

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

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

March 18, 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 . . . . .	16
3.4	Inheritance . . . . .	21
3.5	Symbols/Notations/Verbalizations . . . . .	31
3.6	Term References . . . . .	44
3.7	sref . . . . .	46
3.8	smultiling . . . . .	49
3.9	smglom . . . . .	49
3.10	mathhub . . . . .	50
3.11	omdoc/omgroup . . . . .	50
3.12	omtext . . . . .	53
<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 \ifcsname if@latexml\endcsname\else
20 \ex\newif\csname if@latexml\endcsname\@latexmlfalse
21 \fi

```

The following macro and environment generate LaTeXML annotations as a `<span>` node with the first and second arguments as `property` and `resource` attributes respectively, and the third argument as content. In math mode, the first two arguments are instead used as the `class` attribute, separated by an underscore.

```

22 \def\latexml@annotate#1#2#3{\ifmmode\latexml@annotate@math{#1}{#2}{#3}\else\latexml@annotate@text{#1}{#2}{#3}\fi}
23 \def\latexml@annotate@text#1#2#3{}
24 \def\latexml@annotate@math#1#2#3{}
25 \newenvironment{latexml@annotateenv}[2]{}{}
26 \RequirePackage{xspace}
27 \RequirePackage{standalone}
28 \RequirePackageWithOptions{stex-metakeys}
29 \RequirePackage{xstring}
30 \RequirePackage{etoolbox}

```

### 3.1 sTeX base

The sTeX logo:

```

31 \protected\def\stex{%
32   \@ifundefined{texorpdfstring}%
33   {\let\texorpdfstring\@firstoftwo}%
34   {}%
35   \texorpdfstring{\raisebox{-.5ex}{S}\kern-.5ex\TeX}{sTeX}\xspace%
36 }
37 \def\sTeX{\stex}

```

### 3.2 Paths and URIs

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

```

38 \def\pathsuris@setcatcodes{%
39   \edef\pathsuris@oldcatcode@hash{\the\catcode'\#}%
40   \catcode'\#=12\relax%
41   \edef\pathsuris@oldcatcode@slash{\the\catcode'\/%}%
42   \catcode'\/=12\relax%
43   \edef\pathsuris@oldcatcode@colon{\the\catcode'\:%}%
44   \catcode'\:=12\relax%
45   \edef\pathsuris@oldcatcode@qm{\the\catcode'\?}%
46   \catcode'\?=12\relax%
47 }
48 \def\pathsuris@resetcatcodes{%
49   \catcode'\#\pathsuris@oldcatcode@hash\relax%
50   \catcode'\/>\pathsuris@oldcatcode@slash\relax%
51   \catcode'\:\pathsuris@oldcatcode@colon\relax%
52   \catcode'\?\pathsuris@oldcatcode@qm\relax%
53 }

```

`\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`.

```
54 \def\namespace@read#1{%
55   \edef\namespace@read@path{#1}%
56   \edef\namespace@read@path{\ex\detokenize\ex\namespace@read@path}}%
57   \namespace@continue%
58 }
59 \def\namespace@continue{%
60   \pathsuris@resetcatcodes%
61   \ex\edef\csname\namespace@macroname\endcsname##1{%
62     \namespace@read@path\@Slash##1%
63   }%
64 }
65 \protected\def\namespace#1{%
66   \def\namespace@macroname{#1}%
67   \pathsuris@setcatcodes%
68   \namespace@read%
69 }
70 \let\defpath\namespace
```

### 3.2.1 Path Canonicalization

We define some macros for later comparison.

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

```

90 \let\@Percent%
91 \catcode'\%=\old@percent@catcode

\@cpath Canonicalizes (file) paths:
92 \def\@cpath#1{%
93   \edef\pathsuris@cpath@temp{#1}%
94   \def\@cpath@path{}%
95   \IfBeginWith\pathsuris@cpath@temp\@Slash{%
96     \@cpath@loop%
97     \edef\@cpath@path{\@Slash\@cpath@path}%
98   }{%
99     \IfBeginWith\pathsuris@cpath@temp\@Dot\@Slash{%
100       \StrGobbleLeft\pathsuris@cpath@temp2[\pathsuris@cpath@temp]%
101       \@cpath@loop%
102     }{%
103       \ifx\pathsuris@cpath@temp\@Dot\else%
104       \@cpath@loop\fi%
105     }%
106   }%
107   \IfEndWith\@cpath@path\@Slash{%
108     \ifx\@cpath@path\@Slash\else%
109     \StrGobbleRight\@cpath@path1[\@cpath@path]%
110     \fi%
111   }{}%
112 }
113
114 \def\@cpath@loop{%
115   \IfSubStr\pathsuris@cpath@temp\@Slash{%
116     \StrCut\pathsuris@cpath@temp\@Slash%
117     \pathsuris@cpath@temp@a\pathsuris@cpath@temp%
118     \ifx\pathsuris@cpath@temp@a\@ToTop%
119       \ifx\@cpath@path\@empty%
120         \edef\@cpath@path{\@ToTop}%
121       \else%
122         \edef\@cpath@path{\@cpath@path\@Slash\@ToTop}%
123       \fi%
124       \@cpath@loop%
125     \else%
126     \ifx\pathsuris@cpath@temp@a\@Dot%
127       \@cpath@loop%
128     \else%
129     \IfBeginWith\pathsuris@cpath@temp\@ToTop{%
130       \StrBehind{\pathsuris@cpath@temp}{\@ToTop}%
131       [\pathsuris@cpath@temp]%
132       \IfBeginWith\pathsuris@cpath@temp\@Slash{%
133         \edef\pathsuris@cpath@temp%
134         {\@cpath@path\pathsuris@cpath@temp}%
135       }{%
136         \ifx\@cpath@path\@empty\else%
137           \edef\pathsuris@cpath@temp%

```

