

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

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

December 10, 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
15 \RequirePackage{standalone}
16 \RequirePackage{xspace}
17 \RequirePackage{metakeys}
18
```

```

19 \ifcsvoid{if@latexml}{
20   \newif{if@latexml}@latexmlfalse
21 }{}

```

### 3.1 sTeX base

The sTeX logo:

```

22 \protected\def\stex{%
23   \@ifundefined{texorpdfstring}%
24   {\let\texorpdfstring\@firstoftwo}%
25   }%
26   \texorpdfstring{\raisebox{-.5ex}{S}\kern-.5ex\TeX}{sTeX}\xspace%
27 }
28 \def\sTeX{\stex}

```

and a conditional for LaTeXML:

```

29 \newif{if@latexml}@latexmlfalse

```

### 3.2 Paths and URIs

```

30 \RequirePackage{xstring}
31 \RequirePackage{etoolbox}

```

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

```

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

```

#### 3.2.1 Path Canonicalization

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

```

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

```

```

44 \catcode'\?=12\relax%
45 }
46 \def\pathsuris@resetcatcodes{%
47 \catcode'\#\pathsuris@oldcatcode@hash\relax%
48 \catcode'\/\pathsuris@oldcatcode@slash\relax%
49 \catcode'\:\pathsuris@oldcatcode@colon\relax%
50 \catcode'\?\pathsuris@oldcatcode@qm\relax%
51 }

```

We define some macros for later comparison.

```

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

```

\@cpath Canonicalizes (file) paths:

```

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

```

```

90      \StrGobbleRight\@CanPath1[\@CanPath]%
91      \fi%
92    }{}%
93 }
94
95 \def\@cpath@loop{%
96   \IfSubStr\pathsuris@cpath@temp\@Slash{%
97     \StrCut\pathsuris@cpath@temp\@Slash\pathsuris@cpath@temp@a\pathsuris@cpath@temp%
98     \ifx\pathsuris@cpath@temp@a\@ToTop%
99       \ifx\@CanPath\@empty%
100         \edef\@CanPath{\@ToTop}%
101       \else%
102         \edef\@CanPath{\@CanPath\@Slash\@ToTop}%
103       \fi%
104       \@cpath@loop%
105     \else%
106     \ifx\pathsuris@cpath@temp@a\@Dot%
107       \@cpath@loop%
108     \else%
109     \IfBeginWith\pathsuris@cpath@temp\@ToTop{%
110       \StrBehind{\pathsuris@cpath@temp}{\@ToTop}[\pathsuris@cpath@temp]%
111       \IfBeginWith\pathsuris@cpath@temp\@Slash{%
112         \edef\pathsuris@cpath@temp{\@CanPath\pathsuris@cpath@temp}%
113       }{%
114         \ifx\@CanPath\@empty\else%
115           \edef\pathsuris@cpath@temp{\@CanPath\@Slash\pathsuris@cpath@temp}
116         \fi%
117       }%
118       \def\@CanPath{}%
119       \@cpath@loop%
120     }{%
121       \ifx\@CanPath\@empty%
122         \edef\@CanPath{\pathsuris@cpath@temp@a}%
123       \else%
124         \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp@a}%
125       \fi%
126       \@cpath@loop
127     }%
128   \fi\fi%
129 }{
130   \ifx\@CanPath\@empty%
131     \edef\@CanPath{\pathsuris@cpath@temp}%
132   \else%
133     \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp}%
134   \fi%
135 }%
136 }

```

**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.

```

137 \newcommand\cpath[1]{%
138   \@cpath{#1}%
139   \@CanPath%
140 }
```

`\path@filename`

```

141 \def\path@filename#1#2{%
142   \edef\filename@oldpath{#1}%
143   \StrCount\filename@oldpath\@Slash[\filename@lastslash]%
144   \ifnum\filename@lastslash>0%
145     \StrBehind[\filename@lastslash]\filename@oldpath\@Slash[\filename@oldpath]%
146     \edef#2{\filename@oldpath}%
147   \else%
148     \edef#2{\filename@oldpath}%
149   \fi%
150 }
```

**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:

```

151 \newif\if@iswindows@\@iswindows@false
152 \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:

```

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

```

155 \@windowstopath@inpath@false
156 \def\windows@temp{}
157 \edef\windows@path{#1}
158 \ifx\windows@path\@empty\else
159 \expandafter\windows@path@loop\windows@path\windows@path@end
160 \fi
161 \let#1\windows@temp
162 }
163 \def\windows@path@loop#1#2\windows@path@end{
164 \def\windows@temp@b{#2}
165 \ifx\windows@temp@b\@empty
166 \def\windows@continue{}
167 \else
168 \def\windows@continue{\windows@path@loop#2\windows@path@end}
169 \fi
170 \if@windowstopath@inpath@
171 \ifx#1\@BackSlash
172 \edef\windows@temp{\windows@temp\@Slash}
173 \else
174 \edef\windows@temp{\windows@temp#1}
175 \fi
176 \else
177 \ifx#1:
178 \edef\windows@temp{\@Slash\windows@temp}
179 \@windowstopath@inpath@true
180 \else
181 \edef\windows@temp{\windows@temp#1}
182 \fi
183 \fi
184 \windows@continue
185 }

```

#### 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:

```

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

```



```

199     \def\windows@continue{}
200   \else
201     \def\windows@continue{\path@windows@loop#2\windows@path@end}
202   \fi
203   \if@windowstopath@inpath@
204     \ifx#1/
205       \edef\windows@temp{\windows@temp\@BackSlash}
206     \else
207       \edef\windows@temp{\windows@temp#1}
208     \fi
209   \else
210     \ifx#1/
211       \edef\windows@temp{\windows@temp:\@BackSlash}
212       \@windowstopath@inpath@true
213     \else
214       \edef\windows@temp{\windows@temp#1}
215     \fi
216   \fi
217   \windows@continue
218 }

```

**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:

```

219 \def\trimstring#1{%
220   \edef\pathsuris@trim@temp{#1}%
221   \IfBeginWith\pathsuris@trim@temp\@Space{%
222     \StrGobbleLeft\pathsuris@trim@temp1[#1]%
223     \trimstring{#1}%
224   }{%
225     \IfEndWith\pathsuris@trim@temp\@Space{%
226       \StrGobbleRight\pathsuris@trim@temp1[#1]%
227       \trimstring{#1}%
228     }{%
229       \edef#1{\pathsuris@trim@temp}%
230     }%
231   }%
232 }

```

**Test:**

»bla blubb«

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

```

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}

```

**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:

```

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

```

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

```

```

275     \global\let\stex@currfilename\stex@currfilename%
276     \fi%
277 }

```

**\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:

```

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

```

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

```

300 \def\findmanifest#1{
301     \@cpath{#1}
302     \ifx\@CanPath\@Slash
303         \def\manifest@mf{}
304     \else\ifx\@CanPath\@empty
305         \def\manifest@mf{}
306     \else
307         \edef\@findmanifest@path{\@CanPath/MANIFEST.MF}

```

```

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

```

### Test:

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

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

```

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

```

the next helper function iterates over lines in MANIFEST.MF

```

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

```

```

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

```

`\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.

```

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

