

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

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

March 15, 2021

## **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 . . . . .	20
3.5	Symbols/Notations/Verbalizations . . . . .	31
3.6	Term References . . . . .	43
3.7	sref . . . . .	46
3.8	smultiling . . . . .	48
3.9	smglom . . . . .	48
3.10	mathhub . . . . .	49
3.11	omdoc/omgroup . . . . .	50
3.12	omtext . . . . .	52
<b>4</b>	<b>Things to deprecate</b>	<b>58</b>

# 1 Introduction

TODO

## 2 User commands

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

## 3 Implementation

```
1 <*cls>
2 \LoadClass{standalone}
3 \RequirePackage{stex}
4 </cls>
5 <*package>
6 \let\ex\expandafter
7 % TODO
8 \newif\if@stex@debugmode\@stex@debugmodefalse
9 \DeclareOption{debug}{\@stex@debugmodetrue}
10 \def\stex@debug#1{\if@stex@debugmode\message{^^J#1^^J}\fi}
11 % Modules:
12 \newif\ifmod@show\mod@showfalse
13 \DeclareOption{showmods}{\mod@showtrue}
14 % sref:
15 \newif\ifextrefs\extrefsfalse
16 \DeclareOption{extrefs}{\extrefstrue}
17 %
18 \ProcessOptions
```

A conditional for LaTeXML:

```

19 \ifcname if@latexml\endcsname\else
20 \ex\newif\cname if@latexml\endcsname\@latexmlfalse
21 \fi
22 \RequirePackage{xspace}
23 \RequirePackage{standalone}
24 \RequirePackageWithOptions{stex-metakeys}
25 \RequirePackage{xstring}
26 \RequirePackage{etoolbox}

```

### 3.1 sTeX base

The sTeX logo:

```

27 \protected\def\stex{%
28 \ifundefined{texorpdfstring}%
29 {\let\texorpdfstring\@firstoftwo}%
30 {}%
31 \texorpdfstring{\raisebox{-.5ex}{S\kern-.5ex\TeX}{sTeX}\xspace}%
32 }
33 \def\sTeX{\stex}

```

### 3.2 Paths and URIs

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

```

34 \def\pathsuris@setcatcodes{%
35 \edef\pathsuris@oldcatcode@hash{\the\catcode'\#}%
36 \catcode'\#=12\relax%
37 \edef\pathsuris@oldcatcode@slash{\the\catcode'\/%}
38 \catcode'\/=12\relax%
39 \edef\pathsuris@oldcatcode@colon{\the\catcode'\:%}
40 \catcode'\:=12\relax%
41 \edef\pathsuris@oldcatcode@qm{\the\catcode'\?}%
42 \catcode'\?=12\relax%
43 }
44 \def\pathsuris@resetcatcodes{%
45 \catcode'\#\pathsuris@oldcatcode@hash\relax%
46 \catcode'\/>\pathsuris@oldcatcode@slash\relax%
47 \catcode'\:\pathsuris@oldcatcode@colon\relax%
48 \catcode'\?\pathsuris@oldcatcode@qm\relax%
49 }

```

`\defpath` `\defpath{macro name}{base path}` defines a new macro which can take another path to form one integrated path. For example, `\MathHub` 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.

```

50 \def\namespace@read#1{%
51   \edef\namespace@read@path{#1}%
52   \edef\namespace@read@path{\ex\detokenize\ex{\namespace@read@path}}%
53   \namespace@continue%
54 }
55 \def\namespace@continue{%
56   \pathsuris@resetcatcodes%
57   \ex\edef\csname\namespace@macroname\endcsname##1{%
58     \namespace@read@path\@Slash##1%
59   }%
60 }
61 \protected\def\namespace#1{%
62   \def\namespace@macroname{#1}%
63   \pathsuris@setcatcodes%
64   \namespace@read%
65 }
66 \let\defpath\namespace

```

### 3.2.1 Path Canonicalization

We define some macros for later comparison.

```

67 \pathsuris@setcatcodes
68 \def\@ToTop{..}
69 \def\@Slash{/}
70 \def\@Colon{:}
71 \def\@Space{ }
72 \def\@QuestionMark{?}
73 \def\@Dot{.}
74 \catcode'\&=12
75 \def\@Ampersand{&}
76 \catcode'\&=4
77 \def\@Fragment{#}
78 \pathsuris@resetcatcodes
79 \catcode'\.=0
80 .catcode'\.=12
81 .let.\@BackSlash\
82 .catcode'\.=0
83 \catcode'\.=12
84 \edef\old@percent@catcode{\the\catcode'\%}
85 \catcode'\%=12
86 \let\@Percent%
87 \catcode'\%=\old@percent@catcode

```

\@cpath Canonicalizes (file) paths:

```

88 \def\@cpath#1{%
89   \edef\pathsuris@cpath@temp{#1}%
90   \def\@cpath@path{}%
91   \IfBeginWith\pathsuris@cpath@temp\@Slash{%

```

```

92     \@cpath@loop%
93     \edef\@cpath@path{\@Slash\@cpath@path}%
94 }{%
95     \IfBeginWith\pathsuris@cpath@temp{\@Dot\@Slash}{%
96         \StrGobbleLeft\pathsuris@cpath@temp2[\pathsuris@cpath@temp]%
97         \@cpath@loop%
98     }{%
99         \ifx\pathsuris@cpath@temp\@Dot\else%
100         \@cpath@loop\fi%
101     }%
102 }%
103 \IfEndWith\@cpath@path\@Slash{%
104     \ifx\@cpath@path\@Slash\else%
105         \StrGobbleRight\@cpath@path1[\@cpath@path]%
106         \fi%
107 }{}%
108 }
109
110 \def\@cpath@loop{%
111     \IfSubStr\pathsuris@cpath@temp\@Slash{%
112         \StrCut\pathsuris@cpath@temp\@Slash%
113         \pathsuris@cpath@temp@a\pathsuris@cpath@temp%
114         \ifx\pathsuris@cpath@temp@a\@ToTop%
115             \ifx\@cpath@path\@empty%
116                 \edef\@cpath@path{\@ToTop}%
117             \else%
118                 \edef\@cpath@path{\@cpath@path\@Slash\@ToTop}%
119             \fi%
120             \@cpath@loop%
121         \else%
122             \ifx\pathsuris@cpath@temp@a\@Dot%
123                 \@cpath@loop%
124             \else%
125                 \IfBeginWith\pathsuris@cpath@temp\@ToTop{%
126                     \StrBehind{\pathsuris@cpath@temp}{\@ToTop}%
127                     [\pathsuris@cpath@temp]%
128                     \IfBeginWith\pathsuris@cpath@temp\@Slash{%
129                         \edef\pathsuris@cpath@temp%
130                         {\@cpath@path\pathsuris@cpath@temp}%
131                     }{%
132                         \ifx\@cpath@path\@empty\else%
133                             \edef\pathsuris@cpath@temp%
134                             {\@cpath@path\@Slash\pathsuris@cpath@temp}%
135                         \fi%
136                     }%
137                     \def\@cpath@path{}%
138                     \@cpath@loop%
139                 }{%
140                     \ifx\@cpath@path\@empty%
141                         \edef\@cpath@path{\pathsuris@cpath@temp@a}%

```

```

142         \else%
143         \edef\@cpath@path%
144             {\@cpath@path\@Slash\pathsuris@cpath@temp@a}%
145         \fi%
146         \@cpath@loop%
147     }%
148     \fi\fi%
149 }{%
150     \ifx\@cpath@path\@empty%
151         \edef\@cpath@path{\pathsuris@cpath@temp}%
152     \else%
153         \edef\@cpath@path{\@cpath@path\@Slash\pathsuris@cpath@temp}%
154     \fi%
155 }%
156 }

```

#### Test 1:

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

```

157 \newcommand\cpath@print[1]{%
158     \@cpath{#1}%
159     \@cpath@path%
160 }

```

\path@filename

```

161 \def\path@filename#1#2{%
162     \edef\filename@oldpath{#1}%
163     \StrCount\filename@oldpath\@Slash[\filename@lastslash]%
164     \ifnum\filename@lastslash>0%
165         \StrBehind[\filename@lastslash]\filename@oldpath%
166         \@Slash[\filename@oldpath]%
167         \edef#2{\filename@oldpath}%
168     \else%
169         \edef#2{\filename@oldpath}%
170     \fi%
171 }

```

**Test 2:** Path: /foo/bar/baz.tex  
Filename: baz.tex

`\path@filename@noext`

```
172 \def\path@filename@noext#1#2{%
173     \path@filename{#1}{#2}%
174     \edef\filename@oldpath{#2}%
175     \StrCount\filename@oldpath\@Dot[\filename@lastdot]%
176     \ifnum\filename@lastdot>0%
177         \StrBefore[\filename@lastdot]\filename@oldpath%
178         \@Dot[\filename@oldpath]%
179         \edef#2{\filename@oldpath}%
180     \else%
181         \edef#2{\filename@oldpath}%
182     \fi%
183 }
```

**Test 3:** Path: /foo/bar/baz.tex  
Filename: baz

### 3.2.2 Windows

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

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

**Test 4:** We are on windows: no.

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

```
186 \newif\if@windowstopath@inpath@
187 \def\windows@to@path#1{%
188     \@windowstopath@inpath@false%
189     \def\windows@temp{}%
190     \edef\windows@path{#1}%
191     \ifx\windows@path\empty\else%
192         \ex\windows@path@loop\windows@path\windows@path@end%
193     \fi%
194     \let#1\windows@temp%
195 }
196 \def\windows@path@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{\windows@path@loop#2\windows@path@end}%
202     \fi%
```



```

203     \if@windowstopath@inpath%
204         \ifx#1\@BackSlash%
205             \edef\windows@temp{\windows@temp\@Slash}%
206         \else%
207             \edef\windows@temp{\windows@temp#1}%
208         \fi%
209     \else%
210         \ifx#1:%
211             \edef\windows@temp{\@Slash\windows@temp}%
212             \@windowstopath@inpath@true%
213         \else%
214             \edef\windows@temp{\windows@temp#1}%
215         \fi%
216     \fi%
217     \windows@continue%
218 }

```

**Test 5:** 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:

```

219 \def\path@to@windows#1{%
220     \@windowstopath@inpath@false%
221     \def\windows@temp{}%
222     \edef\windows@path{#1}%
223     \edef\windows@path{\expandafter\@gobble\windows@path}%
224     \ifx\windows@path\@empty\else%
225         \expandafter\path@windows@loop\windows@path\windows@path@end%
226     \fi%
227     \let#1\windows@temp%
228 }
229 \def\path@windows@loop#1#2\windows@path@end{%
230     \def\windows@temp@b{#2}%
231     \ifx\windows@temp@b\@empty%
232         \def\windows@continue{}%
233     \else%
234         \def\windows@continue{\path@windows@loop#2\windows@path@end}%
235     \fi%
236     \if@windowstopath@inpath%
237         \ifx#1/%
238             \edef\windows@temp{\windows@temp\@BackSlash}%
239         \else%
240             \edef\windows@temp{\windows@temp#1}%
241         \fi%
242     \else%
243         \ifx#1/%
244             \edef\windows@temp{\windows@temp:\@BackSlash}%
245             \@windowstopath@inpath@true%
246         \else%

```

```

247         \edef\windows@temp{\windows@temp#1}%
248     \fi%
249 \fi%
250 \windows@continue%
251 }

```

**Test 6:**   Input: /C/foo/bar.baz  
Output: C:\foo\bar.baz

### 3.2.3 Auxiliary methods

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

```

252 \def\path@trimstring#1{%
253     \edef\pathsuris@trim@temp{#1}%
254     \IfBeginWith\pathsuris@trim@temp\@Space{%
255         \StrGobbleLeft\pathsuris@trim@temp1[#1]%
256         \path@trimstring{#1}%
257     }{%
258         \IfEndWith\pathsuris@trim@temp\@Space{%
259             \StrGobbleRight\pathsuris@trim@temp1[#1]%
260             \path@trimstring{#1}%
261         }{%
262             \edef#1{\pathsuris@trim@temp}%
263         }%
264     }%
265 }

```

**Test 7:**   >foo bar<

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

```

266 %\if@latexml\else
267 \def\@kpsewhich#1#2{\begingroup%
268     \edef\kpsewhich@cmd{"|kpsewhich #2"%
269     \everyeof{\noexpand}%
270     \catcode'\=12%
271     \edef#1{\@input\kpsewhich@cmd\@Space}%
272     \path@trimstring#1%
273     \if@iswindows@ \windows@to@path#1\fi%
274     \xdef#1{\ex\detokenize\expandafter{#1}}%
275 \endgroup}
276 %\fi

```

**Test 8:**   /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:

```

277 \edef\pwd@cmd{\if@iswindows@ -expand-var \@Percent%

```

```

278 CD\@Percent\else -var-value PWD\fi}
279 \@kpsewhich\stex@PWD\pwd@cmd
280 \edef\stex@mainfile{\stex@PWD\@Slash\jobname}
281 \edef\stex@mainfile{\ex\detokenize\ex{\stex@mainfile}}