```

138             {\cpath@path\@Slash\pathsuris@cpath@temp}%
139         \fi%
140     }%
141     \def\@cpath@path{}%
142     \@cpath@loop%
143 }{%
144     \ifx\@cpath@path\@empty%
145         \edef\@cpath@path{\pathsuris@cpath@temp@a}%
146     \else%
147         \edef\@cpath@path%
148             {\@cpath@path\@Slash\pathsuris@cpath@temp@a}%
149     \fi%
150     \@cpath@loop%
151 }%
152 \fi\fi%
153 }{%
154     \ifx\@cpath@path\@empty%
155         \edef\@cpath@path{\pathsuris@cpath@temp}%
156     \else%
157         \edef\@cpath@path{\@cpath@path\@Slash\pathsuris@cpath@temp}%
158     \fi%
159 }%
160 }

```

#### 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.

```

161 \newcommand\cpath@print[1]{%
162     \@cpath{#1}%
163     \@cpath@path%
164 }

```

\path@filename

```

165 \def\path@filename#1#2{%
166     \edef\filename@oldpath{#1}%

```

```

167 \StrCount\filename@oldpath\@Slash[\filename@lastslash]%
168 \ifnum\filename@lastslash>0%
169 \StrBehind[\filename@lastslash]\filename@oldpath%
170 \@Slash[\filename@oldpath]%
171 \edef#2{\filename@oldpath}%
172 \else%
173 \edef#2{\filename@oldpath}%
174 \fi%
175 }

```

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

\path@filename@noext

```

176 \def\path@filename@noext#1#2{%
177 \path@filename{#1}{#2}%
178 \edef\filename@oldpath{#2}%
179 \StrCount\filename@oldpath\@Dot[\filename@lastdot]%
180 \ifnum\filename@lastdot>0%
181 \StrBefore[\filename@lastdot]\filename@oldpath%
182 \@Dot[\filename@oldpath]%
183 \edef#2{\filename@oldpath}%
184 \else%
185 \edef#2{\filename@oldpath}%
186 \fi%
187 }

```

**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:

```

188 \newif\if@iswindows@\@iswindows@false
189 \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:

```

190 \newif\if@windowstopath@inpath@
191 \def\windows@to@path#1{%
192 \@windowstopath@inpath@false%
193 \def\windows@temp{}%
194 \edef\windows@path{#1}%
195 \ifx\windows@path\empty\else%
196 \ex\windows@path@loop\windows@path\windows@path@end%

```



```

197     \fi%
198     \let#1\windows@temp%
199 }
200 \def\windows@path@loop#1#2\windows@path@end{%
201     \def\windows@temp@b{#2}%
202     \ifx\windows@temp@b\@empty%
203         \def\windows@continue{}}%
204     \else%
205         \def\windows@continue{\windows@path@loop#2\windows@path@end}%
206     \fi%
207     \if@windowstopath@inpath@%
208         \ifx#1\@BackSlash%
209             \edef\windows@temp{\windows@temp\@Slash}%
210         \else%
211             \edef\windows@temp{\windows@temp#1}%
212         \fi%
213     \else%
214         \ifx#1:%
215             \edef\windows@temp{\@Slash\windows@temp}%
216             \@windowstopath@inpath@true%
217         \else%
218             \edef\windows@temp{\windows@temp#1}%
219         \fi%
220     \fi%
221     \windows@continue%
222 }

```

**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:

```

223 \def\path@to@windows#1{%
224     \@windowstopath@inpath@false%
225     \def\windows@temp{}%
226     \edef\windows@path{#1}%
227     \edef\windows@path{\expandafter\@gobble\windows@path}%
228     \ifx\windows@path\@empty\else%
229         \expandafter\path@windows@loop\windows@path\windows@path@end%
230     \fi%
231     \let#1\windows@temp%
232 }
233 \def\path@windows@loop#1#2\windows@path@end{%
234     \def\windows@temp@b{#2}%
235     \ifx\windows@temp@b\@empty%
236         \def\windows@continue{}}%
237     \else%
238         \def\windows@continue{\path@windows@loop#2\windows@path@end}%
239     \fi%
240     \if@windowstopath@inpath@%

```

```

241     \ifx#1/%
242         \edef\windows@temp{\windows@temp\@BackSlash}%
243     \else%
244         \edef\windows@temp{\windows@temp#1}%
245     \fi%
246 \else%
247     \ifx#1/%
248         \edef\windows@temp{\windows@temp:\@BackSlash}%
249         \@windowstopath@inpath@true%
250     \else%
251         \edef\windows@temp{\windows@temp#1}%
252     \fi%
253 \fi%
254 \windows@continue%
255 }

```

**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:

```

256 \def\path@trimstring#1{%
257     \edef\pathsuris@trim@temp{#1}%
258     \IfBeginWith\pathsuris@trim@temp\@Space{%
259         \StrGobbleLeft\pathsuris@trim@temp1[#1]%
260         \path@trimstring{#1}%
261     }{%
262         \IfEndWith\pathsuris@trim@temp\@Space{%
263             \StrGobbleRight\pathsuris@trim@temp1[#1]%
264             \path@trimstring{#1}%
265         }{%
266             \edef#1{\pathsuris@trim@temp}%
267         }%
268     }%
269 }

