```
EdN:1 \strucdec 1
1718 \newcommand\strucdec[2] [] {#2}
```

```
EdN:2 \impdec 2
1719 \newcommand\impdec[2] [] {#2}
```

### 3.12.4 Block-Level Markup

```
sblockquote
1720 \def\begin@sblockquote{\begin{quote}}\s1}
1721 \def\end@sblockquote{\end{quote}}
1722 \def\begin@@sblockquote#1{\begin@sblockquote}
1723 \def\end@@sblockquote#1{\def\@lec#1{\textrm{##1}}\@lec{#1}\end@sblockquote}
1724 \newenvironment{sblockquote}[1] []
1725 {\def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
1726 {\ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}

sboxquote
1727 \newenvironment{sboxquote}[1] []
1728 {\def\@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
1729 {\@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.

```
1730 \providecommand{\@lec}[1]{(##1)}
1731 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@lec{#1}}
1732 \def\lec#1{\@lec{#1}\par}
```

---

<sup>1</sup>EDNOTE: document above

<sup>2</sup>EDNOTE: document above

### 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.

```

1733 \addmetakey{omdoc@index}{at}
1734 \addmetakey[false]{omdoc@index}{loadmodules}[true]
1735 \newcommand\omdoc@indexi[2][\ifindex%
1736 \metasetkeys{omdoc@index}{#1}%
1737 \@bsphack\begingroup\@sanitize%
1738 \protected@write\@indexfile{}\string\indexentry%
1739 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1740 \ifx\omdoc@index@loadmodules\@true%
1741 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
1742 \else #2\fi% loadmodules
1743 }\the page}}%
1744 \endgroup\@esphack\fi}%ifindex
1745 \newcommand\omdoc@indexii[3][\ifindex%
1746 \metasetkeys{omdoc@index}{#1}%
1747 \@bsphack\begingroup\@sanitize%
1748 \protected@write\@indexfile{}\string\indexentry%
1749 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1750 \ifx\omdoc@index@loadmodules\@true%
1751 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1752 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1753 \else #2!#3\fi% loadmodules
1754 }\the page}}%
1755 \endgroup\@esphack\fi}%ifindex
1756 \newcommand\omdoc@indexiii[4][\ifindex%
1757 \metasetkeys{omdoc@index}{#1}%
1758 \@bsphack\begingroup\@sanitize%
1759 \protected@write\@indexfile{}\string\indexentry%
1760 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1761 \ifx\omdoc@index@loadmodules\@true%
1762 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1763 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1764 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1765 \else #2!#3!#4\fi% loadmodules
1766 }\the page}}%
1767 \endgroup\@esphack\fi}%ifindex
1768 \newcommand\omdoc@indexiv[5][\ifindex%
1769 \metasetkeys{omdoc@index}{#1}%
1770 \@bsphack\begingroup\@sanitize%
1771 \protected@write\@indexfile{}\string\indexentry%
1772 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%

```

```

1773 \ifx\omdoc@index@loadmodules \@true%
1774 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1775 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1776 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1777 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
1778 \else #2!#3!#4!#5\fi% loadmodules
1779 }\the page}}%
1780 \endgroup\@esphack\fi}%ifindex

```

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

`\*indi*`