```

**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.

```

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

```

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

```

429 \newif\if@inmhrepos\@inmhreposfalse
430 \ifcsvoid{stex@maindir}{%
431   \parsemanifest{currentrepos@}\stex@maindir
432   \@setcurrentreposinfo
433   \ifcsvoid{currentrepos@dir}\PackageWarning{stex}{Not currently in a MathHub repository}{%
434     \message{Current repository: \mh@currentrepos}

```

```

435 }
436 }

```

### 3.3 Modules

```

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

```

Aux:

```

438 \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>

```

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

```

```

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

```

Options for the module-environment:

```

441 \addmetakey*{module}{title}

```

```

442 \addmetakey*{module}{name}

```

```

443 \addmetakey*{module}{creators}

```

```

444 \addmetakey*{module}{contributors}

```

```

445 \addmetakey*{module}{srccite}

```

```

446 \addmetakey*{module}{ns}

```

```

447 \addmetakey*{module}{narr}

```

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

```

448 \ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%

```

```

449 \newrobustcmd\module@heading{%

```

```

450 \stepcounter{module}%

```

```

451 \ifmod@show%

```

```

452 \noindent{\textbf{Module} \thesection.\thetitle [\module@name]]%

```

```

453 \sref@label{id}{Module \thesection.\thetitle [\module@name]]%

```

```

454 \ifx\module@title\empty : \quad\else\quad(\module@title)\hfill\\\fi%

```

```

455 \fi%

```

```

456 }%

```

**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.

```

457 \newenvironment{module}[1][{}]{%

```

```

458 \begin{@module}[#1]%

```

```

459 \module@heading% make the headings

```

```

460 \ignorespacesandpars\parsemodule@maybesetcodes}{%

```

```

461 \end{@module}%

```

```

462 \ignorespacesafterend%

```

```

463 }%

```

```

464 \ifmod@show\surroundwithmdframed{module@om@common}\fi%

```

Some auxiliary methods:

```

465 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}

```

```

466 \def\addto@thismodule#1{%

```

```

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

```

**@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`.

```

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

```

### 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.

```

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

```

### 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.

```

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

```

```

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

```

**Test:**

**Module 3.2[Foo]:**

Name: Foo

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

this@module: macro:->

**Test:**

Faking a MathHub archive Foo/Bar with URI <http://foo.bar/baz>:

**Module 3.3[Foo2]:**

Name: Foo2

URI: <http://foo.bar/baz?Foo2>

this@module: macro:->\edef\mh@old@repos@<http://foo.bar/baz?Foo2>{\mh@currentrepos}  
\setcurrentreposinfo {Foo/Bar}

**Test:**

Removing the /home/jazzpirate/work/MathHub/ system variable first:

**Module 3.4[Foo]:**

Name: Foo

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

this@module: macro:->Faking a MathHub archive Foo/Bar with URI <http://foo.bar/baz>:

**Module 3.5[Foo2]:**

Name: Foo2

URI: <http://foo.bar/baz?Foo2>

```
this@module: macro:->\edef \mh@old@repos@http://foo.bar/baz?Foo2 {\mh@currentrepos
}\setcurrentreposinfo {Foo/Bar}
```

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

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

## 3.4 Inheritance

### 3.4.1 Selective Inclusion

The next great goal is to establish the  $\backslash\text{requiremodules}$  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  $\backslash\text{requiremodules}$ .

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

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

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

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

```

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

```

604 \catcode'\.=0
605 .catcode'\=13

```

```

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

```

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

`\set@parsemodule@catcodes`

```

619 \def\set@parsemodule@catcodes{%
620     \global\catcode'\=13%
621     \global\catcode'\#=12%
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     \expandafter\let\@active@slash\parsemodule@escapechar%
629 }

```

`\reset@parsemodule@catcodes`

```

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

```

`\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.

```

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

```

`\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:

```
643
644 \def\parsemodule@escapechar{%
645     \def\parsemodule@escape@currcls{}%
646     \parsemodule@escape@parse@nextchar%
647 }%
```

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`.

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

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.

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

```

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

```

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.

```

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

```

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.

```

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

```

```

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

```

`\@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.

```

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

```

`\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.

```

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

```

`\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.

```

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

```

#### 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`

```

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

```

`\importmodule`

```

765 %\srefaddidkey{importmodule}
766 \addmetakey{importmodule}{mhrepos}
767 \newcommand\importmodule[2][\@importmodule{#1}{#2}{export}]
768 \newcommand\@importmodule[3][\@importmodule@bookkeeping{#1}{#2}{%
769   \importmodule@bookkeeping{#1}{#2}{%
770     \@importmodule[\importmodule@dir]\importmodule@modulename{#3}%
771   }%
772 }

```

`\@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`.

```

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

```

**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:

```

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

```

`\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.

```

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

```

```

843     }
844   }{%
845     \ifcsundef{module@\csname stex@module@#1\endcsname\@URI @activated}%
846     {\csname module@defs@\csname stex@module@#1\endcsname\@URI\endcsname}{}}%
847     \namedef{module@\csname stex@module@#1\endcsname\@URI @activated}{true}%
848   }%
849 }%

```

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

```
850 \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.

```

851 \def\inputref@preskip{}
852 \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).

```

853 \newrobustcmd\inputref[2] [] {%
854   \importmodule@bookkeeping{#1}{#2}{%
855     %\inputreftrue
856     \inputref@preskip%
857     \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
858     \inputref@postskip%
859   }%
860 }%

```

## 3.5 Symbols/Notations/Verbalizations

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

```
861 \newif\if@symdeflocal\@symdeflocalfalse
```

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

```

862 \def\define@in@module#1#2{
863   \expandafter\edef\csname #1\endcsname{#2}%
864   \edef\define@in@module@temp{%
865     \def\expandafter\noexpand\csname#1\endcsname%
866     {#2}%
867   }%
868   \if@symdeflocal\else%

```

```

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

\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.

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

```

```

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

```

**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:

```

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

```

```

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

```

```

\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}{...}

1003 \newif@if@inverbalization\@inverbalizationfalse
1004 % parses the first two arguments:
1005 \providerobustcmd\notation[2][ ]{%
1006   \edef\notation@first{#1}%
1007   \edef\notation@second{#2}%
1008   \notation@%
1009 }
1010
1011 \providerobustcmd\verbalization{%
1012   \@inverbalizationtrue%
1013   \notation%
1014 }
1015
1016 % parses the last two arguments
1017 \newcommand\notation@[2][0]{%
1018   \edef\notation@donext{\noexpand\notation@[ \notation@first]%
1019     {\notation@second}[#1]}%
1020   \notation@donext{#2}%
1021 }
1022
1023 % parses the notation arguments and wraps them in
1024 % \notation@assoc and \notation@argprec for flexary arguments and precedences
1025 \def\notation@[#1]#2[#3]#4{%
1026   \modules@getURIfromName{#2}%
1027   \notation@parse@params{#1}{#3}
1028   \let\notation@curr@todo@args\notation@curr@args%
1029   \def\notation@temp@notation{%
1030     \StrLen\notation@curr@args[\notation@temp@arity]%
1031     \expandafter\renewcommand\expandafter\notation@temp@notation%
1032       \expandafter[\notation@temp@arity]{#4}%
1033     % precedence
1034     \IfSubStr\notation@curr@args;{%
1035       \StrCut\notation@curr@args;\notation@curr@prec\notation@curr@args%
1036       \ifx\notation@curr@prec\@empty\def\notation@curr@prec{0}\fi%
1037     }{%
1038       \ifx\notation@curr@args\@empty%
1039         \ifnum\notation@temp@arity=0\relax%
1040           \edef\notation@curr@prec{\infprec}%
1041         \else%
1042           \def\notation@curr@prec{0}%
1043         \fi%
1044       \else%
1045         \edef\notation@curr@prec{\notation@curr@args}%
1046         \def\notation@curr@args{}%
1047       \fi%
1048     }%