```

**Test 7:** >foo bar<

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

```

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

```

```

279 \endgroup}
280 %\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:

```

281 \edef\pwd@cmd{\if@iswindows@ -expand-var \@Percent%
282   CD\@Percent\else -var-value PWD\fi}
283 \@kpsewhich\stex@PWD\pwd@cmd
284 \edef\stex@mainfile{\stex@PWD\@Slash\jobname}
285 \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:

```

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

```

```

318     \global\let\stex@currfilename\stex@currfilename%
319   \fi%
320 }

```

**\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:

```

321 \def\stexinput#1{%
322   \stex@iffileexists{#1}{%
323     \stex@currfile@push\stex@temp@path%
324     \input{\stex@currfile}%
325     \stex@currfile@pop%
326   }%
327   {%
328     \PackageError{stex}{File does not exist %
329       (#1): \stex@temp@path}{}%
330   }%
331 }
332 \def\stex@iffileexists#1#2#3{%
333   \edef\stex@temp@path{#1}%
334   \if@iswindows@\path@to@windows\stex@temp@path\fi%
335   \IfFileExists\stex@temp@path{#2}{#3}%
336 }
337 \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:

```

338 \@kpsewhich\mathhub@path{--var-value MATHHUB}
339 \if@iswindows@\windows@to@path\mathhub@path\fi
340 \ifx\mathhub@path\@empty
341   \PackageWarning{stex}{MATHHUB system variable not %
342     found or wrongly set}{%
343     \defpath{MathHub}{%
344     \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.

```

345 \def\mathhub@findmanifest#1{%
346   \@cpath{#1}%
347   \ifx\@cpath@path\@Slash%
348     \def\manifest@mf{}%
349   \else\ifx\@cpath@path\@empty%

```

```

350     \def\manifest@mf{}%
351 \else%
352   \edef\@findmanifest@path{\@cpath@path/MANIFEST.MF}%
353   \if@iswindows@ \path@to@windows \@findmanifest@path\fi%
354   \IfFileExists{\@findmanifest@path}{%
355     \edef\manifest@mf{\@findmanifest@path}%
356     \xdef\temp@archive@dir{\ex\detokenize\ex{\@cpath@path}}%
357   }{%
358     \edef\@findmanifest@path{\@cpath@path/META-INF/MANIFEST.MF}%
359     \if@iswindows@ \path@to@windows \@findmanifest@path\fi%
360     \IfFileExists{\@findmanifest@path}{%
361       \edef\manifest@mf{\@findmanifest@path}%
362       \xdef\temp@archive@dir{\ex\detokenize\ex{\@cpath@path}}%
363     }{%
364       \edef\@findmanifest@path{\@cpath@path/meta-inf/MANIFEST.MF}%
365       \if@iswindows@ \path@to@windows \@findmanifest@path\fi%
366       \IfFileExists{\@findmanifest@path}{%
367         \edef\manifest@mf{\@findmanifest@path}%
368         \xdef\temp@archive@dir{\ex\detokenize\ex{\@cpath@path}}%
369       }{%
370         \mathhub@findmanifest{\@cpath@path/..}%
371       }}}}
372 \fi\fi%
373 }

```

**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

```

374 \def\split@manifest@key{%
375   \IfSubStr{\manifest@line}{\@Colon}{%
376     \StrBefore{\manifest@line}{\@Colon}[\manifest@key]%
377     \StrBehind{\manifest@line}{\@Colon}[\manifest@line]%
378     \path@trimstring\manifest@line%
379     \path@trimstring\manifest@key%
380   }{%
381     \def\manifest@key{}%
382   }%
383 }

```

the next helper function iterates over lines in MANIFEST.MF

```

384 \def\parse@manifest@loop{%
385   \ifeof\@manifest%
386   \else%
387     \read\@manifest to \manifest@line\relax%
388     \split@manifest@key%
389     % id
390     \IfStrEq\manifest@key{id}{%
391       \xdef\manifest@mf{id\manifest@line}%

```

```

392   }{%
393   % narration-base
394   \IfStrEq\manifest@key{narration-base}{%
395       \xdef\manifest@mf@narr{\manifest@line}%
396   }{%
397   % namespace
398   \IfStrEq\manifest@key{source-base}{%
399       \xdef\manifest@mf@ns{\manifest@line}%
400   }{%
401   \IfStrEq\manifest@key{ns}{%
402       \xdef\manifest@mf@ns{\manifest@line}%
403   }{%
404   % dependencies
405   \IfStrEq\manifest@key{dependencies}{%
406       \xdef\manifest@mf@deps{\manifest@line}%
407   }{%
408   }}}}}%
409   \parse@manifest@loop%
410   \fi%
411 }

```

`\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.

```

412 \newread\@manifest
413 \def\mathhub@parsemanifest#1#2{%
414     \gdef\temp@archive@dir{}%
415     \mathhub@findmanifest{#2}%
416     \begingroup%
417         \newlinechar=-1%
418         \endlinechar=-1%
419         \gdef\manifest@mf@id{}%
420         \gdef\manifest@mf@narr{}%
421         \gdef\manifest@mf@ns{}%
422         \gdef\manifest@mf@deps{}%
423         \immediate\openin\@manifest=\manifest@mf\relax%
424         \parse@manifest@loop%
425         \immediate\closein\@manifest%
426     \endgroup%
427     \if@iswindows@ \windows@to@path\manifest@mf\fi%
428     \cslet{#1id}\manifest@mf@id%
429     \cslet{#1narr}\manifest@mf@narr%
430     \cslet{#1ns}\manifest@mf@ns%
431     \cslet{#1deps}\manifest@mf@deps%
432     \ifcsvoid\manifest@mf@id{}{%
433         \cslet{#1dir}\temp@archive@dir%
434     }%
435 }