```

**Test 9:** [/home/jazzpirate/work/Software/ext/sTeX/sty/stex-master](#)

We keep a stack of \inputed files:

```

282 \def\stex@currfile@stack{}
283
284 \def\stex@currfile@push#1{%
285     \edef\stex@temppath{#1}%
286     \edef\stex@temppath{\ex\detokenize\ex{\stex@temppath}}%
287     \edef\stex@currfile@stack{\stex@currfile%
288         \ifx\stex@currfile@stack\@empty\else,\stex@currfile@stack\fi}
289     \IfBeginWith\stex@temppath\@Slash{\@cpath{\stex@temppath}}{%
290         \@cpath{\stex@PWD\@Slash#1}%
291     }
292     \let\stex@currfile\@cpath@path%
293     \path@filename\stex@currfile\stex@currfilename%
294     \StrLen\stex@currfilename[\stex@currfile@tmp]%
295     \StrGobbleRight\stex@currfile{\the\numexpr%
296         \stex@currfile@tmp+1 }[\stex@currpath]%
297     \global\let\stex@currfile\stex@currfile%
298     \global\let\stex@currpath\stex@currpath%
299     \global\let\stex@currfilename\stex@currfilename%
300 }
301 \def\stex@currfile@pop{%
302     \ifx\stex@currfile@stack\@empty%
303         \global\let\stex@currfile\stex@mainfile%
304         \global\let\stex@currpath\stex@PWD%
305         \global\let\stex@currfilename\jobname%
306     \else%
307         \StrCut\stex@currfile@stack,\stex@currfile\stex@currfile@stack%
308         \path@filename\stex@currfile\stex@currfilename%
309         \StrLen\stex@currfilename[\stex@currfile@tmp]%
310         \StrGobbleRight\stex@currfile{\the\numexpr%
311             \stex@currfile@tmp+1 }[\stex@currpath]%
312         \global\let\stex@currfile\stex@currfile%
313         \global\let\stex@currpath\stex@currpath%
314         \global\let\stex@currfilename\stex@currfilename%
315     \fi%
316 }

```

**\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:

```

317 \def\stexinput#1{%
318     \stex@iffileexists{#1}{%
319         \stex@currfile@push\stex@temp@path%
320         \input{\stex@currfile}%

```

```

321     \stex@currfile@pop%
322   }%
323   {%
324     \PackageError{stex}{File does not exist %
325       (#1): \stex@temp@path}{}%
326   }%
327 }
328 \def\stex@iffileexists#1#2#3{%
329   \edef\stex@temp@path{#1}%
330   \if@iswindows@\path@to@windows\stex@temp@path\fi%
331   \IfFileExists\stex@temp@path{#2}{#3}%
332 }
333 \stex@currfile@pop

```

**Test 10:** This file: /home/jazzpirate/work/Software/ext/sTeX/sty/stex-master/stex  
A test file: /home/jazzpirate/work/Software/ext/sTeX/sty/stex-master/testfile.tex  
Back: /home/jazzpirate/work/Software/ext/sTeX/sty/stex-master/stex

### 3.2.5 MathHub repositories

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

```

334 \@kpsewhich\mathhub@path{--var-value MATHHUB}
335 \if@iswindows@\windows@to@path\mathhub@path\fi
336 \ifx\mathhub@path\empty
337   \PackageWarning{stex}{MATHHUB system variable not %
338     found or wrongly set}{%
339     \defpath{MathHub}{%
340     \else\defpath{MathHub}\mathhub@path\fi

```

**Test 11:** /home/jazzpirate/work/MathHub

\mathhub@findmanifest \mathhub@findmanifest{<path>} searches for a file MANIFEST.MF up and over  
<path> in the file system tree.

```

341 \def\mathhub@findmanifest#1{%
342   \@cpath{#1}%
343   \ifx\@cpath@path\@Slash%
344     \def\manifest@mf{%
345   \else\ifx\@cpath@path\@empty%
346     \def\manifest@mf{%
347   \else%
348     \edef\@findmanifest@path{\@cpath@path/MANIFEST.MF}%
349     \if@iswindows@\path@to@windows\@findmanifest@path\fi%
350     \IfFileExists{\@findmanifest@path}{%
351       \edef\manifest@mf{\@findmanifest@path}%
352       \xdef\temp@archive@dir{\ex\detokenize\ex{\@cpath@path}}%
353     }{%
354     \edef\@findmanifest@path{\@cpath@path/META-INF/MANIFEST.MF}%
355     \if@iswindows@\path@to@windows\@findmanifest@path\fi%

```

```

356 \IfFileExists{\@findmanifest@path}{%
357 \edef\manifest@mf{\@findmanifest@path}%
358 \xdef\temp@archive@dir{\ex\detokenize\ex{\@cpath@path}}%
359 }{%
360 \edef\@findmanifest@path{\@cpath@path/meta-inf/MANIFEST.MF}%
361 \if@iswindows@path@to@windows\@findmanifest@path\fi%
362 \IfFileExists{\@findmanifest@path}{%
363 \edef\manifest@mf{\@findmanifest@path}%
364 \xdef\temp@archive@dir{\ex\detokenize\ex{\@cpath@path}}%
365 }{%
366 \mathhub@findmanifest{\@cpath@path/..}%
367 }}%
368 \fi\fi%
369 }

```

**Test 12:** In `/home/jazzpirate/work/MathHub/smgglom/mv/source:/home/jazzpirate/work/MathHub/smgglom/mv/META-INF/MANIFEST.MF`

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

```

370 \def\split@manifest@key{%
371 \IfSubStr{\manifest@line}{\@Colon}{%
372 \StrBefore{\manifest@line}{\@Colon}[\manifest@key]%
373 \StrBehind{\manifest@line}{\@Colon}[\manifest@line]%
374 \path@trimstring\manifest@line%
375 \path@trimstring\manifest@key%
376 }{%
377 \def\manifest@key{}%
378 }%
379 }

```

the next helper function iterates over lines in MANIFEST.MF

```

380 \def\parse@manifest@loop{%
381 \ifeof\@manifest%
382 \else%
383 \read\@manifest to \manifest@line\relax%
384 \split@manifest@key%
385 % id
386 \IfStrEq\manifest@key{id}{%
387 \xdef\manifest@mf{id}\manifest@line}%
388 }{%
389 % narration-base
390 \IfStrEq\manifest@key{narration-base}{%
391 \xdef\manifest@mf@narr{\manifest@line}%
392 }{%
393 % namespace
394 \IfStrEq\manifest@key{source-base}{%
395 \xdef\manifest@mf@ns{\manifest@line}%
396 }{%
397 \IfStrEq\manifest@key{ns}{%

```

```

398     \xdef\manifest@mf@ns{\manifest@line}%
399   }{%
400   % dependencies
401   \IfStrEq\manifest@key{dependencies}{%
402     \xdef\manifest@mf@deps{\manifest@line}%
403   }{%
404   }}}}
405   \parse@manifest@loop%
406 \fi%
407 }

```

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

```

408 \newread\@manifest
409 \def\mathhub@parsemanifest#1#2{%
410   \gdef\temp@archive@dir{}%
411   \mathhub@findmanifest{#2}%
412   \begingroup%
413     \newlinechar=-1%
414     \endlinechar=-1%
415     \gdef\manifest@mf@id{}%
416     \gdef\manifest@mf@narr{}%
417     \gdef\manifest@mf@ns{}%
418     \gdef\manifest@mf@deps{}%
419     \immediate\openin\@manifest=\manifest@mf\relax%
420     \parse@manifest@loop%
421     \immediate\closein\@manifest%
422   \endgroup%
423   \if@iswindows@\windows@to@path\manifest@mf\fi%
424   \cslet{#1id}\manifest@mf@id%
425   \cslet{#1narr}\manifest@mf@narr%
426   \cslet{#1ns}\manifest@mf@ns%
427   \cslet{#1deps}\manifest@mf@deps%
428   \ifcsvoid\manifest@mf@id{}{%
429     \cslet{#1dir}\temp@archive@dir%
430   }%
431 }

```

**Test 13:**     `id`: FOO/BAR  
`ns`: <http://mathhub.info/FOO/BAR>  
`dir`: FOO

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

```

432 \def\mathhub@setcurrentreposinfo#1{%
433   \edef\mh@currentrepos{#1}%
434   \ifx\mh@currentrepos\@empty%

```

```

435 \edef\currentrepos@dir{\@Dot}%
436 \def\currentrepos@narr{%
437 \def\currentrepos@ns{%
438 \def\currentrepos@id{%
439 \def\currentrepos@deps{%
440 \else%
441 \ifcsdef{mathhub@dir@\mh@currentrepos}{%
442 \inmhreposttrue
443 \ex\let\ex\currentrepos@dir\csname mathhub@dir@#1\endcsname%
444 \ex\let\ex\currentrepos@narr\csname mathhub@narr@#1\endcsname%
445 \ex\let\ex\currentrepos@ns\csname mathhub@ns@#1\endcsname%
446 \ex\let\ex\currentrepos@deps\csname mathhub@deps@#1\endcsname%
447 }{%
448 \mathhub@parsemanifest{currentrepos@}{\MathHub{#1}}%
449 \@setcurrentreposinfo%
450 \ifcsvoid{currentrepos@dir}{\PackageError{stex}{No archive with %
451 name #1 found!}{make sure that #1 is directly in your MATHHUB folder %
452 and contains a MANIFEST.MF, either directly in #1 or in a meta-inf %
453 subfolder.}}{\inmhreposttrue}%
454 }%
455 \fi%
456 }
457
458 \def\@setcurrentreposinfo{%
459 \edef\mh@currentrepos{\currentrepos@id}%
460 \ifcsvoid{currentrepos@dir}{%
461 \csxdef{mathhub@dir@\currentrepos@id}{\currentrepos@dir}%
462 \csxdef{mathhub@narr@\currentrepos@id}{\currentrepos@narr}%
463 \csxdef{mathhub@ns@\currentrepos@id}{\currentrepos@ns}%
464 \csxdef{mathhub@deps@\currentrepos@id}{\currentrepos@deps}%
465 }%
466 }

```

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

```

467 \newif\if@inmhrepos\@inmhreposfalse
468 \ifcsvoid{stex@PWD}{%
469 \mathhub@parsemanifest{currentrepos@}\stex@PWD
470 \@setcurrentreposinfo
471 \ifcsvoid{currentrepos@dir}{\message{sTeX: Not currently in a MathHub repository}}{%
472 \message{Current sTeX repository: \mh@currentrepos}
473 }
474 }

```

### 3.3 Modules

```

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

```

Aux:

```

476 %\def\ignorespacesandpars{\begingroup\catcode13=10%
477 % \@ifnextchar\relax{\endgroup}{\endgroup}}

```

and more adapted from <http://tex.stackexchange.com/questions/179016/ignore-spaces-and-pars-after-an-environment>

```

478 %\def\ignorespacesandparsafterend#1\ignorespaces\fi{#1%
479 % \fi\ignorespacesandpars}
480 %\def\ignorespacesandpars{\ifhmode\unskip\fi\@ifnextchar\par%
481 % {\ex\ignorespacesandpars\@gobble}{}}

```

Options for the module-environment:

```

482 \addmetakey*{module}{title}
483 \addmetakey*{module}{name}
484 \addmetakey*{module}{creators}
485 \addmetakey*{module}{contributors}
486 \addmetakey*{module}{srccite}
487 \addmetakey*{module}{ns}
488 \addmetakey*{module}{narr}

```

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

```

489 \ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%
490 \newrobustcmd\module@heading{%
491   \stepcounter{module}%
492   \ifmod@show%
493     \noindent{\textbf{Module} \thesection.\thetitle [\module@name}}%
494     \sref@label{id{Module \thesection.\thetitle [\module@name}}%
495     \ifx\module@title\@empty : \quad\else\quad(\module@title)\hfill\\fi%
496   \fi%
497 }%

```

#### Test 14: 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.

```

498 \newenvironment{module}[1] [] {%
499   \begin{@module}[#1]%
500   \module@heading% make the headings
501   %\ignorespacesandpars
502   \parsemodule@maybesetcodes}%
503   \end{@module}%
504   \ignorespacesafterend%
505 }%
506 \ifmod@show\surroundwithmdframed{module@om@common}\fi%

```

Some auxiliary methods:

```

507 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}
508 \def\addto@thismodule#1{%
509   \ifundefined{this@module}{-}{%
510     \expandafter\g@addto@macro@safe\this@module{#1}%
511   }%
512 }
513 \def\addto@thismoduleex#1{%
514   \ifundefined{this@module}{-}{%

```



```

515 \edef\addto@thismodule@exp{#1}%
516 \expandafter\expandafter\expandafter\g@addto@macro@safe%
517 \expandafter\this@module\expandafter{\addto@thismodule@exp}%
518 }}

```

**@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`.

```

519 \newif\ifarchive@ns@empty@ \archive@ns@empty@false
520 \def\set@default@ns{%
521   \edef\@module@ns@temp{\stex@currpath}%
522   \if@iswindows@ \windows@to@path \@module@ns@temp\fi%
523   \archive@ns@empty@false%
524   \stex@debug{Generate new namespace^^J Filepath: \@module@ns@temp}%
525   \ifcvoid{mh@currentrepos}{\archive@ns@empty@true}%
526   {\ex\ifx\csname mathhub@ns@\mh@currentrepos\endcsname\@empty\archive@ns@empty@true\fi%
527   }%
528   \stex@debug{ \ifarchive@ns@empty@ Namespace empty\else Namespace not empty\fi}%
529   \ifarchive@ns@empty@%
530     \edef\@module@ns@tempuri{file\@Colon\@Slash\@Slash\@module@ns@temp}%
531   \else%
532     \edef\@module@filepath@temppath{\@module@ns@temp}%
533     \edef\@module@ns@tempuri{\csname mathhub@ns@\mh@currentrepos\endcsname}%
534     \edef\@module@archivedirpath{\csname mathhub@dir@\mh@currentrepos\endcsname\@Slash source}%
535     \edef\@module@archivedirpath{\ex\detokenize\ex{\@module@archivedirpath}}%
536     \IfBeginWith\@module@filepath@temppath\@module@archivedirpath{%
537       \StrLen\@module@archivedirpath[\ns@temp@length]%
538       \StrGobbleLeft\@module@filepath@temppath\@module@archivedirpath[\@module@filepath@temprest]%
539       \edef\@module@ns@tempuri{\@module@ns@tempuri\@module@filepath@temprest}%
540     }{}%
541   \fi%
542   \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]}%
543   \setkeys{module}{ns=\@module@ns@tempuri}%
544 }