```



```

1049 % arguments
1050 \def\notation@curr@extargs{}
1051 \def\notation@nextarg@index{1}%
1052 \notation@do@args%
1053 }
1054
1055 % parses additional notation components for (associative) arguments
1056 \def\notation@do@args{%
1057   \def\notation@nextarg@temp{}%
1058   \ifx\notation@curr@todo@args\@empty%
1059     \notation@after%
1060   \else%
1061     % argument precedence
1062     \IfSubStr\notation@curr@prec{s}{x}{%
1063       \StrCut\notation@curr@prec{s}{x}\notation@curr@argprec\notation@curr@prec%
1064     }{%
1065       \edef\notation@curr@argprec{\notation@curr@prec}%
1066       \def\notation@curr@prec{}%
1067     }%
1068     \ifx\notation@curr@argprec\@empty%
1069       \let\notation@curr@argprec\notation@curr@prec%
1070     \fi%
1071     \StrChar\notation@curr@todo@args1[\notation@argchar]%
1072     \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1073     \expandafter\ifx\notation@argchar i%
1074       % normal argument
1075       \edef\notation@nextarg@temp{\noexpand\notation@argprec{\notation@curr@argprec}{#####\
1076       \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }
1077       \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
1078       \expandafter{\notation@nextarg@temp}%
1079       \expandafter\expandafter\expandafter\notation@do@args%
1080     \else%
1081       % associative argument
1082       \expandafter\expandafter\expandafter\notation@parse@assocarg%
1083     \fi%
1084   \fi%
1085 }
1086
1087 \def\notation@parse@assocarg#1{%
1088   \edef\notation@nextarg@temp{\noexpand\notation@argprec{\notation@curr@argprec}{\noexpand\not
1089   \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }%
1090   \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
1091   \expandafter{\notation@nextarg@temp}%
1092   \notation@do@args%
1093 }
1094
1095 \protected\def\safe@newcommand#1{%
1096   \ifdefined#1\expandafter\renewcommand\else\expandafter\newcommand\fi#1%
1097 }
1098

```

```

1099 % finally creates the actual macros
1100 \def\notation@after{
1101   \let\ex\expandafter%
1102   \ex\ex\ex\def\ex\ex\ex\notation@temp\notation\ex\ex\ex%
1103     {\ex\notation@temp\notation\notation@curr@extargs}%
1104   \edef\notation@temp\notation{\noexpand\notation@symprec{\notation@curr@prec}{\ex\unexpanded\ex%
1105     \def\notation@temp@fragment{}}%
1106   \ifx\notation@curr@arity\@empty\else%
1107     \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1108   \fi%
1109   \ifx\notation@curr@lang\@empty\else%
1110     \ifx\notation@temp@fragment\@empty%
1111       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1112     \else%
1113       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1114     \fi%
1115   \fi%
1116   \ifx\notation@curr@variant\@empty\else%
1117     \ifx\notation@temp@fragment\@empty%
1118       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1119     \else%
1120       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1121     \fi%
1122   \fi%
1123   \if@inverbalization\@inverbalizationfalse\verbalization@final%
1124   \else\notation@final\fi%
1125   \parsemodule@maybesetcodes%
1126 }
1127
1128 \def\notation@final{%
1129   \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1130   \ifcsvoid{\notation@csname}{%
1131     \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1132       \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@temp@arity\ex]%
1133     \ex{\notation@temp\notation}%
1134     \edef\symdecl@temps{%
1135       \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@temp@ari
1136     }%
1137     \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1138     \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1139   }{%
1140     \PackageWarning{stex}{notation already defined: \notation@csname}{%
1141       Choose a different set of notation options (variant,lang,arity)%
1142     }%
1143   }%
1144 }
1145
1146 \def\verbalization@final{%
1147   \edef\notation@csname{\notation@uri\@Fragment verb\@Fragment\notation@temp@fragment}%
1148   \ifcsvoid{\notation@csname}{%

```

```

1149 \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1150 \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@temp@arity\ex]%
1151 \ex{\notation@temp@notation}%
1152 \edef\symdecl@temps{%
1153 \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@temp@ari
1154 }%
1155 \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1156 \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1157 }-%
1158 \PackageWarning{stex}{verbalization already defined: \notation@csname}%
1159 Choose a different set of verbalization options (variant,lang,arity)%
1160 }%
1161 }%
1162 }
1163
1164 % parses optional parameters
1165 \def\notation@parse@params#1#2{%
1166 \def\notation@curr@prec{%}%
1167 \def\notation@curr@args{%}%
1168 \def\notation@curr@variant{%}%
1169 \def\notation@curr@arity{%}%
1170 \def\notation@curr@provided@arity#{#2}%
1171 \def\notation@curr@lang{%}%
1172 \def\notation@options@temp#{#1}%
1173 \notation@parse@params@%
1174 \ifx\notation@curr@args\@empty%
1175 \ifx\notation@curr@provided@arity\@empty%
1176 \notation@num@to@ia\notation@curr@arity%
1177 \else%
1178 \notation@num@to@ia\notation@curr@provided@arity%
1179 \fi%
1180 \fi%
1181 }
1182 \def\notation@parse@params@{%
1183 \IfSubStr\notation@options@temp,{%
1184 \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
1185 \notation@parse@param%
1186 \notation@parse@params@%
1187 }{\ifx\notation@options@temp\@empty\else%
1188 \let\notation@option@temp\notation@options@temp%
1189 \notation@parse@param%
1190 \fi}%
1191 }
1192
1193 %parses an individual optional argument/key-value-pair
1194 \def\notation@parse@param{%
1195 \trimstring\notation@option@temp%
1196 \ifx\notation@option@temp\@empty\else%
1197 \IfSubStr\notation@option@temp=%{%
1198 \StrCut\notation@option@temp=\notation@key\notation@value%

```

```

1199     \trimstring\notation@key%
1200     \trimstring\notation@value%
1201     \IfStrEq\notation@key{prec}{%
1202         \edef\notation@curr@prec{\notation@value}%
1203     }{%
1204     \IfStrEq\notation@key{args}{%
1205         \edef\notation@curr@args{\notation@value}%
1206     }{%
1207     \IfStrEq\notation@key{lang}{%
1208         \edef\notation@curr@lang{\notation@value}%
1209     }{%
1210     \IfStrEq\notation@key{variant}{%
1211         \edef\notation@curr@variant{\notation@value}%
1212     }{%
1213     \IfStrEq\notation@key{arity}{%
1214         \edef\notation@curr@arity{\notation@value}%
1215     }{%
1216     }}}}%
1217 }{%
1218     \edef\notation@curr@variant{\notation@option@temp}%
1219 }%
1220 \fi%
1221 }
1222
1223 % converts an integer to a string of 'i's, e.g. 3 => iii,
1224 % and stores the result in \notation@curr@args
1225 \def\notation@num@to@ia#1{%
1226     \IfInteger{#1}{
1227         \notation@num@to@ia@#1%
1228     }{%
1229         %
1230     }%
1231 }
1232 \def\notation@num@to@ia@#1{%
1233     \ifnum#1>0%
1234         \edef\notation@curr@args{\notation@curr@args i}%
1235         \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
1236     \fi%
1237 }