```

**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.
436 \def\mathhub@setcurrentreposinfo#1{%
437   \edef\mh@currentrepos{#1}%
438   \ifx\mh@currentrepos\empty%
439     \edef\currentrepos@dir{\@Dot}%
440     \def\currentrepos@narr{%
441       \def\currentrepos@ns{%
442         \def\currentrepos@id{%
443           \def\currentrepos@deps{%
444             \else%
445             \ifcsdef{mathhub@dir@\mh@currentrepos}{%
446               \@inmhrepostrue
447               \ex\let\ex\currentrepos@dir\csname mathhub@dir@#1\endcsname%
448               \ex\let\ex\currentrepos@narr\csname mathhub@narr@#1\endcsname%
449               \ex\let\ex\currentrepos@ns\csname mathhub@ns@#1\endcsname%
450               \ex\let\ex\currentrepos@deps\csname mathhub@deps@#1\endcsname%
451             }{%
452               \mathhub@parsemanifest{currentrepos@}\MathHub{#1}}%
453             \@setcurrentreposinfo%
454             \ifcsvoid{currentrepos@dir}\PackageError{stex}{No archive with %
455               name #1 found!}{make sure that #1 is directly in your MATHHUB folder %
456               and contains a MANIFEST.MF, either directly in #1 or in a meta-inf %
457               subfolder.}}{\@inmhrepostrue}%
458           }%
459         \fi%
460       }
461     }
462   \def\@setcurrentreposinfo{%
463     \edef\mh@currentrepos{\currentrepos@id}%
464     \ifcsvoid{currentrepos@dir}{%
465       \csxdef{mathhub@dir@\currentrepos@id}{\currentrepos@dir}%
466       \csxdef{mathhub@narr@\currentrepos@id}{\currentrepos@narr}%
467       \csxdef{mathhub@ns@\currentrepos@id}{\currentrepos@ns}%
468       \csxdef{mathhub@deps@\currentrepos@id}{\currentrepos@deps}%
469     }%
470   }
```

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

```
471 \newif\if@inmhrepos\@inmhreposfalse
472 \ifcsvoid{stex@PWD}{%
473   \mathhub@parsemanifest{currentrepos@}\stex@PWD
474   \@setcurrentreposinfo
475   \ifcsvoid{currentrepos@dir}{\message{sTeX: Not currently in a MathHub repository}}{%
```

```

476 \message{Current sTeX repository: \mh@currentrepos}
477 }
478 }

```

### 3.3 Modules

```

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

Aux:
480 %\def\ignorespacesandpars{\begingroup\catcode13=10%
481 % \ifnextchar\relax{\endgroup}{\endgroup}}

and more adapted from http://tex.stackexchange.com/questions/179016/
ignore-spaces-and-pars-after-an-environment
482 %\def\ignorespacesandparsafterend#1\ignorespaces\fi{#1%
483 % \fi\ignorespacesandpars}
484 %\def\ignorespacesandpars{\ifhmode\unskip\fi\ifnextchar\par%
485 % {\ex\ignorespacesandpars\@gobble}{}}

Options for the module-environment:
486 \addmetakey*{module}{title}
487 \addmetakey*{module}{name}
488 \addmetakey*{module}{creators}
489 \addmetakey*{module}{contributors}
490 \addmetakey*{module}{srccite}
491 \addmetakey*{module}{ns}
492 \addmetakey*{module}{narr}

module@heading We make a convenience macro for the module heading. This can be customized.
493 \ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%
494 \newrobustcmd\module@heading{%
495 \stepcounter{module}%
496 \ifmod@show%
497 \noindent{\textbf{Module} \thesection.\thetitle [\module@name]}%
498 \sref@label{id{Module \thesection.\thetitle [\module@name]}}%
499 \ifx\module@title@empty :\quad\else\quad(\module@title)\hfill\\\fi%
500 \fi%
501 }%

```

**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.
502 \newenvironment{module}[1] [] {%
503 \begin{@module}[#1]%
504 \module@heading% make the headings
505 %\ignorespacesandpars
506 \parsemodule@maybesetcodes}%
507 \end{@module}%
508 \ignorespacesafterend%
509 }%

```



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

Some auxiliary methods:

```
511 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}
512 \def\addto@thismodule#1{%
513   \ifundefined{this@module}{}{%
514     \expandafter\g@addto@macro@safe\this@module{#1}%
515   }%
516 }
517 \def\addto@thismodulex#1{%
518   \ifundefined{this@module}{}{%
519     \edef\addto@thismodule@exp{#1}%
520     \expandafter\expandafter\expandafter\g@addto@macro@safe%
521     \expandafter\this@module\expandafter{\addto@thismodule@exp}%
522 }}
```

**@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`.

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

```

545 \fi%
546 \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]}%
547 \setkeys{module}{ns=\@module@ns@tempuri}%
548 }