```

**Test 15:** <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 via the filename, e.g. `stex`, `stex1`, etc.

```

545 \def\set@next@moduleid{%
546   \path@filename@noext\stex@currfile\stex@next@moduleid@filename%

```

```

547 \edef\set@nextmoduleid@csname{namespace@\module@ns\@QuestionMark\stex@next@moduleid@filename
548 \unless\ifcsname\set@nextmoduleid@csname\endcsname%
549 \csgdef{\set@nextmoduleid@csname}{0}%
550 \fi%
551 \edef\namespace@currrnum{\csname\set@nextmoduleid@csname\endcsname}%
552 \edef\module@temp@setidname{\noexpand\setkeys{module}{name=%
553 \stex@next@moduleid@filename\ex\unless\ex\ifnum\csname\set@nextmoduleid@csname\endcsname=0.
554 \module@temp@setidname%
555 \csxdef{\set@nextmoduleid@csname}{\the\numexpr\namespace@currrnum+1}%
556 }

```

## Test 16: stex

### stex.1

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
- `\mathop{\langle uri \rangle}` that expands to `\invoke@module{\mathop{\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.

```

557 \newenvironment{@module}[1][{}]{%
558 \metasetkeys{module}{#1}%
559 \ifcsvoid{module@name}{\let\module@name\module@id}{}% % TODO deprecate
560 \ifcsvoid{module@name}{\set@next@moduleid}{}%
561 \let\module@id\module@name% % TODO deprecate
562 \ifcsvoid{currentmodule@uri}{%
563 \ifx\module@ns\@empty\set@default@ns\fi%
564 \ifx\module@narr\@empty%
565 \setkeys{module}{narr=\module@ns}%
566 \fi%

```

```

567 }{
568   \if@smsmode%
569     \ifx\module@ns\@empty\set@default@ns\fi%
570     \ifx\module@narr\@empty%
571       \setkeys{module}{narr=\module@ns}%
572     \fi%
573   \else%
574     % Nested Module:
575     \stex@debug{Nested module! Parent: \currentmodule@uri}%
576     \setkeys{module}{name=\currentmodule@name\@Slash\module@name}%
577     \let\module@id\module@name % TODO deprecate
578     \setkeys{module}{ns=\currentmodule@ns}%
579   \fi%
580 }%
581 \edef\module@uri{\module@ns\@QuestionMark\module@name}%
582 \csgdef{module@names@\module@uri}{}%
583 \csgdef{module@imports@\module@uri}{}%
584 \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
585 \ifcsvoid{stex@module@\module@name}{
586   \ex\global\ex\let\csname stex@module@\module@name\ex\endcsname\csname\module@uri\endcsname%
587 }{
588   \ex\edef\csname stex@module@\module@name\endcsname{\detokenize{ambiguous}}
589 }
590 \edef\this@module{%
591   \ex\noexpand\csname module@defs@\module@uri\endcsname%
592 }%
593 \ex\xdef\csname stex@lastmodule@\module@name\endcsname{\module@uri}%
594 \csdef{module@defs@\module@uri}{}%
595 \ifcsvoid{mh@currentrepos}{}{%
596   \@inmhrepostrue%
597   \addto@thismodulex{\ex\edef\ex\noexpand\csname mh@old@repos@\module@uri\endcsname%
598     {\noexpand\mh@currentrepos}}%
599   \addto@thismodulex{\noexpand\mathhub@setcurrentreposinfo{\mh@currentrepos}}%
600 }%
601 \let\currentmodule@name\module@name%
602 \let\currentmodule@ns\module@ns%
603 \let\currentmodule@uri\module@uri%
604 \stex@debug{^^JNew module: \module@uri^^J}%
605 \parsemodule@maybesetcodes%
606 \begin{latexml@module}{\module@uri}%
607 }{%
608   \end{latexml@module}%
609   \if@inmhrepos%
610     \@inmhreposfalse%
611     \addto@thismodulex{\noexpand\mathhub@setcurrentreposinfo{\expandafter\noexpand\csname mh@old@
612   \fi%
613 }%
614 % For LaTeXML bindings
615 \newenvironment{latexml@module}[1]{}{}

```

**Test 17:**    **Module 3.2**[Foo]:    Name: Foo

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

this@module: »macro:->«

**Test 18:**    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  
}\mathhub@setcurrentreposinfo {Foo/Bar}«

**Test 19:**    Removing the \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:->«

**Test 20:**    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  
}\mathhub@setcurrentreposinfo {Foo/Bar}«

A module with URI  $\langle uri \rangle$  and id  $\langle id \rangle$  creates two macros  $\backslash\langle uri \rangle$  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.

```
616 \def\@URI{uri} % TODO check this
617 \def\@invoke@module#1#2{%
618   \ifx\@URI#2%
619     #1%
620   \else%
621     % TODO something else
622     #2%
623   \fi%
624 }
```

## 3.4 Inheritance

### 3.4.1 Selective Inclusion

The next great goal is to establish the `\requiremodules` macro, which reads an  $\text{\LaTeX}$  file and processes all the module signature information in them, but does not produce any output. This is a tricky business, as we need to “parse” the modules

and treat the module signature macros specially (we refer to this as “**sms mode**”, since it is equivalent to what the – now deprecated – **sms** utility did).

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

```
\parsemodule@allow* The first step is setting up a functionality for registering \TeX macros and envi-
                    ronments as part of a module signature.
625 \newif\if@smsmode\@smsmodefalse
626 \def\parsemodule@allow#1{%
627   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#1\endcsname}%
628 }
629 \def\parsemodule@allowenv#1{%
630   \ex\def\csname parsemodule@allowedenv@#1\endcsname{#1}%
631 }
632 \def\parsemodule@replacemacro#1#2{%
633   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#2\endcsname}%
634 }
635 \def\parsemodule@replaceenv#1#2{%
636   \ex\def\csname parsemodule@allowedenv@#1\endcsname{#2}%
637 }
638 \def\parsemodule@escapechar@beginstring{begin}
639 \def\parsemodule@escapechar@endstring{end}
```

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

```
640 \parsemodule@allow{symdef}
641 \parsemodule@allow{abbrdef}
642 \parsemodule@allow{importmodule}
643 \parsemodule@allowenv{module}
644 \parsemodule@allowenv{@module}
645 \parsemodule@allow{importmhmodule}
646 \parsemodule@allow{gimport}
647 \parsemodule@allowenv{modsig}
648 \parsemodule@allowenv{mhmodsig}
649 \parsemodule@allowenv{mhmodnl}
650 \parsemodule@allowenv{modnl}
651 \parsemodule@allow{symvariant}
652 \parsemodule@allow{symi}
653 \parsemodule@allow{symii}
654 \parsemodule@allow{symiii}
655 \parsemodule@allow{symiv}
656 \parsemodule@allow{notation}
657 \parsemodule@allow{symdecl}
658
659 % to deprecate:
660
661 \parsemodule@allow{defi}
662 \parsemodule@allow{defii}
663 \parsemodule@allow{defiii}
```

```

664 \parsemodule@allow{defiv}
665 \parsemodule@allow{adefii}
666 \parsemodule@allow{adefiii}
667 \parsemodule@allow{adefiiii}
668 \parsemodule@allow{adefiv}
669 \parsemodule@allow{defis}
670 \parsemodule@allow{defiis}
671 \parsemodule@allow{defiiis}
672 \parsemodule@allow{defivs}
673 \parsemodule@allow{Defi}
674 \parsemodule@allow{Defii}
675 \parsemodule@allow{Defiii}
676 \parsemodule@allow{Defiv}
677 \parsemodule@allow{Defis}
678 \parsemodule@allow{Defiis}
679 \parsemodule@allow{Defiiis}
680 \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*).

```

681 \catcode'\.=0
682 .catcode'\.=13
683 .def.\@active@slash{\}
684 .catcode'\.<=1
685 .catcode'\.>=2
686 .catcode'\.=12
687 .catcode'\.=12
688 .def.\@open@brace<{>
689 .def.\@close@brace<}>
690 .catcode'\.=0
691 \catcode'\.=12
692 \catcode'\{=1
693 \catcode'\}=2
694 \catcode'\<=12
695 \catcode'\>=12

```

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

`\set@parsemodule@catcodes`

```

696 \def\parsemodule@ignorepackageerrors{,inputenc,}
697 \let\parsemodule@old@PackageError\PackageError
698 \def\parsemodule@packageerror#1#2#3{%
699   \IfSubStr\parsemodule@ignorepackageerrors{,#1,}{\}%
700   \parsemodule@old@PackageError{#1}{#2}{#3}%

```

```

701     }%
702 }
703 \def\set@parsemodule@catcodes{%
704     \ifcat'\=0%
705     \global\catcode'\=13%
706     \global\catcode'\#=12%
707     \global\catcode'\{=12%
708     \global\catcode'\}=12%
709     \global\catcode'\$=12%$
710     \global\catcode'\^=12%
711     \global\catcode'\_ =12%
712     \global\catcode'\&=12%
713     \ex\global\ex\let\@active@slash\parsemodule@escapechar%
714     \global\let\parsemodule@old@PackageError\PackageError%
715     \global\let\PackageError\parsemodule@packageerror%
716     \fi%
717 }

```

`\reset@parsemodule@catcodes`

```

718 \def\reset@parsemodule@catcodes{%
719     \ifcat'\=13%
720     \global\catcode'\=0%
721     \global\catcode'\#=6%
722     \global\catcode'\{=1%
723     \global\catcode'\}=2%
724     \global\catcode'\$=3%$
725     \global\catcode'\^=7%
726     \global\catcode'\_ =8%
727     \global\catcode'\&=4%
728     \global\let\PackageError\parsemodule@old@PackageError%
729     \fi%
730 }

```

`\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.

```

731 \def\parsemodule@maybesetcodes{%
732     \if@smsmode\set@parsemodule@catcodes\fi%
733 }

```

`\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:

```

734
735 \def\parsemodule@escapechar{%
736   \def\parsemodule@escape@currcls{%
737     \parsemodule@escape@parse@nextchar@%
738 }%

```

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`.

```

739 \long\def\parsemodule@escape@parse@nextchar@#1{%
740   \ifcat a#1\relax%
741     \edef\parsemodule@escape@currcls{\parsemodule@escape@currcls#1}%
742     \let\parsemodule@do@next\parsemodule@escape@parse@nextchar@%
743   \else%
744     \def\parsemodule@last@char{#1}%
745     \ifx\parsemodule@escape@currcls\@empty%
746       \def\parsemodule@do@next{%
747     \else%
748       \def\parsemodule@do@next{\parsemodule@escapechar@checkcls}%
749     \fi%
750   \fi%
751   \parsemodule@do@next%
752 }

```

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.

```

753 \def\parsemodule@escapechar@checkcls{%
754   \ifx\parsemodule@escape@currcls\parsemodule@escapechar@beginstring%
755     \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@last@char}%
756   \else%
757     \ifx\parsemodule@escape@currcls\parsemodule@escapechar@endstring%
758       \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@last@char}%
759     \else%
760       \ifcvoid\parsemodule@allowedmacro@\parsemodule@escape@currcls{%
761         \def\parsemodule@do@next{\relax\parsemodule@last@char}%
762       }{%
763         \ifx\parsemodule@last@char\@open@brace%
764           \ex\let\ex\parsemodule@do@next@ii\cename parsemodule@allowedmacro@\parsemodule@last@char%
765           \edef\parsemodule@do@next{\noexpand\parsemodule@converttoproperbraces\@open@brace%

```



```

766             \else%
767             \reset@parsemodule@catcodes%
768             \edef\parsemodule@do@next{\ex\noexpand\csname parsemodule@allowedmacro@\parsemo
769             \fi%
770             }%
771         \fi%
772     \fi%
773     \parsemodule@do@next%
774 }

```

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.

```

775 \ex\ex\ex\def%
776 \ex\ex\ex\parsemodule@converttoproperbraces%
777 \ex\@open@brace\ex#\ex1\@close@brace{%
778     \reset@parsemodule@catcodes%
779     \parsemodule@do@next@ii{#1}%
780 }

```

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.

```

781 \ex\ex\ex\def%
782 \ex\ex\ex\parsemodule@escapechar@checkbeginenv%
783 \ex\@open@brace\ex#\ex1\@close@brace{%
784     \ifcsvoid{parsemodule@allowedenv@#1}{%
785         \def\parsemodule@do@next{#1}%
786     }{%
787         \reset@parsemodule@catcodes%
788         \edef\parsemodule@envname{\csname parsemodule@allowedenv@#1\endcsname}%
789         \ex\def\ex\parsemodule@do@next\ex{%
790             \ex\begin\ex{\parsemodule@envname}%
791             }%
792         }%
793         \parsemodule@do@next%
794     }
795 \ex\ex\ex\def%
796 \ex\ex\ex\parsemodule@escapechar@checkendenv%
797 \ex\@open@brace\ex#\ex1\@close@brace{%
798     \ifcsvoid{parsemodule@allowedenv@#1}{%
799         \def\parsemodule@do@next{#1}%
800     }{%
801         \edef\parsemodule@envname{\csname parsemodule@allowedenv@#1\endcsname}%
802         \ex\def\ex\parsemodule@do@next\ex{%

```

```

803      \ex\end\ex{\parsemodule@envname}%
804    }%
805  }%
806  \parsemodule@do@next%
807 }

```

`\@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.