```

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

```

1238 \def\notation@assoc#1#2{% function, argv
1239     \let\@tmpop=\relax% do not print the function the first time round
1240     \@for\@I:=#2\do{\@tmpop% print the function
1241         % write the i-th argument with locally updated precedence
1242         \@I%
1243     \def\@tmpop{#1}%
1244     }%
1245 }%

```

```

1246
1247 \def\notation@lparen{()
1248 \def\notation@rparen{)}}
1249 \def\infprec{1000000}
1250 \def\neginfprec{-\infprec}
1251
1252 \newcount\notation@downprec
1253 \notation@downprec=\neginfprec
1254
1255 % patching displaymode
1256 \newif\if@displaymode\@displaymodefalse
1257 \expandafter\everydisplay\expandafter{\the\everydisplay\@displaymodetrue}
1258 \let\old@displaystyle\displaystyle
1259 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1260
1261 \def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1262   \def\notation@innertmp{#1}%
1263   \let\ex\expandafter%
1264   \if@displaymode%
1265     \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1266     \ex\notation@resetbrackets\ex\notation@innertmp%
1267     \ex\right\notation@rparen%
1268   \else%
1269     \ex\ex\ex\notation@lparen%
1270     \ex\notation@resetbrackets\ex\notation@innertmp%
1271     \notation@rparen%
1272   \fi%
1273 }
1274
1275 \def\withbrackets#1#2#3{%
1276   \edef\notation@lparen{#1}%
1277   \edef\notation@rparen{#2}%
1278   #3%
1279   \notation@resetbrackets%
1280 }
1281
1282 \def\notation@resetbrackets{%
1283   \def\notation@lparen{()%
1284   \def\notation@rparen{)}}%
1285 }
1286
1287 \def\notation@symprec#1#2{%
1288   \ifnum#1>\notation@downprec\relax%
1289     \notation@resetbrackets#2%
1290   \else%
1291     \ifnum\notation@downprec=\infprec\relax%
1292       \notation@resetbrackets#2%
1293     \else
1294       \if@inarray@
1295         \notation@resetbrackets#2

```

```

1296         \else\dobrackets{#2}\fi%
1297     \fi\fi%
1298 }
1299
1300 \newif\if@inarray@\@inarray@false
1301
1302 \def\notation@argprec#1#2{%
1303     \def\notation@innertmp{#2}
1304     \edef\notation@downprec@temp{\number#1}%
1305     \notation@downprec=\expandafter\notation@downprec@temp%
1306     \expandafter\relax\expandafter\notation@innertmp%
1307     \expandafter\notation@downprec\expandafter=\number\notation@downprec\relax%
1308 }

```

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

```

1309 \protected\def\@invoke@symbol#1{%
1310     \def\@invoke@symbol@first{#1}%
1311     \symbol@args%
1312 }

```

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

```

1313 \newcommand\symbol@args[1][]{%
1314     \notation@parse@params{#1}{}%
1315     \def\notation@temp@fragment{}%
1316     \ifx\notation@curr@arity\@empty\else%
1317         \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1318     \fi%
1319     \ifx\notation@curr@lang\@empty\else%
1320         \ifx\notation@temp@fragment\@empty%
1321             \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1322         \else%
1323             \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1324         \fi%
1325     \fi%
1326     \ifx\notation@curr@variant\@empty\else%
1327         \ifx\notation@temp@fragment\@empty%
1328             \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1329         \else%
1330             \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1331         \fi%
1332     \fi%
1333     %
1334     \ifmmode\def\invoke@symbol@next{\@invoke@symbol@math\@invoke@symbol@first\notation@temp@fragm
1335     \else\def\invoke@symbol@next{\@invoke@symbol@text\@invoke@symbol@first\notation@temp@fragment
1336     \invoke@symbol@next%
1337 }

```

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

a LaTeXML binding:

```
1338 \def\@invoke@symbol@math#1#2{%
1339   \csname #1\@Fragment#2\endcsname%
1340 }
```

TODO:

```
1341 \def\@invoke@symbol@text#1#2{%
1342   \@termref{#1}{\csname #1\@Fragment verb\@Fragment#2\endcsname}%
1343 }
```

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

```
1344 \def\setstexlang#1{%
1345   \def\stex@lang{#1}%
1346 }%
1347 \setstexlang{en}
1348 \def\setstexvariant#1#2{%
1349   % TODO
1350 }
1351 \def\setstexvariants#1{%
1352   \def\stex@variants{#1}%
1353 }
```

**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`

```
1354 \newif\ifhref\hreffalse%
1355 \AtBeginDocument{%
1356   \ifpackageloaded{hyperref}{%
1357     \hreftrue%
1358   }{%
1359     \hreffalse%
1360   }%
1361 }
```

`\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.

```
1362 \newbox\stex@targetbox
1363 \def\termref@maketarget#1#2{%
1364   % #1: symbol URI
1365   % #2: text
1366   \message{^^JHere: #1 <> #2^^J}%
1367   \ifhref\if@smsmode\else%
1368     \hypertarget{sref@#1@target}{#2}%
1369   \fi\fi%
1370   \message{^^JHere!^^J}%
1371   \expandafter\edef\csname sref@#1\endcsname##1{%
1372     \ifhref\if@smsmode\else\noexpand\hyperlink{sref@#1@target}{##1}\fi\fi%
1373   }%
1374 }
```

`\@termref`

```
1375 \def\@termref#1#2{%
```



```

1376 % #1: symbol URI
1377 % #2: text
1378 \ifcsvoid{#1}{%
1379   \StrCut[2]{#1}\@QuestionMark\termref@mod\termref@name%
1380   \ifcsvoid{\termref@mod}{%
1381     \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
1382   }{%
1383     \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
1384       contains no symbol with name \termref@name.%
1385     }{%
1386     }%
1387   }{%
1388     \ifcsvoid{sref@#1}{%
1389       #2% TODO: No reference point exists!
1390     }{%
1391       \csname sref@#1\endcsname{#2}%
1392     }%
1393   }%
1394 }

```

\tref

```

1395
1396 \def\@capitalize#1{\uppercase{#1}}%
1397 \newrobustcmd\capitalize[1]{\expandafter\@capitalize #1}%
1398
1399 \newcommand\tref[2][]{%
1400   \edef\tref@name{#1}%
1401   \ifx\tref@name\@empty
1402     \symdecl@constructname{#2}%
1403     \edef\tref@name{\symdecl@name}%
1404   \else%
1405     \edef\symdecl@verbalization{#2}%
1406   \fi%
1407   \expandafter\modules@getURIfromName\expandafter{\tref@name}%
1408   \expandafter\@termref\expandafter{\notation@uri}{\symdecl@verbalization}%
1409 }
1410 \def\trefs#1{%
1411   \modules@getURIfromName{#1}%
1412   \expandafter\@termref\expandafter{\notation@uri}{\csname\notation@uri\@Fragment verb\@Fragment
1413 }
1414 \def\Tref#1{%
1415   \modules@getURIfromName{#1}%
1416   \expandafter\@termref\expandafter{\notation@uri}{\expandafter\capitalize\csname\notation@uri\
1417 }
1418 \def\Trefs#1{%
1419   \modules@getURIfromName{#1}%
1420   \expandafter\@termref\expandafter{\notation@uri}{\expandafter\capitalize\csname\notation@uri\
1421 }