```

**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.

```

549 \def\set@next@moduleid{%
550 \path@filename@noext\stex@currfile\stex@next@moduleid@filename%
551 \edef\set@next@moduleid@csname{namespace@\module@ns\@QuestionMark\stex@next@moduleid@filename}%
552 \unless\ifcsname\set@next@moduleid@csname\endcsname%
553 \csgdef{\set@next@moduleid@csname}{0}%
554 \fi%
555 \edef\namespace@currnum{\csname\set@next@moduleid@csname\endcsname}%
556 \edef\module@temp@setidname{\noexpand\setkeys{module}{name=%
557 \stex@next@moduleid@filename\ex\unless\ex\ifnum\csname\set@next@moduleid@csname\endcsname=0.
558 \module@temp@setidname%
559 \csxdef{\set@next@moduleid@csname}{\the\numexpr\namespace@currnum+1}%
560 }

```

**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.

```

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

```

```

611 }{%
612   \end{latexml@module}%
613   \if@inmhrepos%
614   \@inmhreposfalse%
615   \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\expandafter\noexpand\curname mh@old@
616   \fi%
617 }%
618 % For LaTeXML bindings
619 \newenvironment{latexml@module}[1]{\begin{latexml@annotateenv}{stex:theory}{#1}}{\end{latexml@ar

```

**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.

```

620 \def\@URI{uri} % TODO check this
621 \def\@invoke@module#1#2{%
622   \ifx\@URI#2%
623     #1%
624   \else%
625     % TODO something else
626     #2%

```

```

627 \fi%
628 }

```

## 3.4 Inheritance

### 3.4.1 Selective Inclusion

The next great goal is to establish the `\requiremodules` macro, which reads an `STEX` 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.
629 \newif\if@smsmode\@smsmodefalse
630 \def\parsemodule@allow#1{%
631   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#1\endcsname}%
632 }
633 \def\parsemodule@allowenv#1{%
634   \ex\def\csname parsemodule@allowedenv@#1\endcsname{#1}%
635 }
636 \def\parsemodule@replacemacro#1#2{%
637   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#2\endcsname}%
638 }
639 \def\parsemodule@replaceenv#1#2{%
640   \ex\def\csname parsemodule@allowedenv@#1\endcsname{#2}%
641 }
642 \def\parsemodule@escapechar@beginstring{begin}
643 \def\parsemodule@escapechar@endstring{end}

```

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

```

644 \parsemodule@allow{symdef}
645 \parsemodule@allow{abbrdef}
646 \parsemodule@allow{importmodule}
647 \parsemodule@allowenv{module}
648 \parsemodule@allowenv{@module}
649 \parsemodule@allow{importmhmodule}
650 \parsemodule@allow{gimport}
651 \parsemodule@allowenv{modsig}
652 \parsemodule@allowenv{mhmodsig}
653 \parsemodule@allowenv{mhmodnl}
654 \parsemodule@allowenv{modnl}
655 \parsemodule@allow{symvariant}
656 \parsemodule@allow{symi}
657 \parsemodule@allow{symii}

```

```

658 \parsemodule@allow{symiii}
659 \parsemodule@allow{symiv}
660 \parsemodule@allow{notation}
661 \parsemodule@allow{symdecl}
662
663 % to deprecate:
664
665 \parsemodule@allow{defi}
666 \parsemodule@allow{defii}
667 \parsemodule@allow{defiii}
668 \parsemodule@allow{defiv}
669 \parsemodule@allow{adefi}
670 \parsemodule@allow{adefii}
671 \parsemodule@allow{adefiii}
672 \parsemodule@allow{adefiv}
673 \parsemodule@allow{defis}
674 \parsemodule@allow{defiis}
675 \parsemodule@allow{defiiis}
676 \parsemodule@allow{defivs}
677 \parsemodule@allow{Defi}
678 \parsemodule@allow{Defii}
679 \parsemodule@allow{Defiii}
680 \parsemodule@allow{Defiv}
681 \parsemodule@allow{Defis}
682 \parsemodule@allow{Defiis}
683 \parsemodule@allow{Defiiis}
684 \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*).

```

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

```

```

698 \catcode'\<=12
699 \catcode'\>=12

```

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

`\set@parsemodule@catcodes`

```

700 \def\parsemodule@ignorepackageerrors{,inputenc,}
701 \let\parsemodule@old@PackageError\PackageError
702 \def\parsemodule@packageerror#1#2#3{%
703   \IfSubStr\parsemodule@ignorepackageerrors{,#1,}{%}{%
704     \parsemodule@old@PackageError{#1}{#2}{#3}%
705   }%
706 }
707 \def\set@parsemodule@catcodes{%
708   \ifcat'\=0%
709     \global\catcode'\=13%
710     \global\catcode'\#=12%
711     \global\catcode'\{=12%
712     \global\catcode'\}=12%
713     \global\catcode'\$=12%$
714     \global\catcode'\~=12%
715     \global\catcode'\_ =12%
716     \global\catcode'\&=12%
717     \ex\global\ex\let\@active@slash\parsemodule@escapechar%
718     \global\let\parsemodule@old@PackageError\PackageError%
719     \global\let\PackageError\parsemodule@packageerror%
720     \fi%
721 }

```

`\reset@parsemodule@catcodes`

```

722 \def\reset@parsemodule@catcodes{%
723   \ifcat'\=13%
724     \global\catcode'\=0%
725     \global\catcode'\#=6%
726     \global\catcode'\{=1%
727     \global\catcode'\}=2%
728     \global\catcode'\$=3%$
729     \global\catcode'\~=7%
730     \global\catcode'\_ =8%
731     \global\catcode'\&=4%
732     \global\let\PackageError\parsemodule@old@PackageError%
733     \fi%
734 }

```

`\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.

```

735 \def\parsemodule@maybesetcodes{%