```

1781 \newcommand\aindi[3] []{{#2}\omdoc@indexi[#1]{#3}}
1782 \newcommand\indi[2] []{{#2}\omdoc@indexi[#1]{#2}}
1783 \newcommand\indis[2] []{{#2}\omdoc@indexi[#1]{#2s}}
1784 \newcommand\Indi[2] []{{\captitalize{#2}}\omdoc@indexi[#1]{#2}}
1785 \newcommand\Indis[2] []{{\capitalize{#2}}\omdoc@indexi[#1]{#2s}}
1786
1787 \newcommand\@indii[3] []{\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
1788 \newcommand\aindii[4] []{{#2}\@indii[#1]{#3}{#4}}
1789 \newcommand\indii[3] []{{#2 #3}\@indii[#1]{#2}{#3}}
1790 \newcommand\indiis[3] []{{#2 #3s}\@indii[#1]{#2}{#3}}
1791 \newcommand\Indii[3] []{{\captitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1792 \newcommand\Indiis[3] []{{\capitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1793
1794 \newcommand\@indiii[4] []{\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexiii[#1]{#3}{#2 (#4)}}
1795 \newcommand\aindiii[5] []{{#2}\@indiii[#1]{#3}{#4}{#5}}
1796 \newcommand\indiii[4] []{{#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
1797 \newcommand\indiis[4] []{{#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}
1798 \newcommand\Indiii[4] []{{\captitalize{#2 #3 #4}}\@indiii[#1]{#2}{#3}{#4}}
1799 \newcommand\Indiis[4] []{{\capitalize{#2 #3 #4s}}\@indiii[#1]{#2}{#3}{#4}}
1800
1801 \newcommand\@indiv[5] []{\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
1802 \newcommand\aindiv[6] []{{#2}\@indiv[#1]{#3}{#4}{#5}{#6}}
1803 \newcommand\indiv[5] []{{#2 #3 #4 #5}\@indiv[#1]{#2}{#3}{#4}{#5}}
1804 \newcommand\indivs[5] []{{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
1805 \newcommand\Indiv[5] []{{\capitalize{#2 #3 #4 #5s}}\@indiv[#1]{#2}{#3}{#4}{#5}}
1806 \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<sup>A</sup>T<sub>E</sub>X<sub>M</sub>L.

```

1807 \newcommand\hateq{\ensuremath{\widehat{=}}\xspace}
1808 \newcommand\hatequiv{\ensuremath{\widehat{\equiv}}\xspace}
1809 \@ifundefined{ergo}%
1810 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1811 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1812 \newcommand{\reflectsquig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%

```

```

1813 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
1814 \newcommand\notergo{\ensuremath{\not\leadsto}}
1815 \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*`

```

1816 \newcommand\indextoo[2] [] {\indi[#1]{#2}}%
1817 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}%
1818 \newcommand\indexalt[2] [] {\aindi[#1]{#2}}%
1819 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead}%
1820 \newcommand\twintoo[3] [] {\indii[#1]{#2}{#3}}%
1821 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}%
1822 \newcommand\twinalt[3] [] {\aindii[#1]{#2}{#3}}%
1823 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead}%
1824 \newcommand\atwintoo[4] [] {\indiii[#1]{#2}{#3}{#4}}%
1825 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead}%
1826 \newcommand\atwinalt[4] [] {\aindii[#1]{#2}{#3}{#4}}%
1827 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instead}%
1828 \</package>

```

`\my*graphics`

```

1829 \newcommand\mygraphics[2] [] {\includegraphics[#1]{#2}}%
1830 \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics instead}%
1831 \newcommand\mycgraphics[2] [] {\begin{center}\mygraphics[#1]{#2}\end{center}}%
1832 \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics instead}%
1833 \newcommand\mybgraphics[2] [] {\fbox{\mygraphics[#1]{#2}}}%
1834 \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics instead}%
1835 \newcommand\mycbgraphics[2] [] {\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}%
1836 \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphics instead}%

```

## 4 Things to deprecate

Module options:

```

1837 \addmetakey*{module}{id} % TODO: deprecate properly
1838 \addmetakey*{module}{load}
1839 \addmetakey*{module}{path}
1840 \addmetakey*{module}{dir}
1841 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
1842 \addmetakey*{module}{noalign}[true]
1843
1844 \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<sup>A</sup>T<sub>E</sub>X part.

```

1845 %\srefaddidkey{symdef}% what does this do?
1846 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
1847 \define@key{symdef}{noverb}[all]{}%
1848 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
1849 \define@key{symdef}{specializes}{}%
1850 \addmetakey*{symdef}{noalign}[true]
1851 \define@key{symdef}{primary}[true]{}%
1852 \define@key{symdef}{assocarg}{}%
1853 \define@key{symdef}{bvars}{}%
1854 \define@key{symdef}{bargs}{}%
1855 \addmetakey{symdef}{lang}%
1856 \addmetakey{symdef}{prec}%
1857 \addmetakey{symdef}{arity}%
1858 \addmetakey{symdef}{variant}%
1859 \addmetakey{symdef}{ns}%
1860 \addmetakey{symdef}{args}%
1861 \addmetakey{symdef}{name}%
1862 \addmetakey*{symdef}{title}%
1863 \addmetakey*{symdef}{description}%
1864 \addmetakey{symdef}{subject}%
1865 \addmetakey*{symdef}{display}%
1866 \addmetakey*{symdef}{gfc}%