```

**Test:**

foo bar  
foo-bar  
finite group

```
\defi
1422 \addmetakey{defi}{name}
1423 \def\@definiendum#1#2{%
1424   \parsemodule@maybesetcodes%
1425   \message{^^JHere: #1 | #2^^J}%
1426   \termref@maketarget{#1}{#2}%\termref@maketarget{#1}{\defemph{#2}}%
1427 }
1428
1429 \newcommand\defi[2][{}]{%
1430   \metasetkeys{defi}{#1}%
1431   \ifx\defi@name\@empty%
1432     \symdecl@constructname{#2}%
1433     \let\defi@name\symdecl@name%
1434     \let\defi@verbalization\symdecl@verbalization%
1435   \else%
1436     \edef\defi@verbalization{#2}%
1437   \fi%
1438   \ifcvoid{\module@uri\@QuestionMark\defi@name}{%
1439     \symdecl\defi@name%
1440   }{\edef\symdecl@uri{\module@uri\@QuestionMark\defi@name}}%
1441   \@definiendum\symdecl@uri\defi@verbalization%
1442 }
1443 \def\Defi#1{%
1444   \symdecl{#1}%
1445   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization}%
1446 }
1447 \def\defis#1{%
1448   \symdecl{#1}%
1449   \@definiendum\symdecl@uri{\symdecl@verbalization s}%
1450 }
1451 \def\Defis#1{%
1452   \symdecl{#1}%
1453   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization s}%
1454 }
```

**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
```

```

1455 \newif\ifhref\hreffalse%
1456 \AtBeginDocument{%
1457   \ifpackageloaded{hyperref}{%
1458     \hreftrue%
1459   }{%
1460     \hreffalse%
1461   }%
1462 }%
1463 \newcommand\sref@href@ifh[2]{%
1464   \ifhref%
1465     \href{#1}{#2}%
1466   \else%
1467     #2%
1468   \fi%
1469 }%
1470 \newcommand\sref@hlink@ifh[2]{%
1471   \ifhref%
1472     \hyperlink{#1}{#2}%
1473   \else%
1474     #2%
1475   \fi%
1476 }%
1477 \newcommand\sref@target@ifh[2]{%
1478   \ifhref%
1479     \hypertarget{#1}{#2}%
1480   \else%
1481     #2%
1482   \fi%
1483 }%

```

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.

```

1484 \def\sref@target{%
1485   \ifx\sref@id\@empty%
1486     \relax%
1487   \else%
1488     \edef\@target{\sref@ifcsundef{sref@part}}{\sref@part @}\sref@id @target}%
1489     \sref@target@ifh\@target{}%
1490   \fi%
1491 }%

```

**\srefaddidkey** **\srefaddidkey**[*keyval*]{*group*} extends the metadata keys of the *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.

```

1492 \addmetakey{srefaddidkey}{prefix}
1493 \newcommand\srefaddidkey[2][]{%
1494   \metasetkeys{srefaddidkey}{#1}%
1495   \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
1496   \metakeys@ext@clear@keys{#2}{id}{}%
1497   \metakeys@ext@showkeys{#2}{id}%
1498   \define@key{#2}{id}{%
1499     \edef\sref@id{\srefaddidkey@prefix ##1}%
1500     %\expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
1501     \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1502   }%
1503 }%

\@sref@def This macro stores the value of its last argument in a custom macro for reference.
1504 \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.
1505 \ifextrefs%
1506   \newwrite\refs@file%
1507 \else%
1508   \def\refs@file{\@auxout}%
1509 \fi%

\sref@def This macro writes an \@sref@def command to the current aux file and also exe-
cutes it.
1510 \newcommand\sref@def[3]{%
1511   \protected@write\refs@file{}{\string\sref@def{#1}{#2}{#3}}%
1512 }%

\sref@label The \sref@label macro writes a label definition to the auxfile.
1513 \newcommand\sref@label[2]{%
1514   \sref@def{\ifcsundef{sref@part}{}\sref@part @}{#2}{page}{\thepage}%
1515   \sref@def{\ifcsundef{sref@part}{}\sref@part @}{#2}{label}{#1}%
1516 }%

\sreflabel The \sreflabel macro is a semantic version of \label, it combines the catego-
rization given in the first argument with LATEX's \@currentlabel.
1517 \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.
1518 \def\sref@id{} % make sure that defined
1519 \newcommand\sref@label@id[1]{%
1520   \ifx\sref@id\@empty%
1521     \relax%
1522   \else%
1523     \sref@label{#1}{\sref@id}%
1524   \fi%
1525 }%

```

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

```

1526 \newcommand\sref@label@id@arg[2]{%
1527   \def\@@id{#2}
1528   \ifx\@@id\@empty%
1529     \relax%
1530   \else%
1531     \sref@label{#1}{\@@id}%
1532   \fi%
1533 }%
```

### 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`.

```

1534 \newenvironment{modsig}[2][\def\@test{#1}%
1535 \ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1536 \expandafter\gdef\csname mod@#2\multiling\endcsname{true}%
1537 \ignorespacesandpars}
1538 {\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`.

```

1539 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1540 \newrobustcmd\@gimport@star[2][\def\@test{#1}%
1541 \edef\mh@repos{\mh@currentrepos}%
1542 \ifx\@test\@empty%
1543   \importmhmodule[conservative,mhrepos=\mh@repos,path=#2]{#2}%
1544 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1545 \setcurrentreposinfo{\mh@repos}%
1546 \ignorespacesandpars\parsemodule@maybesetcodes}
1547 \newrobustcmd\@gimport@nostar[2][\def\@test{#1}%
1548 \edef\mh@repos{\mh@currentrepos}%
```

```

1549 \ifx\@test\@empty%
1550 \importmhmodule[mhrepos=\mh@repos,path=#2]{#2}%
1551 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1552 \setcurrentreposinfo{\mh@repos}%
1553 \ignorespacesandpars\parsemodule@maybesetcodes}

```

### 3.10 mathhub

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

```

1554 \def\modules@@first#1/#2;{#1}
1555 \newcommand\libinput[1]{%
1556 \ifcsvoid{mh@currentrepos}{%
1557   \PackageError{stex}{current MathHub repository not found}{}}%
1558 {}
1559 \edef\@mh@group{\expandafter\modules@@first\mh@currentrepos;}
1560 \let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
1561 \def\mh@inffile{\MathHub{\@mh@group/meta-inf/lib/#1}}
1562 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1563 \IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
1564 \IfFileExists\mh@inffile{}\IfFileExists\mh@libfile{}{%
1565   {\PackageError{stex}
1566     {Library file missing; cannot input #1.tex\MessageBreak%
1567       Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
1568       do not exist}%
1569     {Check whether the file name is correct}}}%
1570 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1571 \let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}

```

### 3.11 omdoc/omgroup

```

1572 \newcount\section@level
1573
1574 \section@level=2
1575 \ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{%
1576 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{%
1577 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{%
1578 \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>*.