```

```

736 \if@smsmode\set@parsemodule@catcodes\fi%
737 }

```

`\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:

```

738
739 \def\parsemodule@escapechar{%
740   \def\parsemodule@escape@currcls{%
741     \parsemodule@escape@parse@nextchar@%
742   }%

```

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`.

```

743 \long\def\parsemodule@escape@parse@nextchar@#1{%
744   \ifcat a#1\relax%
745     \edef\parsemodule@escape@currcls{\parsemodule@escape@currcls#1}%
746     \let\parsemodule@do@next\parsemodule@escape@parse@nextchar@%
747   \else%
748     \def\parsemodule@last@char{#1}%
749     \ifx\parsemodule@escape@currcls\@empty%
750       \def\parsemodule@do@next{%
751         \else%
752         \def\parsemodule@do@next{\parsemodule@escapechar@checkcls}%
753       \fi%
754     \fi%
755     \parsemodule@do@next%
756 }

```

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.

```

757 \def\parsemodule@escapechar@checkcls{%
758   \ifx\parsemodule@escape@currcls\parsemodule@escapechar@beginstring%

```



```

759     \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@la
760 \else%
761     \ifx\parsemodule@escape@currcls\parsemodule@escapechar@endstring%
762     \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@la
763 \else%
764     \ifcsvoid{parsemodule@allowedmacro@\parsemodule@escape@currcls}{%
765     \def\parsemodule@do@next{\relax\parsemodule@last@char}%
766     }{%
767     \ifx\parsemodule@last@char\@open@brace%
768     \ex\let\ex\parsemodule@do@next@ii\csname parsemodule@allowedmacro@\parsemodule@
769     \edef\parsemodule@do@next{\noexpand\parsemodule@converttoproperbraces\@open@bra
770     \else%
771     \reset@parsemodule@catcodes%
772     \edef\parsemodule@do@next{\ex\noexpand\csname parsemodule@allowedmacro@\parsemo
773     \fi%
774     }%
775     \fi%
776 \fi%
777 \parsemodule@do@next%
778 }

```

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.

```

779 \ex\ex\ex\def%
780 \ex\ex\ex\parsemodule@converttoproperbraces%
781 \ex\@open@brace\ex#\ex1\@close@brace{%
782 \reset@parsemodule@catcodes%
783 \parsemodule@do@next@ii{#1}%
784 }

```

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.

```

785 \ex\ex\ex\def%
786 \ex\ex\ex\parsemodule@escapechar@checkbeginenv%
787 \ex\@open@brace\ex#\ex1\@close@brace{%
788 \ifcsvoid{parsemodule@allowedenv@#1}{%
789 \def\parsemodule@do@next{#1}%
790 }{%
791 \reset@parsemodule@catcodes%
792 \edef\parsemodule@envname{\csname parsemodule@allowedenv@#1\endcsname}%
793 \ex\def\ex\parsemodule@do@next\ex{%
794 \ex\begin\ex{\parsemodule@envname}%
795 }%

```

```

796     }%
797     \parsemodule@do@next%
798 }
799 \ex\ex\ex\def%
800 \ex\ex\ex\parsemodule@escapechar@checkendenv%
801 \ex\@open@brace\ex#\ex1\@close@brace{%
802   \ifcsvoid{parsemodule@allowedenv@#1}{%
803     \def\parsemodule@do@next{#1}%
804   }{%
805     \edef\parsemodule@envname{\csname parsemodule@allowedenv@#1\endcsname}%
806     \ex\def\ex\parsemodule@do@next\ex{%
807       \ex\end\ex{\parsemodule@envname}%
808     }%
809   }%
810   \parsemodule@do@next%
811 }

```

`\@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.

```

812 \newrobustcmd\@requiremodules[1]{%
813   \if@tempswa\requiremodules{#1}\fi%
814 }%

```

`\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.

```

815 \newrobustcmd\requiremodules[1]{%
816   \mod@showfalse%
817   \edef\mod@path{#1}%
818   \edef\mod@path{\expandafter\detokenize\expandafter{\mod@path}}%
819   \requiremodules@smsmode{#1}%
820 }%

```

`\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.

```

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

```

```

832     }%
833     \parsemodule@maybesetcodes%
834 }

```

**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

```

835 \newif\if@importmodule@switchrepos\@importmodule@switchreposfalse
836 \def\importmodule@bookkeeping#1#2#3{%
837   \@importmodule@switchreposfalse%
838   \stex@debug{Importmodule: #1^^J #2^^J\detokenize{#3}}%
839   \metasetkeys{importmodule}{#1}%
840   \ifcsvoid{importmodule@mhrepos}{%
841     \ifcsvoid{currentrepos@dir}{%
842       \stex@debug{Importmodule: Set importmodule@dir to \stex@PWD}%
843       \let\importmodule@dir\stex@PWD%
844     }{%
845       \stex@debug{Importmodule: Set importmodule@dir to \currentrepos@dir\@Slash source}%
846     }%
847   }%
848   \if@importmodule@switchreposfalse%
849     \stex@debug{Importmodule: Repository switch to \importmodule@mhrepos}%
850     \stex@debug{Importmodule: Current repos: \mh@currentrepos}%
851     \ex\let\csname importmodule@oldrepos@#2\endcsname\mh@currentrepos%
852     \mathhub@setcurrentreposinfo\importmodule@mhrepos%
853     \stex@debug{Importmodule: New repos: \mh@currentrepos^^J Namespace: \currentrepos@ns}%
854     \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
855   }%
856   \StrCut{#2}\@QuestionMark\importmodule@subdir\importmodule@modulename%
857   \ifx\importmodule@modulename\@empty%
858     \let\importmodule@modulename\importmodule@subdir%
859     \let\importmodule@subdir\@empty%
860   \else%
861     \ifx\importmodule@subdir\@empty\else%
862       \edef\importmodule@dir{\importmodule@dir\@Slash\importmodule@subdir}%
863     \fi%
864   \fi%
865   \fi%
866   #3%
867   \if@importmodule@switchrepos%
868     \ex\mathhub@setcurrentreposinfo\csname importmodule@oldrepos@#2\endcsname%
869     \stex@debug{Importmodule: switched back to: \mh@currentrepos}%
870   \fi%
871   %\ignorespacesandpars%
872 }