```

808 \newrobustcmd\@requiremodules[1]{%
809   \if@tempswa\requiremodules{#1}\fi%
810 }%

```

`\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.

```

811 \newrobustcmd\requiremodules[1]{%
812   \mod@showfalse%
813   \edef\mod@path{#1}%
814   \edef\mod@path{\expandafter\detokenize\expandafter{\mod@path}}%
815   \requiremodules@smsmode{#1}%
816 }%

```

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

```

817 \newbox\modules@import@tempbox
818 \def\requiremodules@smsmode#1{%
819   \setbox\modules@import@tempbox\vbox{%
820     \@smsmodetrue%
821     \set@parsemodule@catcodes%
822     \hbadness=100000\relax%
823     \hfuzz=10000pt\relax%
824     \vbadness=100000\relax%
825     \vfuzz=10000pt\relax%
826     \stexinput{#1.tex}%
827     \reset@parsemodule@catcodes%
828   }%
829   \parsemodule@maybesetcodes%
830 }

```

**Test 21:** parsing F00/testmodule.tex

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

### 3.4.2 importmodule

`\importmodule@bookkeeping`

```

831 \newif\if@importmodule@switchrepos\@importmodule@switchreposfalse
832 \def\importmodule@bookkeeping#1#2#3{%
833   \@importmodule@switchreposfalse%
834   \stex@debug{Importmodule: #1^^J #2^^J\detokenize{#3}}%
835   \metasetkeys{importmodule}{#1}%
836   \ifcvoid{importmodule@mhrepos}{%
837     \ifcvoid{currentrepos@dir}{%
838       \stex@debug{Importmodule: Set importmodule@dir to \stex@PWD}%
839       \let\importmodule@dir\stex@PWD%
840     }{%
841       \stex@debug{Importmodule: Set importmodule@dir to \currentrepos@dir\@Slash source}%
842       \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
843     }%
844   }{%
845     \@importmodule@switchrepostrue%
846     \stex@debug{Importmodule: Repository switch to \importmodule@mhrepos}%
847     \stex@debug{Importmodule: Current repos: \mh@currentrepos}%
848     \ex\let\csname importmodule@oldrepos@#2\endcsname\mh@currentrepos%
849     \mathhub@setcurrentreposinfo\importmodule@mhrepos%
850     \stex@debug{Importmodule: New repos: \mh@currentrepos^^J Namespace: \currentrepos@ns}%
851     \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
852   }%
853   \StrCut{#2}\@QuestionMark\importmodule@subdir\importmodule@modulename%
854   \ifx\importmodule@modulename\empty%
855     \let\importmodule@modulename\importmodule@subdir%
856     \let\importmodule@subdir\empty%
857   \else%
858     \ifx\importmodule@subdir\empty\else%
859       \edef\importmodule@dir{\importmodule@dir\@Slash\importmodule@subdir}%
860     \fi%
861   \fi%
862   #3%
863   \if@importmodule@switchrepos%
864     \ex\mathhub@setcurrentreposinfo\csname importmodule@oldrepos@#2\endcsname%
865     \stex@debug{Importmodule: switched back to: \mh@currentrepos}%
866   \fi%
867   %\ignorespacesandpars%
868 }

```

\importmodule

```

869 %\srefaddidkey{importmodule}
870 \addmetakey{importmodule}{mhrepos}
871 \newcommand\importmodule[2][\@importmodule{#1}{#2}{export}]
872 \newcommand\@importmodule[3][\@importmodule[2][\@importmodule{#1}{#2}{export}]]
873 \importmodule@bookkeeping{#1}{#2}{%
874   \@importmodule[\importmodule@dir]\importmodule@modulename{#3}%
875 }%
876 }

```

\@importmodule \@importmodule[\<filepath>]{\<mod>}{\<export?>} loads \<filepath>.tex and acti-

vates the module  $\langle mod \rangle$ . If  $\langle export? \rangle$  is `export`, then it also re-exports the `\symdefs` from  $\langle mod \rangle$ .

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

```

877 \newcommand\@importmodule[3][\{%
878   {%
879     \edef\@load{#1}%
880     \edef\@importmodule@name{#2}%
881     \stex@debug{Loading #1}%
882     \if@smsmode\else\ifcsvoid{stex@module@\@importmodule@name}{% TODO check this
883       \stex@iffileexists\@load{
884         \stex@debug{Exists: #1}%
885         \requiremodules\@load}{%
886         \stex@debug{Does not exist: #1^^JTrying \@load\@Slash\@importmodule@name}%
887         \requiremodules{\@load\@Slash\@importmodule@name}%
888       }%
889     }\fi%
890     \ifx\@load\@empty\else%
891       {% TODO
892       %   \edef\@path{\csname module@#2@path\endcsname}%
893       %   \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do not
894       %   {\PackageError{stex}% else signal an error
895       %     {Module Name Clash\MessageBreak%
896       %       A module with name #2 was already loaded under the path "\@path"\MessageBreak%
897       %       The imported path "\@load" is probably a different module with the\MessageBreak%
898       %       same name; this is dangerous -- not importing}%
899       %     {Check whether the Module name is correct}%
900       %   }%
901     }%
902     \fi%
903     \global\let\@importmodule@load\@load%
904   }%
905   \edef\@export{#3}\def\@@export{export}%prepare comparison
906   %\ifx\@export\@@export\export@defs{#2}\fi% export the module
907   \ifx\@export\@@export\addto@thismodulex{%
908     \noexpand\@importmodule[\@importmodule@load]{#2}{noexport}%
909   }%
910   \if@smsmode\else
911   \ifcsvoid{this@module}{\fi%
912     \ifcsvoid{module@imports@\module@uri}{
913       \csxdef{module@imports@\module@uri}{%
914         \csname stex@module@#2\endcsname\@URI% TODO check this
915       }%
916     }%
917     \csxdef{module@imports@\module@uri}{%
918       \csname stex@module@#2\endcsname\@URI,% TODO check this

```

```

919         \csname module@imports@\module@uri\endcsname%
920     }%
921 }%
922 }%
923 \fi\fi%
924 \if@smsmode\else%
925     \edef\activate@module@name{#2}%
926     \StrCount\activate@module@name\@Slash[\activate@module@lastslash]%
927     \ifnum\activate@module@lastslash>0%
928         \StrCut[\activate@module@lastslash]\activate@module@name\@Slash\activate@module@temp\activa
929     \fi%
930     \ifcsvoid{stex@lastmodule@\activate@module@name}{%
931         \PackageError{stex}{No module with name \activate@module@name found}{}%
932     }{%
933         \ex\ex\ex\activate@defs\ex\ex\ex{\csname stex@lastmodule@\activate@module@name\endcsname}%
934     }%
935 \fi% activate the module
936 }%

```

**Test 22:** `\importmodule {testmoduleimporta}:`  
`>macro:->\@invoke@module {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master`  
`>macro:->\@invoke@symbol {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master`

**Test 23:** `\importmodule {testmoduleimportb?importb}:`  
`>macro:->\@invoke@module {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master`  
`>macro:->\@invoke@symbol {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master`

**Test 24:** `>macro:->\edef \mh@old@repos@http://mathhub.info/FoMID/Core/foundations/types?`  
`{\mh@currentrepos }\mathhub@setcurrentreposinfo {FoMID/Core}\ifcsvoid`  
`{stex@symbol@type}{\edef \stex@symbol@type {http://mathhub.info/FoMID/Core/foundations/t`  
`\stex@symbol@type {ambiguous}}\def \http://mathhub.info/FoMID/Core/foundations/types?typ`  
`{\@invoke@symbol {http://mathhub.info/FoMID/Core/foundations/types?type.en?type}}\def`  
`\type {\@invoke@symbol {http://mathhub.info/FoMID/Core/foundations/types?type.en?type}}\`  
`{stex@symbol@hastype}{\edef \stex@symbol@hastype {http://mathhub.info/FoMID/Core/foundat`  
`\stex@symbol@hastype {ambiguous}}\def \http://mathhub.info/FoMID/Core/foundations/types?`  
`{\@invoke@symbol {http://mathhub.info/FoMID/Core/foundations/types?type.en?hastype}}\def`  
`\hastype {\@invoke@symbol {http://mathhub.info/FoMID/Core/foundations/types?type.en?hast`  
`{\mh@old@repos@http://mathhub.info/FoMID/Core/foundations/types?type.en`  
`}<`  
`>macro:->\@invoke@symbol {http://mathhub.info/FoMID/Core/foundations/types?type.en?type}`

Default document module:

```

937 \AtBeginDocument{%
938     \set@default@ns%
939     \ifx\module@narr\@empty\setkeys{module}{narr=\module@ns}\fi%
940     \let\module@name\jobname%
941     \let\module@id\module@name % TODO deprecate

```

```

942 \edef\module@uri{\module@ns\@QuestionMark\module@name}%
943 \csgdef{module@names@\module@uri}{}%
944 \csgdef{module@imports@\module@uri}{}%
945 \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
946 \expandafter\global\expandafter\let\csname stex@module@\module@name\expandafter\endcsname\csname
947 \edef\this@module{%
948   \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
949 }%
950 \csdef{module@defs@\module@uri}{}%
951 \ifcsvoid{mh@currentrepos}{-}{%
952   \@inmhrepostrue%
953   \addto@thismodule{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\endcsname
954     {\noexpand\mh@currentrepos}}%
955   \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\mh@currentrepos}}%
956 }%
957 }

```

**Test 25:** <file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master?stex>

`\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.

```

958 \newif\if@inimport\@inimportfalse
959 \def\latexml@import#1{\stex@debug{LaTeXML Import: #1}}%
960 \def\activate@defs#1{%
961   \stex@debug{Activating import #1}%
962   \if@inimport\else%
963     \latexml@import{#1}%
964     \def\inimport@module{#1}%
965     \stex@debug{Entering import #1}%
966     \@inimporttrue%
967   \fi%
968   \edef\activate@defs@uri{#1}%
969   \ifcsundef{module@defs@\activate@defs@uri}{%
970     \PackageError{stex}{No module with URI \activate@defs@uri loaded}{Probably missing an
971       \detokenize{\importmodule} (or variant) somewhere?
972   }
973 }{%
974   \ifcsundef{module@\activate@defs@uri @activated}%
975     {\csname module@defs@\activate@defs@uri\endcsname}{}%
976   \namedef{module@\activate@defs@uri @activated}{true}%
977 }%
978 \def\inimport@thismodule{#1}%
979 \stex@debug{End of import #1}%
980 \ifx\inimport@thismodule\inimport@module\@inimportfalse%
981   \stex@debug{Leaving import #1}%
982 \fi%
983 }%

```

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

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

**Test 26:**     Module 3.10[Foo]:           Module 3.11[Bar]:     »macro:->\@invoke@symbol  
                   {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master?Foo?foo}«

Module 3.12[Baz]:     Should be undefined: »undefined«

Should be defined: »macro:->\@invoke@symbol {file:///home/jazzpirate/work/Software/ext/sTeX

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

```
985 \def\inputref@preskip{}
```

```
986 \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).

```
987 \newrobustcmd\inputref[2] [] {%
```

```
988   \importmodule@bookkeeping{#1}{#2}{%
```

```
989     %\inputreftrue
```

```
990     \inputref@preskip%
```

```
991     \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
```

```
992     \inputref@postskip%
```

```
993   }%
```

```
994 }%
```

**Test 27:**           Module 3.13[type.en]:

### 3.5 Symbols/Notations/Verbalizations

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

```
995 \newif\if@symdeflocal\@symdeflocalfalse
```

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

```
996 \def\define@in@module#1#2{
```

```
997   \expandafter\edef\csname #1\endcsname{#2}%
```

```
998   \edef\define@in@module@temp{%
```

```
999     \def\expandafter\noexpand\csname#1\endcsname%
```

```
1000     {#2}%
```

```
1001   }%
```

```
1002   \if@symdeflocal\else%
```

```
1003     \expandafter\g@addto@macro@safe\csname module@defs\@module@uri%
```

```
1004     \expandafter\endcsname\expandafter{\define@in@module@temp}%
```

```
1005   \fi%
```

```
1006 }
```

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

```

1007 \addmetakey{symdecl}{name}%
1008 \addmetakey{symdecl}{type}%
1009 \addmetakey{symdecl}{args}%
1010 \addmetakey[false]{symdecl}{local}[true]%
1011
1012 \newcommand\symdecl[2][{}]{%
1013   \ifcsdef{this@module}{%
1014     \metasetkeys{symdecl}{#1}%
1015     \ifcsvoid{symdecl@name}{%
1016       \edef\symdecl@name{#2}%
1017     }{%
1018       \edef\symdecl@uri{module@uri\@QuestionMark\symdecl@name}%
1019       \ifcsvoid{stex@symbol@\symdecl@name}{%
1020         \expandafter\edef\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}%
1021       }{%
1022         \expandafter\def\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}%
1023       }%
1024       \edef\symdecl@symbolmacro{%
1025         \noexpand\ifcsvoid{stex@symbol@\symdecl@name}{%
1026           \expandafter\edef\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}%
1027         }{%
1028           \expandafter\def\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}%
1029         }%
1030       }%
1031       \ifcsvoid{symdecl@type}{%
1032         \setbox\modules@import@tempbox\hbox{$\symdecl@type$} % only to have latex check this
1033       }%
1034       \ifcsvoid{symdecl@args}{\csgdef{\symdecl@uri\@QuestionMark args}{}}{%
1035         \IfInteger\symdecl@args{\notation@num@to@ia@\symdecl@args\csxdef{\symdecl@uri\@QuestionMark args}{\notation@num@to@ia@\symdecl@args}}{%
1036           \ex@globale\ex\let\csname\symdecl@uri\@QuestionMark args\endcsname\symdecl@args%
1037         }%
1038       }%
1039       \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1040       \expandafter\endcsname\expandafter{\symdecl@symbolmacro}%
1041       \ifcsvoid{\symdecl@uri}{%
1042         \ifcsvoid{module@names@\module@uri}{%
1043           \csxdef{module@names@\module@uri}{\symdecl@name}%
1044         }{%
1045           \csxdef{module@names@\module@uri}{\symdecl@name,%
1046           \csname module@names@\module@uri\endcsname}%
1047         }%
1048       }{%
1049         % not compatible with circular dependencies, e.g. test/omdoc/07-modules/smsstesta.tex
1050         \PackageWarning{stex}{symbol already defined: \symdecl@uri}%
1051         You need to pick a fresh name for your symbol%
1052       }%
1053     }%
1054     \define@in@module\symdecl@uri{\noexpand\@invoke@symbol{\symdecl@uri}}%
1055     \define@in@module{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
1056   }{%

```



```

1057 \PackageError{stex}{\detokenize{\symdecl} not in a module}{You need to be in a module%
1058 in order to declare a new symbol}
1059 }%
1060 \if@inimport\else\latexml@symdecl\symdecl@uri{\$ \symdecl@type$}\fi%
1061 \if@insymdef\else\parsemodule@maybesetcodes\fi%
1062 }
1063 \def\latexml@symdecl#1{}

```

**Test 28:** Module 3.14[foo]: \symdecl {bar}

Yields: »macro:->\@invoke@symbol {file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex.