```

1579 \newcommand\omgroup@nonum[2]{%
1580 \ifx\hyper@anchor\@undefined\else\phantomsection\fi%
1581 \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.

```

1582 \newcommand\omgroup@num[2]{%
1583 \edef\@@ID{\sref@id}
1584 \ifx\omgroup@short\empty% no short title
1585 \@nameuse{#1}{#2}%
1586 \else% we have a short title
1587 \@ifundefined{rdfmeta@sectioning}%
1588   {\@nameuse{#1}[\omgroup@short]{#2}}%
1589   {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
1590 \fi%
1591 \sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@@ID}

```

omgroup

```

1592 \def\@true{true}
1593 \def\@false{false}
1594 \srefaddidkey{omgroup}
1595 \addmetakey{omgroup}{date}
1596 \addmetakey{omgroup}{creators}
1597 \addmetakey{omgroup}{contributors}
1598 \addmetakey{omgroup}{srccite}
1599 \addmetakey{omgroup}{type}
1600 \addmetakey*{omgroup}{short}
1601 \addmetakey*{omgroup}{display}
1602 \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.

```

1603 \newif\if@mainmatter\@mainmattertrue
1604 \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.

```

1605 \addmetakey{omdoc@sect}{name}
1606 \addmetakey[false]{omdoc@sect}{clear}[true]
1607 \addmetakey{omdoc@sect}{ref}
1608 \addmetakey[false]{omdoc@sect}{num}[true]
1609 \newcommand\omdoc@sectioning[3][]{\metasetkeys{omdoc@sect}{#1}%
1610 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
1611 \if@mainmatter% numbering not overridden by frontmatter, etc.
1612 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
1613 \def\current@section@level{\omdoc@sect@name}%
1614 \else\omgroup@nonum{#2}{#3}%
1615 \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.

```

1616 \newcommand\omgroup@redefine@addtocontents[1]{%

```

```

1617 %\edef\@import{#1}%
1618 %\@for\@I:=\@import\do{%
1619 %\edef\@path{\csname module@\@I @path\endcsname}%
1620 %\@ifundefined{tf@toc}\relax%
1621 %    {\protected@write\tf@toc}{\string\@requiremodules{\@path}}}%
1622 %\ifx\hyper@anchor\@undefined% hyperref.sty loaded?
1623 %\def\addcontentsline##1##2##3{%
1624 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}%
1625 %\else% hyperref.sty not loaded
1626 %\def\addcontentsline##1##2##3{%
1627 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}{\@c
1628 %\fi
1629 }% 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.

1630 \newcount\omgroup@level
1631 \newenvironment{omgroup}[2] []% keys, title
1632 {\metasetkeys{omgroup}{#1}\sref@target%
1633 \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.

1634 \ifx\omgroup@loadmodules\@true%
1635 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
1636 {\@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.

1637 \advance\section@level by 1\relax%
1638 \ifcase\section@level%
1639 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
1640 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
1641 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
1642 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
1643 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%
1644 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{#2}%
1645 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{#2}%
1646 \fi% \ifcase
1647 \at@begin{omgroup}[#1]\section@level{#2}}% for customization
1648 {\advance\section@level by -1\advance\omgroup@level by -1}

    and finally, we localize the sections

1649 \newcommand\omdoc@part@kw{Part}
1650 \newcommand\omdoc@chapter@kw{Chapter}
1651 \newcommand\omdoc@section@kw{Section}
1652 \newcommand\omdoc@subsection@kw{Subsection}

```



```

1653 \newcommand\omdoc@subsubsection@kw{Subsubsection}
1654 \newcommand\omdoc@paragraph@kw{paragraph}
1655 \newcommand\omdoc@subparagraph@kw{subparagraph}

\setSGvar set a global variable
1656 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}

\useSGvar use a global variable
1657 \newrobustcmd\useSGvar[1]{%
1658   \ifundefined{sTeX@Gvar@#1}
1659   {\PackageError{omdoc}
1660     {The sTeX Global variable #1 is undefined}
1661     {set it with \protect\setSGvar}}
1662 \@nameuse{sTeX@Gvar@#1}}

blindomgroup
1663 \newcommand\at@begin@blindomgroup[1]{%
1664 \newenvironment{blindomgroup}
1665 {\advance\section@level by 1\at@begin@blindomgroup\setion@level}
1666 {\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).

```

1667 \srefaddidkey{omtext}
1668 \addmetakey[]{omtext}{functions}
1669 \addmetakey*{omtext}{display}
1670 \addmetakey{omtext}{for}
1671 \addmetakey{omtext}{from}
1672 \addmetakey{omtext}{type}
1673 \addmetakey*{omtext}{title}
1674 \addmetakey*{omtext}{start}
1675 \addmetakey{omtext}{theory}
1676 \addmetakey{omtext}{continues}
1677 \addmetakey{omtext}{verbalizes}
1678 \addmetakey{omtext}{subject}

\st@flow We define this macro, so that we can test whether the display key has the value
flow
1679 \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.

```

1680 \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.
1681 \def\omtext@pre@skip{\smallskip}
1682 \def\omtext@post@skip{}
1683 \newenvironment{omtext}[1][\@in@omtexttrue%
1684   \bgroup\metasetkeys{omtext}{#1}\sref@label{id{this paragraph}}%
1685   \def\lec##1{\@lec{##1}}}%
1686   \omtext@pre@skip\par\noindent%
1687   \ifx\omtext@title\@empty%
1688     \ifx\omtext@start\@empty\else%
1689       \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
1690     \fi% end omtext@start empty
1691   \else\stDMemph{\omtext@title}:\enspace%
1692     \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
1693   \fi% end omtext@title empty
1694   \ignorespacesandpars}
1695 {\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

```

1696 \srefaddidkey{phrase}
1697 \addmetakey{phrase}{style}
1698 \addmetakey{phrase}{class}
1699 \addmetakey{phrase}{index}
1700 \addmetakey{phrase}{verbalizes}
1701 \addmetakey{phrase}{type}
1702 \addmetakey{phrase}{only}
1703 \newcommand\phrase[2][\metasetkeys{phrase}{#1}%
1704 \ifx\phrase@only\@empty\only<\phrase@only>{#2}\else #2\fi}

```

`\coref*`

```

1705 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}
1706 \newcommand\corefs[2]{#1\textsubscript{#2}}
1707 \newcommand\coreft[2]{#1\textsuperscript{#2}}

```

`\n*lex`

```

1708 \newcommand\nlex[1]{\green{\sl{#1}}}
1709 \newcommand\nlcex[1]{*\green{\sl{#1}}}

```

`sinlinequote`

```

1710 \def\@sinlinequote#1{‘‘{\sl{#1}}}’’}
1711 \def\@sinlinequote#1#2{\@sinlinequote{#2}~#1}
1712 \newcommand\sinlinequote[2][
1713 {\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.

```
1714 \newcommand\vdec[2] [] {#2}
1715 \newcommand\vrest[2] [] {#2}
1716 \newcommand\vcond[2] [] {#2}
```

```
EdN:1 \strucdec 1
1717 \newcommand\strucdec[2] [] {#2}
```

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

### 3.12.4 Block-Level Markup

**sblockquote**

```
1719 \def\begin@sblockquote{\begin{quote}\sl}
1720 \def\end@sblockquote{\end{quote}}
1721 \def\begin@@sblockquote#1{\begin@sblockquote}
1722 \def\end@@sblockquote#1{\def\@lec#1{\textrm{##1}}\@lec{#1}\end@sblockquote}
1723 \newenvironment{sblockquote}[1] []
1724 {\def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
1725 {\ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}
```

**sboxquote**

```
1726 \newenvironment{sboxquote}[1] []
1727 {\def\@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
1728 {\@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.

```
1729 \providecommand{\@lec}[1]{(#1)}
1730 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@lec{#1}}
1731 \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.