```

3

EdN:3

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

```

1867 \def\symdef{\@ifnextchar[{\@symdef}{\@symdef[]}}%
1868 \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.

```

1869 \def\@@symdef[#1]#2[#3]{%
1870   \@insymdef@true%
1871   \metasetkeys{symdef}{#1}%
1872   \edef\symdef@tmp@optpars{\ifcsvoid{symdef@name}{[]}{[name=\symdef@name]}}%
1873   \expandafter\symdecl\symdef@tmp@optpars{#2}%
1874   \@insymdef@false%
1875   \notation[#1]{#2}[#3]%
1876 }% mod@show
1877 \def\symdef@type{Symbol}%
1878 \providecommand{\stDMemph}[1]{\textbf{#1}}

```

---

<sup>3</sup>EDNOTE: MK@MK: we need to document the binder keys above.

`\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`.

```

1879 \def\symvariant#1{%
1880   \ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}%
1881   }%
1882 \def\@symvariant#1[#2]#3#4{%
1883   \notation[#3]{#1}[#2]{#4}%
1884 \ignorespacesandpars}%

```

`\abbrdef` The `\abbrdef` macro is a variant of `\symdef` that does the same on the L<sup>A</sup>T<sub>E</sub>X level.

```

1885 \let\abbrdef\symdef%

```

`\@sym*` has a starred form for primary symbols. The key/value interface has no effect on the L<sup>A</sup>T<sub>E</sub>X side. We read the to check whether only allowed ones are used.

```

1886 \newif\if@importing\@importingfalse
1887 \define@key{symi}{noverb}[all]{}%
1888 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
1889 \define@key{symi}{specializes}{}%
1890 \define@key{symi}{gfc}{}%
1891 \define@key{symi}{noalign}[true]{}%
1892 \newcommand\symi{\@ifstar\@symi@star\@symi}
1893 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
1894   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces}
1895 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
1896   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignorespaces}
1897 \newcommand\symii{\@ifstar\@symii@star\@symii}
1898 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
1899   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespaces}
1900 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
1901   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\ignorespaces}
1902 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
1903 \newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%
1904   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignorespaces}
1905 \newcommand\@symiii@star[4][]{\metasetkeys{symi}{#1}%
1906   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi\ignorespaces}
1907 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
1908 \newcommand\@symiv[5][]{\metasetkeys{symi}{#1}%
1909   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces}
1910 \newcommand\@symiv@star[5][]{\metasetkeys{symi}{#1}%
1911   \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` compar-



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

```

1912 %\srefaddidkey{importmhmodule}%
1913 \addmetakey{importmhmodule}{mhrepos}%
1914 \addmetakey{importmhmodule}{path}%
1915 \addmetakey{importmhmodule}{ext}% why does this exist?
1916 \addmetakey{importmhmodule}{dir}%
1917 \addmetakey[false]{importmhmodule}{conservative}[true]%
1918 \newcommand\importmhmodule[2][]{%
1919   \parsemodule@maybesetcodes
1920   \metasetkeys{importmhmodule}{#1}%
1921   \ifx\importmhmodule@dir\empty%
1922     \edef\@path{\importmhmodule@path}%
1923   \else\edef\@path{\importmhmodule@dir/#2}\fi%
1924   \ifx\@path\empty% if module name is not set
1925     \@importmodule[]{#2}{export}%
1926   \else%
1927     \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
1928     \ifx\importmhmodule@mhrepos\empty% if in the same repos
1929       \relax% no need to change mh@currentrepos, i.e., current directory.
1930     \else%
1931       \setcurrentreposinfo\importmhmodule@mhrepos% change it.
1932       \addto@thismodule{x}\noexpand\setcurrentreposinfo{\importmhmodule@mhrepos}}%
1933     \fi%
1934     \@importmodule[\MathHub{\mh@currentrepos/source/\@path}]{#2}{export}%
1935     \setcurrentreposinfo\mh@@repos% after importing, reset to old value
1936     \addto@thismodule{x}\noexpand\setcurrentreposinfo{\mh@@repos}}%
1937   \fi%
1938   \ignorespacesandpars%
1939 }