```

\importmodule

```

873 %\srefaddidkey{importmodule}
874 \addmetakey{importmodule}{mhrepos}
875 \newcommand\importmodule[2][\@importmodule[#1]{#2}{export}}
876 \newcommand\@importmodule[3][\@%
877   \importmodule@bookkeeping{#1}{#2}{%
878     \@importmodule[\importmodule@dir]\importmodule@modulename{#3}%
879   }%
880 }

```

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

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

```

881 \newcommand\@importmodule[3][\@%
882   {%
883     \edef\@load{#1}%
884     \edef\@importmodule@name{#2}%
885     \stex@debug{Loading #1}%
886     \if@smsmode\else\ifcsvoid{stex@module@\@importmodule@name}{% TODO check this
887       \stex@iffileexists\@load{
888         \stex@debug{Exists: #1}%
889         \requiremodules\@load}{%
890         \stex@debug{Does not exist: #1^^Trying \@load\@Slash\@importmodule@name}%
891         \requiremodules{\@load\@Slash\@importmodule@name}%
892       }%
893     }\fi%
894     \ifx\@load\@empty\else%
895       {% TODO
896         % \edef\@path{\csname module@#2@path\endcsname}%
897         % \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do no
898         % {\PackageError{stex}% else signal an error
899         %   {Module Name Clash\MessageBreak%
900         %     A module with name #2 was already loaded under the path "\@path"\MessageBreak%
901         %     The imported path "\@load" is probably a different module with the\MessageBreak%
902         %     same name; this is dangerous -- not importing}%
903         %   {Check whether the Module name is correct}%
904         %   }%
905       }%
906     \fi%
907     \global\let\@importmodule@load\@load%
908   }%
909   \edef\@export{#3}\def\@export{export}%prepare comparison
910   %\ifx\@export\@export\export@defs{#2}\fi% export the module
911   \ifx\@export\@export\addto@thismodulex{%

```

```

912 \noexpand\@importmodule[\@importmodule@load]{#2}{noexport}%
913 }%
914 \if@smsmode\else
915 \ifcsvoid{this@module}{}%
916 \ifcsvoid{module@imports@\module@uri}{
917 \csxdef{module@imports@\module@uri}{%
918 \csname stex@module@#2\endcsname\@URI% TODO check this
919 }%
920 }{%
921 \csxdef{module@imports@\module@uri}{%
922 \csname stex@module@#2\endcsname\@URI,% TODO check this
923 \csname module@imports@\module@uri\endcsname%
924 }%
925 }%
926 }%
927 \fi\fi%
928 \if@smsmode\else%
929 \edef\activate@module@name{#2}%
930 \StrCount\activate@module@name\@Slash[\activate@module@lastslash]%
931 \ifnum\activate@module@lastslash>0%
932 \StrCut[\activate@module@lastslash]\activate@module@name\@Slash\activate@module@temp\activa
933 \fi%
934 \ifcsvoid{stex@lastmodule@\activate@module@name}{%
935 \PackageError{stex}{No module with name \activate@module@name found}{}%
936 }{%
937 \ex\ex\ex\activate@defs\ex\ex\ex{\csname stex@lastmodule@\activate@module@name\endcsname}%
938 }%
939 \fi activate the module
940 }%

```

**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?type  
 {\@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}}\br/>
 {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:

```

941 \AtBeginDocument{%
942   \set@default@ns%
943   \ifx\module@narr\@empty\setkeys{module}{narr=\module@ns}\fi%
944   \let\module@name\jobname%
945   \let\module@id\module@name % TODO deprecate
946   \edef\module@uri{\module@ns\@QuestionMark\module@name}%
947   \csgdef{module@names@\module@uri}{}%
948   \csgdef{module@imports@\module@uri}{}%
949   \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
950   \expandafter\global\expandafter\let\csname stex@module@\module@name\expandafter\endcsname\csname
951   \edef\this@module{%
952     \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
953   }%
954   \csdef{module@defs@\module@uri}{}%
955   \ifcvoid{mh@currentrepos}{}%
956     \@inmhrepostrue%
957     \addto@thismodule{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\endcsname
958       {\noexpand\mh@currentrepos}}%
959     \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\mh@currentrepos}}%
960   }%
961 }

```

**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.

```

962 \newif\if@inimport\@inimportfalse
963 \def\latexml@import#1{\latexml@annotate{stex:import}{#1}{\ }}%
964 \def\activate@defs#1{%
965   \stex@debug{Activating import #1}%
966   \if@inimport\else%
967     \latexml@import{#1}%
968     \def\inimport@module{#1}%
969     \stex@debug{Entering import #1}%
970     \@inimporttrue%
971   \fi%
972   \edef\activate@defs@uri{#1}%
973   \ifcundef{module@defs@\activate@defs@uri}{%
974     \PackageError{stex}{No module with URI \activate@defs@uri loaded}{Probably missing an
975       \detokenize{\importmodule} (or variant) somewhere?