```

1732 \addmetakey{omdoc@index}{at}
1733 \addmetakey[false]{omdoc@index}{loadmodules}[true]
1734 \newcommand\omdoc@indexi[2][\ifindex%
1735 \metasetkeys{omdoc@index}{#1}%
1736 \@bsphack\begingroup\@sanitize%
1737 \protected@write\@indexfile{}\string\indexentry%
1738 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1739 \ifx\omdoc@index@loadmodules\@true%
1740 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
1741 \else #2\fi% loadmodules
1742 }\the page}}%
1743 \endgroup\@esphack\fi}%ifindex
1744 \newcommand\omdoc@indexii[3][\ifindex%
1745 \metasetkeys{omdoc@index}{#1}%
1746 \@bsphack\begingroup\@sanitize%
1747 \protected@write\@indexfile{}\string\indexentry%
1748 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1749 \ifx\omdoc@index@loadmodules\@true%
1750 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1751 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1752 \else #2!#3\fi% loadmodules
1753 }\the page}}%
1754 \endgroup\@esphack\fi}%ifindex
1755 \newcommand\omdoc@indexiii[4][\ifindex%
1756 \metasetkeys{omdoc@index}{#1}%
1757 \@bsphack\begingroup\@sanitize%
1758 \protected@write\@indexfile{}\string\indexentry%
1759 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1760 \ifx\omdoc@index@loadmodules\@true%
1761 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1762 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1763 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1764 \else #2!#3!#4\fi% loadmodules
1765 }\the page}}%
1766 \endgroup\@esphack\fi}%ifindex
1767 \newcommand\omdoc@indexiv[5][\ifindex%
1768 \metasetkeys{omdoc@index}{#1}%
1769 \@bsphack\begingroup\@sanitize%
1770 \protected@write\@indexfile{}\string\indexentry%
1771 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%

```

```

1772 \ifx\omdoc@index@loadmodules \@true%
1773 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1774 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1775 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1776 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
1777 \else #2!#3!#4!#5\fi% loadmodules
1778 }\the page}}%
1779 \endgroup\@esphack\fi}%ifindex

```

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

`\*indi*`

```

1780 \newcommand\aindi[3] []{{#2}\omdoc@indexi[#1]{#3}}
1781 \newcommand\indi[2] []{{#2}\omdoc@indexi[#1]{#2}}
1782 \newcommand\indis[2] []{{#2}\omdoc@indexi[#1]{#2s}}
1783 \newcommand\Indi[2] []{{\captitalize{#2}}\omdoc@indexi[#1]{#2}}
1784 \newcommand\Indis[2] []{{\capitalize{#2}}\omdoc@indexi[#1]{#2s}}
1785
1786 \newcommand\@indii[3] []{\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
1787 \newcommand\aindii[4] []{{#2}\@indii[#1]{#3}{#4}}
1788 \newcommand\indii[3] []{{#2 #3}\@indii[#1]{#2}{#3}}
1789 \newcommand\indiis[3] []{{#2 #3s}\@indii[#1]{#2}{#3}}
1790 \newcommand\Indii[3] []{{\captitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1791 \newcommand\Indiis[3] []{{\capitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1792
1793 \newcommand\@indiii[4] []{\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexiii[#1]{#3}{#2 (#4)}}
1794 \newcommand\aindiii[5] []{{#2}\@indiii[#1]{#3}{#4}{#5}}
1795 \newcommand\indiii[4] []{{#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
1796 \newcommand\indiis[4] []{{#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}
1797 \newcommand\Indiii[4] []{{\captitalize{#2 #3 #4}}\@indiii[#1]{#2}{#3}{#4}}
1798 \newcommand\Indiis[4] []{{\capitalize{#2 #3 #4s}}\@indiii[#1]{#2}{#3}{#4}}
1799
1800 \newcommand\@indiv[5] []{\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
1801 \newcommand\aindiv[6] []{{#2}\@indiv[#1]{#3}{#4}{#5}{#6}}
1802 \newcommand\indiv[5] []{{#2 #3 #4 #5}\@indiv[#1]{#2}{#3}{#4}{#5}}
1803 \newcommand\indivs[5] []{{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
1804 \newcommand\Indiv[5] []{{\capitalize{#2 #3 #4 #5s}}\@indiv[#1]{#2}{#3}{#4}{#5}}
1805 \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.

```

1806 \newcommand\hateq{\ensuremath{\widehat{=}}\xspace}
1807 \newcommand\hatequiv{\ensuremath{\widehat{\equiv}}\xspace}
1808 \@ifundefined{ergo}%
1809 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1810 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1811 \newcommand{\reflectsquig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%

```

```

1812 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
1813 \newcommand\notergo{\ensuremath{\not\leadsto}}
1814 \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*`

```

1815 \newcommand\indextoo[2] [] {\indi[#1]{#2}}%
1816 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}%
1817 \newcommand\indexalt[2] [] {\aindi[#1]{#2}}%
1818 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead}%
1819 \newcommand\twintoo[3] [] {\indii[#1]{#2}{#3}}%
1820 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}%
1821 \newcommand\twinalt[3] [] {\aindii[#1]{#2}{#3}}%
1822 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead}%
1823 \newcommand\atwintoo[4] [] {\indiii[#1]{#2}{#3}{#4}}%
1824 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead}%
1825 \newcommand\atwinalt[4] [] {\aindii[#1]{#2}{#3}{#4}}%
1826 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instead}%
1827 \</package>

```

`\my*graphics`

```

1828 \newcommand\mygraphics[2] [] {\includegraphics[#1]{#2}}%
1829 \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics instead}%
1830 \newcommand\mycgraphics[2] [] {\begin{center}\mygraphics[#1]{#2}\end{center}}%
1831 \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics instead}%
1832 \newcommand\mybgraphics[2] [] {\fbox{\mygraphics[#1]{#2}}}%
1833 \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics instead}%
1834 \newcommand\mycbgraphics[2] [] {\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}%
1835 \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphics instead}%

```

## 4 Things to deprecate

Module options:

```

1836 \addmetakey*{module}{id} % TODO: deprecate properly
1837 \addmetakey*{module}{load}
1838 \addmetakey*{module}{path}
1839 \addmetakey*{module}{dir}
1840 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
1841 \addmetakey*{module}{noalign}[true]
1842
1843 \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.

```

1844 %\srefaddidkey{symdef}% what does this do?
1845 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
1846 \define@key{symdef}{noverb}[all]{}%
1847 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
1848 \define@key{symdef}{specializes}{}%
1849 \addmetakey*{symdef}{noalign}[true]
1850 \define@key{symdef}{primary}[true]{}%
1851 \define@key{symdef}{assocarg}{}%
1852 \define@key{symdef}{bvars}{}%
1853 \define@key{symdef}{bargs}{}%
1854 \addmetakey{symdef}{lang}%
1855 \addmetakey{symdef}{prec}%
1856 \addmetakey{symdef}{arity}%
1857 \addmetakey{symdef}{variant}%
1858 \addmetakey{symdef}{ns}%
1859 \addmetakey{symdef}{args}%
1860 \addmetakey{symdef}{name}%
1861 \addmetakey*{symdef}{title}%
1862 \addmetakey*{symdef}{description}%
1863 \addmetakey{symdef}{subject}%
1864 \addmetakey*{symdef}{display}%
1865 \addmetakey*{symdef}{gfc}%

```

3

EdN:3

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

```

1866 \def\symdef{\@ifnextchar[{\@symdef}{\@symdef[]}}%
1867 \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.