```

`\usemhmodule`

```

1940 \addmetakey{importmhmodule}{load}
1941 \addmetakey{importmhmodule}{id}
1942 \addmetakey{importmhmodule}{dir}
1943 \addmetakey{importmhmodule}{mhrepos}
1944
1945 \addmetakey{importmodule}{load}
1946 \addmetakey{importmodule}{id}
1947
1948 \newcommand\usemhmodule[2][]{%
1949   \metasetkeys{importmhmodule}{#1}%
1950   \ifx\importmhmodule@dir\empty%
1951     \edef\@path{\importmhmodule@path}%
1952   \else\edef\@path{\importmhmodule@dir/#2}\fi%
1953   \ifx\@path\empty%
1954     \usemodule[id=\importmhmodule@id]{#2}%
1955   \else%
1956     \edef\mh@@repos{\mh@currentrepos}%

```

```

1957 \ifx\importmhmodule@mhrepos\@empty%
1958 \else\setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
1959 \usemodule{\@path\@QuestionMark#2}%
1960 %\usemodule[load=MathHub{\mh@currentrepos/source/\@path},
1961 %                                id=\importmhmodule@id]{#2}%
1962 \setcurrentreposinfo\mh@trepos%
1963 \fi%
1964 \ignorespacesandpars}

\mhinputref
1965 \newcommand\mhinputref[2][{}]{%
1966   \edef\mhinputref@first{#1}%
1967   \ifx\mhinputref@first\@empty%
1968     \inputref{#2}%
1969   \else%
1970     \inputref[mhrepos=\mhinputref@first]{#2}%
1971   \fi%
1972 }

\trefi*
1973 \newcommand\trefi[2][{}]{%
1974   \edef\trefi@mod{#1}%
1975   \ifx\trefi@mod\@empty\tref{#2}\else\tref{#1\@QuestionMark#2}\fi%
1976 }
1977 \newcommand\trefii[3][{}]{%
1978   \edef\trefi@mod{#1}%
1979   \ifx\trefi@mod\@empty\tref{#2-#3}\else\tref{#1\@QuestionMark#2-#3}\fi%
1980 }

\defi*
1981 \def\defii#1#2{\defi{#1!#2}}
1982 \def\Defii#1#2{\Defi{#1!#2}}
1983 \def\defiis#1#2{\defis{#1!#2}}
1984 \def\Defiis#1#2{\Defis{#1!#2}}
1985 \def\defiii#1#2#3{\defi{#1!#2!#3}}
1986 \def\Defiii#1#2#3{\Defi{#1!#2!#3}}
1987 \def\defiiis#1#2#3{\defis{#1!#2!#3}}
1988 \def\Defiiis#1#2#3{\Defis{#1!#2!#3}}
1989 \def\defiv#1#2#3#4{\defi{#1!#2!#3!#4}}
1990 \def\Defiv#1#2#3#4{\Defi{#1!#2!#3!#4}}
1991 \def\defivs#1#2#3#4{\defis{#1!#2!#3!#4}}
1992 \def\Defivs#1#2#3#4{\Defis{#1!#2!#3!#4}}
1993 \def\adefi#1#2{\defi[name=#2]{#1}}
1994 \def\adefii#1#2#3{\defi[name=#2-#3]{#1}}
1995 \def\adefiii#1#2#3#4{\defi[name=#2-#3-#4]{#1}}
1996 \def\adefiv#1#2#3#4#5{\defi[name=#2-#3-#4-#5]{#1}}

1997 \newlinechar=\old@newlinechar

```