### 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:

```

1064 \edef\stex@ambiguous{\detokenize{ambiguous}}
1065 \edef\stex@macrostring{\detokenize{macro:->\@invoke@symbol}}
1066 \def\modules@getURIfromName#1{%
1067   \def\notation@uri{}%
1068   \edef\modules@getURI@name{#1}%
1069   \ifcsvoid{\modules@getURI@name}{
1070     \edef\modules@temp@meaning{
1071     }{
1072       \edef\modules@temp@meaning{\expandafter\meaning\csname\modules@getURI@name\endcsname}
1073     }
1074     \IfBeginWith\modules@temp@meaning\stex@macrostring{
1075       % is a \@invoke@symbol macro
1076       \StrPosition\modules@temp@meaning\@close@brace[\stex@tempnum]
1077       \StrMid\modules@temp@meaning{26}{\the\numexpr\stex@tempnum-1\@Space}[\notation@uri]
1078     }{
1079       % Check whether full URI or module?symbol or just name
1080       \StrCount\modules@getURI@name\@QuestionMark[\isuri@number]
1081       \ifnum\isuri@number=2
1082         \edef\notation@uri{\modules@getURI@name}
1083       \else
1084         \ifnum\isuri@number=1
1085           % module?name
1086           \StrCut\modules@getURI@name\@QuestionMark\isuri@mod\isuri@name
1087           \ifcsvoid{\stex@module@\isuri@mod}{
1088             \PackageError{stex}{No module with name \isuri@mod\@Space loaded}{}
1089           }{
1090             \expandafter\ifx\csname \stex@module@\isuri@mod\endcsname\stex@ambiguous
1091             \PackageError{stex}{Module name \isuri@mod\@Space is ambiguous}{}
1092           \else
1093             \edef\notation@uri{\csname \stex@module@\isuri@mod\endcsname\@URI\@QuestionMark\isur
1094           \fi
1095         }
1096       \else

```

```

1097      %name
1098      \ifcsvoid{stex@symbol@\modules@getURI@name}{
1099          \PackageError{stex}{No symbol with name \modules@getURI@name\@Space known}{}
1100      }{
1101          \ifcsvoid{\module@uri\@QuestionMark\modules@getURI@name}{
1102              \expandafter\ifx\csname stex@symbol@\modules@getURI@name\endcsname\stex@ambiguous
1103              % Symbol name ambiguous and not in current module
1104              \PackageError{stex}{Symbol name, URI or macroname \detokenize{#1} found!}{}%
1105          \else
1106              % Symbol not in current module, but unambiguous
1107              \edef\notation@uri{\csname stex@symbol@\modules@getURI@name\endcsname}
1108          \fi
1109      }{ % Symbol in current module
1110          \edef\notation@uri{\module@uri\@QuestionMark\modules@getURI@name}
1111      }
1112  }
1113  \fi
1114  \fi
1115  }
1116  }

```

**\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}`  
the actual notation is ultimately stored in `\langle uri \rangle \# \langle variant \rangle`, where `\langle variant \rangle`  
contains `arity`, `lang` and `variant` in that order.

```

1117 \newif\if@innotation\@innotationfalse

```

First, we eat the optional arguments in two separate macros and pass them on:

```

1118 \providerobustcmd\notation[2] [] {%
1119     \edef\notation@first{#1}%
1120     \edef\notation@second{#2}%
1121     \notation@%
1122 }
1123
1124 \newcommand\notation@[2] [0] {%
1125     \edef\notation@donext{\noexpand\notation@@[\notation@first]%
1126         {\notation@second}[#1]}%
1127     \notation@donext{#2}%
1128 }
1129

```

The next method actually parses the optional arguments and stores them in helper macros. This method will also be used later in symbol invocations to construct the `\langle variant \rangle`:

```

1130 \def\notation@parse@params#1#2{%
1131     \def\notation@curr@prec{}%
1132     \def\notation@curr@args{}%
1133     \def\notation@curr@variant{}%
1134     \def\notation@curr@arityvar{}%
1135     \def\notation@curr@provided@arity{#2}

```

```

1136 \def\notation@curr@lang{%
1137 \def\notation@options@temp{#1}
1138 \notation@parse@params@%
1139 \ifx\notation@curr@args\@empty%
1140 \ifx\notation@curr@provided@arity\@empty%
1141 \notation@num@to@ia\notation@curr@arityvar%
1142 \else%
1143 \notation@num@to@ia\notation@curr@provided@arity%
1144 \fi%
1145 \fi%
1146 \StrLen\notation@curr@args[\notation@curr@arity]%
1147 }
1148 \def\notation@parse@params@{%
1149 \IfSubStr\notation@options@temp,{%
1150 \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
1151 \notation@parse@param%
1152 \notation@parse@params@%
1153 }{\ifx\notation@options@temp\@empty\else%
1154 \let\notation@option@temp\notation@options@temp%
1155 \notation@parse@param%
1156 \fi}%
1157 }
1158
1159 \def\notation@parse@param{%
1160 \path@trimstring\notation@option@temp%
1161 \ifx\notation@option@temp\@empty\else%
1162 \IfSubStr\notation@option@temp={%
1163 \StrCut\notation@option@temp=\notation@key\notation@value%
1164 \path@trimstring\notation@key%
1165 \path@trimstring\notation@value%
1166 \IfStrEq\notation@key{prec}{%
1167 \edef\notation@curr@prec{\notation@value}%
1168 }{%
1169 \IfStrEq\notation@key{args}{%
1170 \edef\notation@curr@args{\notation@value}%
1171 }{%
1172 \IfStrEq\notation@key{lang}{%
1173 \edef\notation@curr@lang{\notation@value}%
1174 }{%
1175 \IfStrEq\notation@key{variant}{%
1176 \edef\notation@curr@variant{\notation@value}%
1177 }{%
1178 \IfStrEq\notation@key{arity}{%
1179 \edef\notation@curr@arityvar{\notation@value}%
1180 }{%
1181 }}}}%
1182 }{%
1183 \edef\notation@curr@variant{\notation@option@temp}%
1184 }%
1185 \fi%

```

```

1186 }
1187
1188 % converts an integer to a string of 'i's, e.g. 3 => iii,
1189 % and stores the result in \notation@curr@args
1190 \def\notation@num@to@ia#1{%
1191   \IfInteger{#1}{
1192     \notation@num@to@ia@#1%
1193   }{%
1194     %
1195   }%
1196 }
1197 \def\notation@num@to@ia@#1{%
1198   \ifnum#1>0%
1199     \edef\notation@curr@args{\notation@curr@args i}%
1200     \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
1201   \fi%
1202 }
1203
1204
1205 \newcount\notation@argument@counter
1206
1207 % parses the notation arguments and wraps them in
1208 % \notation@assoc and \notation@argprec for flexary arguments and precedences
1209 \def\notation@@[#1]#2[#3]#4{%
1210   \modules@getURIfromName{#2}%
1211   \notation@parse@params{#1}{#3}%
1212   \let\notation@curr@todo@args\notation@curr@args%
1213   \def\notation@temp@notation{}%
1214   \ex\renewcommand\ex\notation@temp@notation\ex[\notation@curr@arity]{#4}%
1215   % precedence
1216   \let\notation@curr@precstring\notation@curr@prec%
1217   \IfSubStr\notation@curr@prec;{%
1218     \StrCut\notation@curr@prec;\notation@curr@prec\notation@curr@prec%
1219     \ifx\notation@curr@prec\@empty\def\notation@curr@prec{0}\fi%
1220   }{%
1221     \ifx\notation@curr@prec\@empty%
1222       \ifnum\notation@curr@arity=0\relax%
1223         \edef\notation@curr@prec{\infprec}%
1224       \else%
1225         \def\notation@curr@prec{0}%
1226       \fi%
1227     \else%
1228       \edef\notation@curr@prec{\notation@curr@prec}%
1229       \def\notation@curr@prec{}%
1230     \fi%
1231   }%
1232   % arguments
1233   \notation@argument@counter=0%
1234   \def\notation@curr@extargs{}%

```

```

1235 \notation@do@args%
1236 }
1237
1238 \edef\notation@ichar{\detokenize{i}}%
1239
1240 % parses additional notation components for (associative) arguments
1241 \def\notation@do@args{%
1242   \advance\notation@argument@counter by 1%
1243   \def\notation@nextarg@temp{}%
1244   \ifx\notation@curr@todo@args\@empty%
1245     \ex\notation@after%
1246   \else%
1247     % argument precedence
1248     \IfSubStr\notation@curr@prec{x}{%
1249       \StrCut\notation@curr@prec{x}\notation@curr@argprec\notation@curr@prec%
1250     }{%
1251       \edef\notation@curr@argprec{\notation@curr@prec}%
1252       \def\notation@curr@prec{}%
1253     }%
1254     \ifx\notation@curr@argprec\@empty%
1255       \let\notation@curr@argprec\notation@curr@prec%
1256     \fi%
1257     \StrChar\notation@curr@todo@args1[\notation@argchar]%
1258     \edef\notation@argchar{\ex\detokenize\ex{\notation@argchar}}%
1259     \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1260     \ifx\notation@argchar\notation@ichar%
1261       % normal argument
1262       \edef\notation@nextarg@temp{%
1263         {\stex@arg{\the\notation@argument@counter}{\notation@curr@argprec}{#####\the
1264       }}%
1265       \ex@g@addto@macro@safe\ex\notation@curr@extargs%
1266       \ex{\notation@nextarg@temp}%
1267       \ex\ex\ex\notation@do@args%
1268     \else%
1269       % associative argument
1270       \ex\ex\ex\notation@parse@assocarg%
1271     \fi%
1272   \fi%
1273 }
1274
1275 \def\notation@parse@assocarg#1{%
1276   \edef\notation@nextarg@temp{%
1277     {\stex@arg{\the\notation@argument@counter}{\notation@curr@argprec}{\notation@assoc{#1}{#####
1278   }}%
1279   \ex@g@addto@macro@safe\ex\notation@curr@extargs\ex{\notation@nextarg@temp}%
1280   \notation@do@args%
1281 }
1282
1283 \protected\def\safe@newcommand#1{%
1284   \ifdefined#1\ex\renewcommand\else\ex\newcommand\fi#1%

```

```

1285 }
1286
1287 % finally creates the actual macros
1288 \def\notation@after{
1289   % \notation@curr@prec
1290   % \notation@curr@args
1291   % \notation@curr@variant
1292   % \notation@curr@arity
1293   % \notation@curr@provided@arity
1294   % \notation@curr@lang
1295   % \notation@uri
1296   \def\notation@temp@fragment{}%
1297   \ifx\notation@curr@arityvar\@empty\else%
1298     \edef\notation@temp@fragment{arity=\notation@curr@arityvar}%
1299   \fi%
1300   \ifx\notation@curr@lang\@empty\else%
1301     \ifx\notation@temp@fragment\@empty%
1302       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1303     \else%
1304       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1305     \fi%
1306   \fi%
1307   \ifx\notation@curr@variant\@empty\else%
1308     \ifx\notation@temp@fragment\@empty%
1309       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1310     \else%
1311       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@variant}%
1312     \fi%
1313   \fi%
1314   \ex\ex\ex\def\ex\ex\ex\notation@temp@notation\ex\ex\ex%
1315     {\ex\notation@temp@notation\notation@curr@extargs}%
1316   \ifnum\notation@curr@arity=0
1317     \edef\notation@temp@notation{\stex@oms{\notation@uri\@Fragment\notation@temp@fragment}}{\notation@temp@notation}%
1318   \else
1319     \edef\notation@temp@notation{\stex@oma{\notation@uri\@Fragment\notation@temp@fragment}}{\notation@temp@notation}%
1320   \fi
1321   \stex@debug{Notation \notation@uri: \meaning\notation@temp@notation}%
1322   \notation@final%
1323   \parsemodule@maybesetcodes%
1324 }
1325
1326 \def\notation@final{%
1327   \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1328   \stex@debug{Defining \notation@csname of arity \notation@curr@arity}%
1329   \ifcvoid{\notation@csname}{%
1330     \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1331       \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@curr@arity\ex]%
1332     \ex{\notation@temp@notation}%
1333   \edef\symdecl@temps{%
1334     \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@curr@arity%