```

1868 \def\@@symdef[#1]#2[#3]{%
1869   \@insymdef@true%
1870   \metasetkeys{symdef}{#1}%
1871   \edef\symdef@tmp@optpars{\ifcsvoid{symdef@name}{[]}{[name=\symdef@name]}}%
1872   \expandafter\symdecl\symdef@tmp@optpars{#2}%
1873   \@insymdef@false%
1874   \notation[#1]{#2}[#3]%
1875 }% mod@show
1876 \def\symdef@type{Symbol}%
1877 \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`.

```

1878 \def\symvariant#1{%
1879   \ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}%
1880   }%
1881 \def\@symvariant#1[#2]#3#4{%
1882   \notation[#3]{#1}[#2]{#4}%
1883 \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.

```

1884 \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.

```

1885 \newif\if@importing\@importingfalse
1886 \define@key{symi}{noverb}[all]{}%
1887 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
1888 \define@key{symi}{specializes}{}%
1889 \define@key{symi}{gfc}{}%
1890 \define@key{symi}{noalign}[true]{}%
1891 \newcommand\symi{\@ifstar\@symi@star\symi}
1892 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
1893   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces}
1894 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
1895   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignorespaces}
1896 \newcommand\symii{\@ifstar\@symii@star\symii}
1897 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
1898   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespaces}
1899 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
1900   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\ignorespaces}
1901 \newcommand\symiii{\@ifstar\@symiii@star\symiii}
1902 \newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%
1903   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignorespaces}
1904 \newcommand\@symiii@star[4][]{\metasetkeys{symi}{#1}%
1905   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi\ignorespaces}
1906 \newcommand\symiv{\@ifstar\@symiv@star\symiv}
1907 \newcommand\@symiv[5][]{\metasetkeys{symi}{#1}%
1908   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces}
1909 \newcommand\@symiv@star[5][]{\metasetkeys{symi}{#1}%
1910   \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`.

```

1911 %\srefaddidkey{importmhmodule}%
1912 \addmetakey{importmhmodule}{mhrepos}%
1913 \addmetakey{importmhmodule}{path}%
1914 \addmetakey{importmhmodule}{ext}% why does this exist?
1915 \addmetakey{importmhmodule}{dir}%
1916 \addmetakey[false]{importmhmodule}{conservative}[true]%
1917 \newcommand\importmhmodule[2] [] {%
1918   \parsemodule@maybesetcodes
1919   \metasetkeys{importmhmodule}{#1}%
1920   \ifx\importmhmodule@dir\empty%
1921     \edef\@path{\importmhmodule@path}%
1922   \else\edef\@path{\importmhmodule@dir/#2}\fi%
1923   \ifx\@path\empty% if module name is not set
1924     \@importmodule[] {#2}{export}%
1925   \else%
1926     \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
1927     \ifx\importmhmodule@mhrepos\empty% if in the same repos
1928       \relax% no need to change mh@currentrepos, i.e, current directory.
1929     \else%
1930       \setcurrentreposinfo\importmhmodule@mhrepos% change it.
1931       \addto@thismodule{x}\noexpand\setcurrentreposinfo{\importmhmodule@mhrepos}}%
1932     \fi%
1933     \@importmodule[\MathHub{\mh@currentrepos/source/\@path}] {#2}{export}%
1934     \setcurrentreposinfo\mh@@repos% after importing, reset to old value
1935     \addto@thismodule{x}\noexpand\setcurrentreposinfo{\mh@@repos}}%
1936   \fi%
1937   \ignorespacesandpars%
1938 }

```

`\usemhmodule`

```

1939 \addmetakey{importmhmodule}{load}
1940 \addmetakey{importmhmodule}{id}
1941 \addmetakey{importmhmodule}{dir}
1942 \addmetakey{importmhmodule}{mhrepos}
1943
1944 \addmetakey{importmodule}{load}
1945 \addmetakey{importmodule}{id}
1946
1947 \newcommand\usemhmodule[2] [] {%
1948   \metasetkeys{importmhmodule}{#1}%
1949   \ifx\importmhmodule@dir\empty%
1950     \edef\@path{\importmhmodule@path}%
1951   \else\edef\@path{\importmhmodule@dir/#2}\fi%
1952   \ifx\@path\empty%
1953     \usemodule[id=\importmhmodule@id] {#2}%
1954   \else%
1955     \edef\mh@@repos{\mh@currentrepos}%

```

```

1956 \ifx\importmhmodule@mhrepos\@empty%
1957 \else\setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
1958 \usemodule{\@path\@QuestionMark#2}%
1959 %\usemodule[load=MathHub{\mh@currentrepos/source/\@path},
1960 %                                id=\importmhmodule@id]{#2}%
1961 \setcurrentreposinfo\mh@trepos%
1962 \fi%
1963 \ignorespacesandpars}

\mhinputref
1964 \newcommand\mhinputref[2][\%
1965 \edef\mhinputref@first{#1}%
1966 \ifx\mhinputref@first\@empty%
1967 \inputref{#2}%
1968 \else%
1969 \inputref[mhrepos=\mhinputref@first]{#2}%
1970 \fi%
1971 }

\trefi*
1972 \newcommand\trefi[2][\%
1973 \edef\trefi@mod{#1}%
1974 \ifx\trefi@mod\@empty\tref{#2}\else\tref{#1\@QuestionMark#2}\fi%
1975 }
1976 \newcommand\trefii[3][\%
1977 \edef\trefi@mod{#1}%
1978 \ifx\trefi@mod\@empty\tref{#2-#3}\else\tref{#1\@QuestionMark#2-#3}\fi%
1979 }

\defi*
1980 \def\defii#1#2{\defi{#1!#2}}
1981 \def\Defii#1#2{\Defi{#1!#2}}
1982 \def\defiis#1#2{\defis{#1!#2}}
1983 \def\Defiis#1#2{\Defis{#1!#2}}
1984 \def\defiii#1#2#3{\defi{#1!#2!#3}}
1985 \def\Defiii#1#2#3{\Defi{#1!#2!#3}}
1986 \def\defiiis#1#2#3{\defis{#1!#2!#3}}
1987 \def\Defiiis#1#2#3{\Defis{#1!#2!#3}}
1988 \def\defiv#1#2#3#4{\defi{#1!#2!#3!#4}}
1989 \def\Defiv#1#2#3#4{\Defi{#1!#2!#3!#4}}
1990 \def\defivs#1#2#3#4{\defis{#1!#2!#3!#4}}
1991 \def\Defivs#1#2#3#4{\Defis{#1!#2!#3!#4}}
1992 \def\adefi#1#2{\defi[name=#2]{#1}}
1993 \def\adefii#1#2#3{\defi[name=#2-#3]{#1}}
1994 \def\adefiii#1#2#3#4{\defi[name=#2-#3-#4]{#1}}
1995 \def\adefiv#1#2#3#4#5{\defi[name=#2-#3-#4-#5]{#1}}

1996 \newlinechar=\oldnewlinechar

```