```

```

1335 }%
1336 \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1337 \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1338 }-%
1339 \PackageWarning{stex}{notation already defined: \notation@csname}%
1340 Choose a different set of notation options (variant,lang,arity)%
1341 }%
1342 }%
1343 \innotationfalse%
1344 \ifinimport\else\iflatexml%
1345 \let\notation@simarg@args\notation@curr@args%
1346 \notation@argument@counter=0%
1347 \def\notation@simargs{}%
1348 \notation@simulate@arguments%
1349 \latexml\notation\notation@uri\notation@temp@fragment\notation@curr@args\notation@curr@prec
1350 {\csname\notation@csname\ex\endcsname\notation@simargs}%
1351 \fi\fi%
1352 }
1353 \def\notation@simulate@arguments{%
1354 \ifx\notation@simarg@args\empty\else%
1355 \advance\notation@argument@counter by 1%
1356 \IfBeginWith\notation@simarg@args{i}{%
1357 \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\the\notation@argument
1358 }}%
1359 \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\@Fragment\the\notation
1360 }}%
1361 \StrGobbleLeft\notation@simarg@args1[\notation@simarg@args]%
1362 \notation@simulate@arguments%
1363 \fi%
1364 }
1365 % URI, fragment, arity, notation
1366 \def\latexmlnotation#1#2#3#4{}

```

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

```

1367 \protected\def\notation@assoc#1#2{% function, argv
1368 \let\@tmpop=\relax% do not print the function the first time round
1369 \@for\@I:=#2\do{\@tmpop% print the function
1370 % write the i-th argument with locally updated precedence
1371 \@I%
1372 \def\@tmpop{#1}%
1373 }%
1374 }%
1375
1376 \def\notation@lparen{()
1377 \def\notation@rparen{)}
1378 \def\infprec{1000000}
1379 \def\neginfprec{-\infprec}
1380
1381 \newcount\notation@downprec

```

```

1382 \notation@downprec=\neginfprec
1383
1384 % patching displaymode
1385 \newif\if@displaymode\@displaymodefalse
1386 \ex\everydisplay\ex{\the\everydisplay\@displaymodetrue}
1387 \let\old@displaystyle\displaystyle
1388 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1389
1390 \protected\def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1391   \def\notation@innertmp{#1}%
1392   \if@displaymode%
1393     \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1394     \ex\notation@resetbrackets\ex\notation@innertmp%
1395     \ex\right\notation@rparen%
1396   \else%
1397     \ex\ex\ex\notation@lparen%
1398     \ex\notation@resetbrackets\ex\notation@innertmp%
1399     \notation@rparen%
1400   \fi%
1401 }
1402
1403 \protected\def\withbrackets#1#2#3{%
1404   \edef\notation@lparen{#1}%
1405   \edef\notation@rparen{#2}%
1406   #3%
1407   \notation@resetbrackets%
1408 }
1409
1410 \protected\def\notation@resetbrackets{%
1411   \def\notation@lparen{({}%
1412   \def\notation@rparen{)}}%
1413 }
1414
1415 \protected\def\stex@oms#1#2#3{%
1416   \if@innotation%
1417     \notation@symprec{#2}{#3}%
1418   \else%
1419     \@innotationtrue%
1420     \latexml@oms{#1}{\notation@symprec{#2}{#3}}%
1421     \@innotationfalse%
1422   \fi%
1423 }
1424
1425 % for LaTeXXML Bindings
1426 \def\latexml@oms#1#2{%
1427   #2%
1428 }
1429
1430 \protected\def\stex@oma#1#2#3{%
1431   \if@innotation%

```



```

1432     \notation@symprec{#2}{#3}%
1433 \else%
1434     \@innotationtrue%
1435     \latexml@oma{#1}{\notation@symprec{#2}{#3}}%
1436     \@innotationfalse%
1437 \fi%
1438 }
1439
1440 % for LaTeXML Bindings
1441 \def\latexml@oma#1#2{%
1442     #2%
1443 }
1444
1445 \def\notation@symprec#1#2{%
1446     \ifnum#1>\notation@downprec\relax%
1447         \notation@resetbrackets#2%
1448     \else%
1449         \ifnum\notation@downprec=\infprec\relax%
1450             \notation@resetbrackets#2%
1451         \else
1452             \if@inarray@
1453                 \notation@resetbrackets#2
1454             \else\dobrackets{#2}\fi%
1455         \fi\fi%
1456 }
1457
1458 \newif\if@inarray@\@inarray@false
1459
1460
1461 \protected\def\stex@arg#1#2#3{%
1462     \@innotationfalse%
1463     \latexml@arg{#1}{\notation@argprec{#2}{#3}}%
1464     \@innotationtrue%
1465 }
1466
1467 % for LaTeXML Bindings
1468 \def\latexml@arg#1#2{%
1469     #2%
1470 }
1471
1472 \def\notation@argprec#1#2{%
1473     \def\notation@innertmp{#2}
1474     \edef\notation@downprec@temp{\number#1}%
1475     \notation@downprec=\expandafter\notation@downprec@temp%
1476     \expandafter\relax\expandafter\notation@innertmp%
1477     \expandafter\notation@downprec\expandafter=\number\notation@downprec\relax%
1478 }

```

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

```

1479 \protected\def\@invoke@symbol#1{%

```

```

1480 \def\@invoke@symbol@first{#1}%
1481 \symbol@args%
1482 }

```

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

```

1483 \newcommand\symbol@args[1][]{%
1484 \notation@parse@params{#1}{}%
1485 \def\notation@temp@fragment{}%
1486 \ifx\notation@curr@arityvar\@empty\else%
1487 \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1488 \fi%
1489 \ifx\notation@curr@lang\@empty\else%
1490 \ifx\notation@temp@fragment\@empty%
1491 \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1492 \else%
1493 \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1494 \fi%
1495 \fi%
1496 \ifx\notation@curr@variant\@empty\else%
1497 \ifx\notation@temp@fragment\@empty%
1498 \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1499 \else%
1500 \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@variant}%
1501 \fi%
1502 \fi%
1503 %
1504 \ifmmode\def\invoke@symbol@next{\@invoke@symbol@math\@invoke@symbol@first\notation@temp@fragment}%
1505 \else\def\invoke@symbol@next{\@invoke@symbol@text\@invoke@symbol@first\notation@temp@fragment}%
1506 \invoke@symbol@next%
1507 }

```

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

```

1508 \def\@invoke@symbol@math#1#2{%
1509 \csname #1\@Fragment#2\endcsname%
1510 }

```

TODO:

```

1511 \def\@invoke@symbol@text#1#2{%
1512 }

```

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

```

1513 \def\setstexlang#1{%
1514 \def\stex@lang{#1}%
1515 }%
1516 \setstexlang{en}
1517 \def\setstexvariant#1#2{%
1518 % TODO

```

```

1519 }
1520 \def\setstexvariants#1{%
1521   \def\stex@variants{#1}%
1522 }

```

```

Test 29:      Module 3.15[FooBar]:  \symdecl {barbar}
\notation [arity=0]{barbar}{\psi }
\notation [prec=50;\infprec ]{barbar}[1]{\barbar [arity=0]\dobrackets
{####1}}
\notation [arity=0,variant=cap]{barbar}{\Psi }
\notation [variant=cap]{barbar}[1]{\barbar [arity=0,variant=cap]\dobrackets
{####1}}

```

```

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

```

```

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

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

```

```

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

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


```

```

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

```

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

### 3.6 Term References

\ifhref

```

1523 \newif\ifhhref\hreffalse%
1524 \AtBeginDocument{%
1525   \@ifpackageloaded{hyperref}{%
1526     \hreftrue%
1527   }{%
1528     \hreffalse%
1529   }%
1530 }

\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.

1531 \newbox\stex@targetbox
1532 \def\termref@maketarget#1#2{%
1533   % #1: symbol URI
1534   % #2: text
1535   \stex@debug{Here: #1 <> #2}%
1536   \ifhhref\if@smsmode\else%
1537     \hypertarget{sref@#1@target}{#2}%
1538   \fi\fi%
1539   \stex@debug{Here!}%
1540   \expandafter\edef\csname sref@#1\endcsname##1{%
1541     \ifhhref\if@smsmode\else\noexpand\hyperlink{sref@#1@target}{##1}\fi\fi%
1542   }%
1543 }

\@termref

1544 \def\@termref#1#2{%
1545   % #1: symbol URI
1546   % #2: text
1547   \ifcsvoid{#1}{%
1548     \StrCut[2]{#1}\@QuestionMark\termref@mod\termref@name%
1549     \ifcsvoid{\termref@mod}{%
1550       \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
1551     }{%
1552       \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
1553         contains no symbol with name \termref@name.%
1554       }{%
1555       }%
1556     }%
1557     \ifcsvoid{sref@#1}{%
1558       #2% TODO: No reference point exists!
1559     }{%
1560       \csname sref@#1\endcsname{#2}%
1561     }%
1562   }%
1563 }

\tref
1564

```

```

1565 \def\@capitalize#1{\uppercase{#1}}%
1566 \newrobustcmd\capitalize[1]{\expandafter\@capitalize #1}%
1567
1568 \newcommand\tref[2][]{%
1569   \edef\tref@name{#1}%
1570   \expandafter\modules@getURIfromName\expandafter{\tref@name}%
1571   \expandafter\@termref\expandafter{\notation@uri}{#2}%
1572 }
1573 \def\trefs#1{%
1574   \modules@getURIfromName{#1}%
1575   % TODO
1576 }
1577 \def\Tref#1{%
1578   \modules@getURIfromName{#1}%
1579   % TODO
1580 }
1581 \def\Trefs#1{%
1582   \modules@getURIfromName{#1}%
1583   % TODO
1584 }

\defi
1585 \addmetakey{defi}{name}
1586 \def\@definiendum#1#2{%
1587   \parsemodule@maybesetcodes%
1588   \stex@debug{Here: #1 | #2}%
1589   \termref@maketarget{#1}{#2}%\termref@maketarget{#1}{\defemph{#2}}%
1590 }
1591
1592 \newcommand\defi[2][]{%
1593   \metasetkeys{defi}{#1}%
1594   \ifx\defi@name\@empty%
1595     \symdecl@constructname{#2}%
1596     \let\defi@name\symdecl@name%
1597     \let\defi@verbalization\symdecl@verbalization%
1598   \else%
1599     \edef\defi@verbalization{#2}%
1600   \fi%
1601   \ifcvoid{\module@uri\@QuestionMark\defi@name}{%
1602     \symdecl\defi@name%
1603   }{\edef\symdecl@uri{\module@uri\@QuestionMark\defi@name}}%
1604   \@definiendum\symdecl@uri\defi@verbalization%
1605 }
1606 \def\Defi#1{%
1607   \symdecl{#1}%
1608   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization}%
1609 }
1610 \def\defis#1{%
1611   \symdecl{#1}%
1612   \@definiendum\symdecl@uri{\symdecl@verbalization s}%

```

```

1613 }
1614 \def\Defis#1{%
1615   \symdecl{#1}%
1616   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization s}%
1617 }

```

### 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
1618 \newif\ifhref\hreffalse%
1619 \AtBeginDocument{%
1620   \ifpackageloaded{hyperref}{%
1621     \hreftrue%
1622   }{%
1623     \hreffalse%
1624   }%
1625 }%
1626 \newcommand\sref@href@ifh[2]{%
1627   \ifhref%
1628     \href{#1}{#2}%
1629   \else%
1630     #2%
1631   \fi%
1632 }%
1633 \newcommand\sref@hlink@ifh[2]{%
1634   \ifhref%
1635     \hyperlink{#1}{#2}%
1636   \else%
1637     #2%
1638   \fi%
1639 }%
1640 \newcommand\sref@target@ifh[2]{%
1641   \ifhref%
1642     \hypertarget{#1}{#2}%
1643   \else%
1644     #2%
1645   \fi%
1646 }%

```

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.

```

1647 \def\sref@target{%
1648   \ifx\sref@id\@empty%

```

```

1649     \relax%
1650   \else%
1651     \edef\@target{sref@ifcsundef{sref@part}}{\sref@part @}\sref@id @target}%
1652     \sref@target@ifh\@target}%
1653   \fi%
1654 }%

\srefaddidkey \srefaddidkey[⟨keyval⟩]{⟨group⟩} extends the metadata keys of the group
⟨group⟩ with an id key. In the optional key/value pairs in ⟨keyval⟩ the
prefix key can be used to specify a prefix. Note that the id key defined by
\srefaddidkey[⟨keyval⟩]{⟨group⟩} not only defines \sref@id, which is used for
referencing by the sref package, but also \⟨group⟩@id, which is used for showing
metadata via the showmeta option of the metakeys package.
1655 \addmetakey{srefaddidkey}{prefix}
1656 \newcommand\srefaddidkey[2][]{%
1657   \metasetkeys{srefaddidkey}{#1}%
1658   \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
1659   \metakeys@ext@clear@keys{#2}{id}{}%
1660   \metakeys@ext@showkeys{#2}{id}%
1661   \define@key{#2}{id}{%
1662     \edef\sref@id{\srefaddidkey@prefix ##1}%
1663     %\expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
1664     \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1665   }%
1666 }%

\@sref@def This macro stores the value of its last argument in a custom macro for reference.
1667 \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.
1668 \ifextrefs%
1669   \newwrite\refs@file%
1670 \else%
1671   \def\refs@file{\@auxout}%
1672 \fi%

\sref@def This macro writes an \@sref@def command to the current aux file and also exe-
cutes it.
1673 \newcommand\sref@def[3]{%
1674   \protected@write\refs@file{}{\string\@sref@def{#1}{#2}{#3}}%
1675 }%

\sref@label The \sref@label macro writes a label definition to the auxfile.
1676 \newcommand\sref@label[2]{%
1677   \sref@def{\ifcsundef{sref@part}}{\sref@part @}{#2}{page}{\thepage}%
1678   \sref@def{\ifcsundef{sref@part}}{\sref@part @}{#2}{label}{#1}%
1679 }%

```

`\sreflabel` The `\sreflabel` macro is a semantic version of `\label`, it combines the categorization given in the first argument with L<sup>A</sup>T<sub>E</sub>X's `\@currentlabel`.

```
1680 \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.

```
1681 \def\sref@id{} % make sure that defined
1682 \newcommand\sref@label@id[1]{%
1683   \ifx\sref@id\@empty%
1684     \relax%
1685   \else%
1686     \sref@label{#1}{\sref@id}%
1687   \fi%
1688 }%
```

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

```
1689 \newcommand\sref@label@id@arg[2]{%
1690   \def\@id{#2}
1691   \ifx\@id\@empty%
1692     \relax%
1693   \else%
1694     \sref@label{#1}{\@id}%
1695   \fi%
1696 }%
```

### 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.

```
1697 \newenvironment{modsig}[2][]{\def\@test{#1}%
1698 \ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1699 \expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
1700 %\ignorespacesandpars
1701 }
1702 {\end{module}}\ignorespacesandpars
1703 }
```

### 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`.

```

1704 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1705 \newrobustcmd\@gimport@star[2] [] {\def\@test{#1}%
1706 \edef\mh@currentrepos{\mh@currentrepos}%
1707 \ifx\@test\@empty%
1708 \importmhmodule[conservative,mhrepos=\mh@currentrepos,path=#2]{#2}%
1709 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1710 \mathhub@setcurrentreposinfo{\mh@currentrepos}%
1711 %\ignorespacesandpars
1712 \parsemodule@maybesetcodes}
1713 \newrobustcmd\@gimport@nostar[2] [] {\def\@test{#1}%
1714 \edef\mh@currentrepos{\mh@currentrepos}%
1715 \ifx\@test\@empty%
1716 \importmhmodule[mhrepos=\mh@currentrepos,path=#2]{#2}%
1717 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1718 \mathhub@setcurrentreposinfo{\mh@currentrepos}%
1719 %\ignorespacesandpars
1720 \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.

```

1721 \def\modules@@first#1/#2;{#1}
1722 \newcommand\libinput[1]{%
1723 \stex@debug{Libinput current repo: \meaning\mh@currentrepos}%
1724 \ifcsvoid{mh@currentrepos}{%
1725 \PackageError{stex}{current MathHub repository not found}{}}%
1726 {}
1727 \edef\@mh@group{\expandafter\modules@@first\mh@currentrepos;}
1728 \let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
1729 \def\mh@inffile{\MathHub{\@mh@group/meta-inf/lib/#1}}
1730 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1731 \IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
1732 \IfFileExists\mh@inffile{\IfFileExists\mh@libfile}{%
1733 {\PackageError{stex}
1734 {Library file missing; cannot input #1.tex\MessageBreak%
1735 Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
1736 do not exist}%
1737 {Check whether the file name is correct}}}}

```

```

1738 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1739 \let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}

```

### 3.11 omdoc/omgroup

```

1740 \newcount\section@level
1741
1742 \section@level=2
1743 \ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{}
1744 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{}
1745 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{}
1746 \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>*.

```

1747 \newcommand\omgroup@nonum[2]{%
1748 \ifx\hyper@anchor\undefined\else\phantomsection\fi%
1749 \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.

```

1750 \newcommand\omgroup@num[2]{%
1751 \edef\@@ID{\sref@id}
1752 \ifx\omgroup@short\@empty% no short title
1753 \@nameuse{#1}{#2}%
1754 \else% we have a short title
1755 \@ifundefined{rdfmeta@sectioning}%
1756   {\@nameuse{#1}[\omgroup@short]{#2}}%
1757   {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
1758 \fi%
1759 \sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@@ID}

```

`omgroup`

```

1760 \def\@true{true}
1761 \def\@false{false}
1762 \srefaddidkey{omgroup}
1763 \addmetakey{omgroup}{date}
1764 \addmetakey{omgroup}{creators}
1765 \addmetakey{omgroup}{contributors}
1766 \addmetakey{omgroup}{srccite}
1767 \addmetakey{omgroup}{type}
1768 \addmetakey*{omgroup}{short}
1769 \addmetakey*{omgroup}{display}
1770 \addmetakey[false]{omgroup}{loadmodules}[true]

```

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

`\at@begin@omgroup` The `\at@begin@omgroup` macro allows customization. It is run at the beginning

of the `omgroup`, i.e. after the section heading.

```
1771 \newif\if@mainmatter\@mainmattertrue
1772 \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.

```
1773 \addmetakey{omdoc@sect}{name}
1774 \addmetakey[false]{omdoc@sect}{clear}[true]
1775 \addmetakey{omdoc@sect}{ref}
1776 \addmetakey[false]{omdoc@sect}{num}[true]
1777 \newcommand\omdoc@sectioning[3] [] {\metasetkeys{omdoc@sect}{#1}%
1778 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
1779 \if@mainmatter% numbering not overridden by frontmatter, etc.
1780 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
1781 \def\current@section@level{\omdoc@sect@name}%
1782 \else\omgroup@nonum{#2}{#3}%
1783 \fi}% if@mainmatter
```

and another one, if redefines the `\addtocontentsline` macro of  $\text{\LaTeX}$  to import the respective macros. It takes as an argument a list of module names.

```
1784 \newcommand\omgroup@redefine@addtocontents[1]{%
1785 %\edef\@import{#1}%
1786 %\@for\@I:=\@import\do{%
1787 %\edef\@path{\csname module@\@I @path\endcsname}%
1788 %\@ifundefined{tf@toc}\relax%
1789 %    {\protected@write\tf@toc}{\string\@requiremodules{\@path}}}%
1790 %\ifx\hyper@anchor\undefined% hyperref.sty loaded?
1791 %\def\addcontentsline##1##2##3{%
1792 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{#1}{##3}}{\thepage}}%
1793 %\else% hyperref.sty not loaded
1794 %\def\addcontentsline##1##2##3{%
1795 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{#1}{##3}}{\thepage}}{\@c
1796 %\fi
1797 }% hyperref.sty loaded?
```

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

```
1798 \newcount\omgroup@level
1799 \newenvironment{omgroup}[2] []% keys, title
1800 {\metasetkeys{omgroup}{#1}\sref@target%
1801 \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.

```
1802 \ifx\omgroup@loadmodules\@true%
1803 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
1804 {\@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`.

```

1805 \advance\section@level by 1\relax%
1806 \ifcase\section@level%
1807 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
1808 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
1809 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
1810 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
1811 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%
1812 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{#2}%
1813 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{#2}%
1814 \fi% \ifcase
1815 \at@begin@omgroup[#1]\section@level{#2}}% for customization
1816 {\advance\section@level by -1\advance\omgroup@level by -1}

```

and finally, we localize the sections

```

1817 \newcommand\omdoc@part@kw{Part}
1818 \newcommand\omdoc@chapter@kw{Chapter}
1819 \newcommand\omdoc@section@kw{Section}
1820 \newcommand\omdoc@subsection@kw{Subsection}
1821 \newcommand\omdoc@subsubsection@kw{Subsubsection}
1822 \newcommand\omdoc@paragraph@kw{paragraph}
1823 \newcommand\omdoc@subparagraph@kw{subparagraph}

```

`\setSGvar` set a global variable

```

1824 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}

```

`\useSGvar` use a global variable

```

1825 \newrobustcmd\useSGvar[1]{%
1826   \ifundefined{sTeX@Gvar@#1}
1827   {\PackageError{omdoc}
1828     {The sTeX Global variable #1 is undefined}
1829     {set it with \protect\setSGvar}}
1830 \@nameuse{sTeX@Gvar@#1}}

```

`blindomgroup`

```

1831 \newcommand\at@begin@blindomgroup[1]{%
1832 \newenvironment{blindomgroup}
1833 {\advance\section@level by 1\at@begin@blindomgroup\setion@level}
1834 {\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).

```

1835 \srefaddidkey{omtext}
1836 \addmetakey[] {omtext}{functions}
1837 \addmetakey*{omtext}{display}
1838 \addmetakey{omtext}{for}
1839 \addmetakey{omtext}{from}
1840 \addmetakey{omtext}{type}
1841 \addmetakey*{omtext}{title}
1842 \addmetakey*{omtext}{start}
1843 \addmetakey{omtext}{theory}
1844 \addmetakey{omtext}{continues}
1845 \addmetakey{omtext}{verbalizes}
1846 \addmetakey{omtext}{subject}

\st@flow We define this macro, so that we can test whether the display key has the value
flow
1847 \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.

1848 \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.

1849 \def\omtext@pre@skip{\smallskip}
1850 \def\omtext@post@skip{}
1851 \newenvironment{omtext}[1] [] {\@in@omtexttrue%
1852 \bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
1853 \def\lec##1{\@lec{##1}}%
1854 \omtext@pre@skip\par\noindent%
1855 \ifx\omtext@title\@empty%
1856 \ifx\omtext@start\@empty\else%
1857 \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
1858 \fi% end omtext@start empty
1859 \else\stDMemph{\omtext@title}:\enspace%
1860 \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
1861 \fi% end omtext@title empty
1862 %\ignorespacesandpars
1863 }
1864 {\egroup\omtext@post@skip\@in@omtextfalse%\ignorespacesandpars
1865 }

```

### 3.12.2 Phrase-level Markup

```

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

1866 \srefaddidkey{phrase}
1867 \addmetakey{phrase}{style}
1868 \addmetakey{phrase}{class}
1869 \addmetakey{phrase}{index}

```

```

1870 \addmetakey{phrase}{verbalizes}
1871 \addmetakey{phrase}{type}
1872 \addmetakey{phrase}{only}
1873 \newcommand\phrase[2] [] {\metasetkeys{phrase}{#1}%
1874 \ifx\prhaseonly\empty\only<\phraseonly>{#2}\else #2\fi}

\coref*
1875 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}
1876 \newcommand\corefs[2]{#1\textsubscript{#2}}
1877 \newcommand\coreft[2]{#1\textsuperscript{#2}}

\n*lex
1878 \newcommand\nlex[1]{\green{\sl{#1}}}
1879 \newcommand\nlcex[1]{*\green{\sl{#1}}}

sinlinequote
1880 \def\@sinlinequote#1{‘‘\sl{#1}}’}
1881 \def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
1882 \newcommand\sinlinequote[2] []
1883 {\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.

```

1884 \newcommand\vdec[2] [] {#2}
1885 \newcommand\vest[2] [] {#2}
1886 \newcommand\vcond[2] [] {#2}

```

EdN:1	<pre> \strucdec 1 1887 \newcommand\strucdec[2] [] {#2} </pre>
EdN:2	<pre> \impdec 2 1888 \newcommand\impdec[2] [] {#2} </pre>

### 3.12.4 Block-Level Markup

```

sblockquote
1889 \def\begin@sblockquote{\begin{quote}\sl}
1890 \def\end@sblockquote{\end{quote}}
1891 \def\begin@@sblockquote#1{\begin@sblockquote}
1892 \def\end@@sblockquote#1{\def\@lec##1{\textrm{##1}}\@lec{#1}\end@sblockquote}
1893 \newenvironment{sblockquote}[1] []
1894 {\def\@opt{#1}\ifx\@opt\empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
1895 {\ifx\@opt\empty\end@sblockquote\else\end@@sblockquote\@opt\fi}

```

---

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

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

sboxquote

```
1896 \newenvironment{sboxquote}[1] []
1897 {\def\@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
1898 {\@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.

```
1899 \providecommand{\@lec}[1]{( #1 )}
1900 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@lec{#1}}
1901 \def\lec#1{\@lec{#1}\par}
```

### 3.12.5 Index Markup

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

```
1902 \addmetakey{omdoc@index}{at}
1903 \addmetakey[false]{omdoc@index}{loadmodules}[true]
1904 \newcommand\omdoc@indexi[2] [] {\ifindex%
1905 \metasetkeys{omdoc@index}{#1}%
1906 \@bsphack\begingroup\@sanitize%
1907 \protected@write\@indexfile{}\string\indexentry%
1908 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1909 \ifx\omdoc@index@loadmodules\@true%
1910 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
1911 \else #2\fi% loadmodules
1912 }\thepage}}%
1913 \endgroup\@esphack\fi}%ifindex
1914 \newcommand\omdoc@indexii[3] [] {\ifindex%
1915 \metasetkeys{omdoc@index}{#1}%
1916 \@bsphack\begingroup\@sanitize%
1917 \protected@write\@indexfile{}\string\indexentry%
1918 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1919 \ifx\omdoc@index@loadmodules\@true%
1920 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1921 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
1922 \else #2!#3\fi% loadmodules
1923 }\thepage}}%
1924 \endgroup\@esphack\fi}%ifindex
1925 \newcommand\omdoc@indexiii[4] [] {\ifindex%
1926 \metasetkeys{omdoc@index}{#1}%

```

```

1927 \@bsphack\beginingroup\@sanitize%
1928 \protected@write\@indexfile{}\string\indexentry%
1929 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1930 \ifx\omdoc@index@loadmodules\@true%
1931 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1932 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1933 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1934 \else #2!#3!#4\fi% loadmodules
1935 }\the page}}%
1936 \endgroup\@esphack\fi}%ifindex
1937 \newcommand\omdoc@indexiv[5] [] {\ifindex%
1938 \metasetkeys{omdoc@index}{#1}%
1939 \@bsphack\beginingroup\@sanitize%
1940 \protected@write\@indexfile{}\string\indexentry%
1941 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1942 \ifx\omdoc@index@loadmodules\@true%
1943 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1944 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1945 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1946 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
1947 \else #2!#3!#4!#5\fi% loadmodules
1948 }\the page}}%
1949 \endgroup\@esphack\fi}%ifindex

```

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

\\*indi\*

```

1950 \newcommand\aindi[3] [] {\#2\omdoc@indexi[#1]{#3}}
1951 \newcommand\indi[2] [] {\#2\omdoc@indexi[#1]{#2}}
1952 \newcommand\indis[2] [] {\#2\omdoc@indexi[#1]{#2s}}
1953 \newcommand\Indi[2] [] {\captitalize{\#2}\omdoc@indexi[#1]{#2}}
1954 \newcommand\Indis[2] [] {\capitalize{\#2}\omdoc@indexi[#1]{#2s}}
1955
1956 \newcommand\@indii[3] [] {\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
1957 \newcommand\aindii[4] [] {\#2\@indii[#1]{#3}{#4}}
1958 \newcommand\indii[3] [] {\#2 #3\@indii[#1]{#2}{#3}}
1959 \newcommand\indiis[3] [] {\#2 #3s\@indii[#1]{#2}{#3}}
1960 \newcommand\Indii[3] [] {\captitalize{\#2 #3}\@indii[#1]{#2}{#3}}
1961 \newcommand\Indiis[3] [] {\capitalize{\#2 #3}\@indii[#1]{#2}{#3}}
1962
1963 \newcommand\@indiii[4] [] {\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexiii[#1]{#3}{#2}{#4}}
1964 \newcommand\aindiii[5] [] {\#2\@indiii[#1]{#3}{#4}{#5}}
1965 \newcommand\indiii[4] [] {\#2 #3 #4\@indiii[#1]{#2}{#3}{#4}}
1966 \newcommand\indiis[4] [] {\#2 #3 #4s\@indiii[#1]{#2}{#3}{#4}}
1967 \newcommand\Indiii[4] [] {\captitalize{\#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
1968 \newcommand\Indiis[4] [] {\capitalize{\#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}
1969
1970 \newcommand\@indiv[5] [] {\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
1971 \newcommand\aindiv[6] [] {\#2\@indiv[#1]{#3}{#4}{#5}{#6}}
1972 \newcommand\indiv[5] [] {\#2 #3 #4 #5\@indiv[#1]{#2}{#3}{#4}{#5}}

```



```

1973 \newcommand\indivs[5] [] {\#2 \#3 \#4 \#5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
1974 \newcommand\Indiv[5] [] {\capitaliz{\#2 \#3 \#4 \#5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
1975 \newcommand\Indivs[5] [] {\capitaliz{\#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.

```

1976 \newcommand\hateq{\ensuremath{\widehat{=}}\xspace}
1977 \newcommand\hatequiv{\ensuremath{\widehat{equiv}}\xspace}
1978 \@ifundefined{ergo}%
1979 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1980 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1981 \newcommand{\reflect@squig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%
1982 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
1983 \newcommand\notergo{\ensuremath{\not\leadsto}}
1984 \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*`

```

1985 \newcommand\indextoo[2] [] {\indi[#1]{#2}}%
1986 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}%
1987 \newcommand\indexalt[2] [] {\aindi[#1]{#2}}%
1988 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead}%
1989 \newcommand\twintoo[3] [] {\indii[#1]{#2}{#3}}%
1990 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}%
1991 \newcommand\twinalt[3] [] {\aindii[#1]{#2}{#3}}%
1992 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead}%
1993 \newcommand\atwintoo[4] [] {\indiii[#1]{#2}{#3}{#4}}%
1994 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead}%
1995 \newcommand\atwinalt[4] [] {\aindiii[#1]{#2}{#3}{#4}}%
1996 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instead}%

```

`\my*graphics`

```

1997 \newcommand\mygraphics[2] [] {\includegraphics[#1]{#2}}%
1998 \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics}%
1999 \newcommand\mycgraphics[2] [] {\begin{center}\mygraphics[#1]{#2}\end{center}}%
2000 \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics}%
2001 \newcommand\mybgraphics[2] [] {\fbox{\mygraphics[#1]{#2}}}%
2002 \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics}%
2003 \newcommand\mycbgraphics[2] [] {\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}%
2004 \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphics}%

```

## 4 Things to deprecate

Module options:

```
2005 \addmetakey*{module}{id} % TODO: deprecate properly
2006 \addmetakey*{module}{load}
2007 \addmetakey*{module}{path}
2008 \addmetakey*{module}{dir}
2009 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
2010 \addmetakey*{module}{noalign}[true]
2011
2012 \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.

```
2013 %\srefaddidkey{symdef}% what does this do?
2014 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
2015 \define@key{symdef}{noverb}[all]{}%
2016 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
2017 \define@key{symdef}{specializes}{}%
2018 \addmetakey*{symdef}{noalign}[true]
2019 \define@key{symdef}{primary}[true]{}%
2020 \define@key{symdef}{assocarg}{}%
2021 \define@key{symdef}{bvars}{}%
2022 \define@key{symdef}{bargs}{}%
2023 \addmetakey{symdef}{lang}%
2024 \addmetakey{symdef}{prec}%
2025 \addmetakey{symdef}{arity}%
2026 \addmetakey{symdef}{variant}%
2027 \addmetakey{symdef}{ns}%
2028 \addmetakey{symdef}{args}%
2029 \addmetakey{symdef}{name}%
2030 \addmetakey*{symdef}{title}%
2031 \addmetakey*{symdef}{description}%
2032 \addmetakey{symdef}{subject}%
2033 \addmetakey*{symdef}{display}%
2034 \addmetakey*{symdef}{gfc}%
```

EdN:3

3

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

```
2035 \def\symdef{\@ifnextchar[{\@symdef}{\@symdef []}}%
2036 \def\@symdef[#1]#2{\@ifnextchar[{\@@symdef[#1]{#2}}{\@@symdef[#1]{#2}[0]}}%
```

---

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

`\@@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.

```

2037 \def\@@symdef[#1]#2[#3]{%
2038   \@insymdef@true%
2039   \metasetkeys{symdef}{#1}%
2040   \edef\symdef@tmp@optpars{\ifcvoid{symdef@name}{[]}{[name=\symdef@name]}}%
2041   \expandafter\symdecl\symdef@tmp@optpars{#2}%
2042   \@insymdef@false%
2043   \notation[#1]{#2}[#3]%
2044 }% mod@show
2045 \def\symdef@type{Symbol}%
2046 \providecommand{\stDMemph}[1]{\textbf{#1}}

```

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

```

2047 \def\symvariant#1{%
2048   \@ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}%
2049 }%
2050 \def\@symvariant#1[#2]#3#4{%
2051   \notation[#3]{#1}[#2]{#4}%
2052 %\ignorespacesandpars
2053 }%

```

`\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.

```

2054 \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.

```

2055 \newif\if@importing\@importingfalse
2056 \define@key{symi}{noverb}{all}{}%
2057 \define@key{symi}{align}{WithTheSymbolOfTheSameName}{}%
2058 \define@key{symi}{specializes}{}%
2059 \define@key{symi}{gfc}{}%
2060 \define@key{symi}{noalign}{true}{}%
2061 \newcommand\symi{\@ifstar\@symi@star\@symi}
2062 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
2063   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces
2064 }
2065 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
2066   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignorespaces
2067 }
2068 \newcommand\symii{\@ifstar\@symii@star\@symii}
2069 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
2070   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespaces
2071 }
2072 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%

```

```

2073 \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\
2074 }
2075 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
2076 \newcommand\@symiii[4] [] {\metasetkeys{symi}{#1}%
2077 \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignorespaces
2078 }
2079 \newcommand\@symiii@star[4] [] {\metasetkeys{symi}{#1}%
2080 \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi\ignorespaces
2081 }
2082 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
2083 \newcommand\@symiv[5] [] {\metasetkeys{symi}{#1}%
2084 \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces
2085 }
2086 \newcommand\@symiv@star[5] [] {\metasetkeys{symi}{#1}%
2087 \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces
2088 }

```

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

```

2089 %\srefaddidkey{importmhmodule}%
2090 \addmetakey{importmhmodule}{mhrepos}%
2091 \addmetakey{importmhmodule}{path}%
2092 \addmetakey{importmhmodule}{ext}% why does this exist?
2093 \addmetakey{importmhmodule}{dir}%
2094 \addmetakey[false]{importmhmodule}{conservative}[true]%
2095 \newcommand\importmhmodule[2] [] {%
2096 \parsemodule@maybesetcodes
2097 \metasetkeys{importmhmodule}{#1}%
2098 \ifx\importmhmodule@dir\@empty%
2099 \edef\@path{\importmhmodule@path}%
2100 \else\edef\@path{\importmhmodule@dir/#2}\fi%
2101 \ifx\@path\@empty% if module name is not set
2102 \importmodule[] {#2}{export}%
2103 \else%
2104 \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
2105 \ifx\importmhmodule@mhrepos\@empty% if in the same repos
2106 \relax% no need to change mh@currentrepos, i.e., current directory.
2107 \else%
2108 \mathhub@setcurrentreposinfo\importmhmodule@mhrepos% change it.
2109 \addto@thismodule{x\@noexpand\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}}%
2110 \fi%
2111 \importmodule[\MathHub{\mh@currentrepos/source/\@path}] {#2}{export}%
2112 \mathhub@setcurrentreposinfo\mh@@repos% after importing, reset to old value
2113 \addto@thismodule{x\@noexpand\mathhub@setcurrentreposinfo{\mh@@repos}}%
2114 \fi%

```

```

2115   %\ignorespacesandpars%
2116 }

\usemhmodule

2117 \addmetakey{importmhmodule}{load}
2118 \addmetakey{importmhmodule}{id}
2119 \addmetakey{importmhmodule}{dir}
2120 \addmetakey{importmhmodule}{mhrepos}
2121
2122 \addmetakey{importmodule}{load}
2123 \addmetakey{importmodule}{id}
2124
2125 \newcommand\usemhmodule[2] [] {%
2126   \metasetkeys{importmhmodule}{#1}%
2127   \ifx\importmhmodule@dir\@empty%
2128     \edef\@path{\importmhmodule@path}%
2129   \else\edef\@path{\importmhmodule@dir/#2}\fi%
2130   \ifx\@path\@empty%
2131     \usemodule[id=\importmhmodule@id]{#2}%
2132   \else%
2133     \edef\mh@@repos{\mh@currentrepos}%
2134     \ifx\importmhmodule@mhrepos\@empty%
2135     \else\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
2136     \usemodule{\@path\@QuestionMark#2}%
2137     %\usemodule[load=\MathHub{\mh@currentrepos/source/\@path},
2138     %                                id=\importmhmodule@id]{#2}%
2139     \mathhub@setcurrentreposinfo\mh@@repos%
2140   \fi%
2141   %\ignorespacesandpars
2142 }

\mhinputref

2143 \newcommand\mhinputref[2] [] {%
2144   \edef\mhinputref@first{#1}%
2145   \ifx\mhinputref@first\@empty%
2146     \inputref{#2}%
2147   \else%
2148     \inputref[mhrepos=\mhinputref@first]{#2}%
2149   \fi%
2150 }

\trefi*

2151 \newcommand\trefi[2] [] {%
2152   \edef\trefi@mod{#1}%
2153   \ifx\trefi@mod\@empty\tref{#2}\else\tref{#1\@QuestionMark#2}\fi%
2154 }
2155 \newcommand\trefii[3] [] {%
2156   \edef\trefi@mod{#1}%
2157   \ifx\trefi@mod\@empty\tref{#2-#3}\else\tref{#1\@QuestionMark#2-#3}\fi%
2158 }

```

```

\defi*
2159 \def\defii#1#2{\defi{#1!#2}}
2160 \def\Defii#1#2{\Defi{#1!#2}}
2161 \def\defiis#1#2{\defis{#1!#2}}
2162 \def\Defiis#1#2{\Defis{#1!#2}}
2163 \def\defiii#1#2#3{\defi{#1!#2!#3}}
2164 \def\Defiii#1#2#3{\Defi{#1!#2!#3}}
2165 \def\defiiis#1#2#3{\defis{#1!#2!#3}}
2166 \def\Defiiis#1#2#3{\Defis{#1!#2!#3}}
2167 \def\defiv#1#2#3#4{\defi{#1!#2!#3!#4}}
2168 \def\Defiv#1#2#3#4{\Defi{#1!#2!#3!#4}}
2169 \def\defivs#1#2#3#4{\defis{#1!#2!#3!#4}}
2170 \def\Defivs#1#2#3#4{\Defis{#1!#2!#3!#4}}
2171 \def\adefi#1#2{\defi[name=#2]{#1}}
2172 \def\adefii#1#2#3{\defi[name=#2-#3]{#1}}
2173 \def\adefiii#1#2#3#4{\defi[name=#2-#3-#4]{#1}}
2174 \def\adefiv#1#2#3#4#5{\defi[name=#2-#3-#4-#5]{#1}}

```