```

```

976   }
977 }{%
978   \ifcsundef{module@\activate@defs@uri @activated}%
979   {\csname module@defs@\activate@defs@uri\endcsname}{}%
980   \@namedef{module@\activate@defs@uri @activated}{true}%
981 }%
982 \def\inimport@thismodule{#1}%
983 \stex@debug{End of import #1}%
984 \ifx\inimport@thismodule\inimport@module\@inimportfalse%
985   \stex@debug{Leaving import #1}%
986 \fi%
987 }%

```

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

```

988 \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.

```

989 \def\inputref@preskip{}
990 \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).

```

991 \newrobustcmd\inputref[2] [] {%
992   \importmodule@bookkeeping{#1}{#2}{%
993     %\inputreftrue
994     \inputref@preskip%
995     \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
996     \inputref@postskip%
997   }%
998 }%

```

**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.

```

999 \newif\if@symdeflocal\@symdeflocalfalse

```

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

```

1000 \def\define@in@module#1#2{

```

```

1001 \expandafter\edef\csname #1\endcsname{#2}%
1002 \edef\define@in@module@temp{%
1003   \def\expandafter\noexpand\csname#1\endcsname%
1004     {#2}%
1005 }%
1006 \if@symdeflocal\else%
1007   \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1008   \expandafter\endcsname\expandafter{\define@in@module@temp}%
1009 \fi%
1010 }

```

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

```

1011 \addmetakey{symdecl}{name}%
1012 \addmetakey{symdecl}{type}%
1013 \addmetakey{symdecl}{args}%
1014 \addmetakey[false]{symdecl}{local}[true]%
1015
1016 \newcommand\symdecl[2][]{%
1017   \ifcsdef{this@module}{%
1018     \metasetkeys{symdecl}{#1}%
1019     \ifcsvoid{symdecl@name}{
1020       \edef\symdecl@name{#2}%
1021     }{}%
1022     \edef\symdecl@uri{\module@uri\@QuestionMark\symdecl@name}%
1023     \ifcsvoid{stex@symbol@\symdecl@name}{%
1024       \expandafter\edef\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}%
1025     }{}%
1026     \expandafter\def\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}%
1027   }%
1028   \edef\symdecl@symbolmacro{%
1029     \noexpand\ifcsvoid{stex@symbol@\symdecl@name}{%
1030       \expandafter\edef\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}%
1031     }{}%
1032     \expandafter\def\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}%
1033   }%
1034 }%
1035 \ifcsvoid{symdecl@type}{}%
1036   \setbox\modules@import@tempbox\hbox{$\symdecl@type$} % only to have latex check this
1037 }%
1038 \ifcsvoid{symdecl@args}{\csgdef{\symdecl@uri\@QuestionMark args}{}}{%
1039   \IfInteger\symdecl@args{\notation@num@to@ia@\symdecl@args\csxdef{\symdecl@uri\@QuestionMark args}{\symdecl@uri\@QuestionMark args}}%
1040   \ex\globale\ex\let\csname\symdecl@uri\@QuestionMark args\endcsname\symdecl@args%
1041 }%
1042 }%
1043 \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1044 \expandafter\endcsname\expandafter{\symdecl@symbolmacro}%
1045 \ifcsvoid{\symdecl@uri}{%

```



```

1046 \ifcsvoid{module@names@\module@uri}{%
1047 \csxdef{module@names@\module@uri}{\symdecl@name}%
1048 }{%
1049 \csxdef{module@names@\module@uri}{\symdecl@name,%
1050 \csname module@names@\module@uri\endcsname}%
1051 }%
1052 }{%
1053 % not compatible with circular dependencies, e.g. test/omdoc/07-modules/smsstesta.tex
1054 \PackageWarning{stex}{symbol already defined: \symdecl@uri}{%
1055 You need to pick a fresh name for your symbol%
1056 }%
1057 }%
1058 \define@in@module\symdecl@uri{\noexpand\@invoke@symbol{\symdecl@uri}}%
1059 \define@in@module{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
1060 }{%
1061 \PackageError{stex}{\detokenize{\symdecl} not in a module}{You need to be in a module%
1062 in order to declare a new symbol}
1063 }%
1064 \if@inimport\else\latexml@symdecl\symdecl@uri{\symdecl@type$\fi%
1065 \if@insymdef\else\parsemodule@maybesetcodes\fi%
1066 }
1067 \def\latexml@symdecl#1{\latexml@annotate{stex: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:

```

1068 \edef\stex@ambiguous{\detokenize{ambiguous}}
1069 \edef\stex@macrostring{\detokenize{\macro:->\@invoke@symbol}}
1070 \def\modules@getURIfromName#1{%
1071 \def\notation@uri{}%
1072 \edef\modules@getURI@name{#1}%
1073 \ifcsvoid{\modules@getURI@name}{
1074 \edef\modules@temp@meaning{
1075 }{
1076 \edef\modules@temp@meaning{\expandafter\meaning\csname\modules@getURI@name\endcsname}
1077 }
1078 \IfBeginWith\modules@temp@meaning\stex@macrostring{
1079 % is a \@invoke@symbol macro
1080 \StrPosition\modules@temp@meaning\@close@brace[\stex@tempnum]
1081 \StrMid\modules@temp@meaning{26}{\the\numexpr\stex@tempnum-1\@Space}{\notation@uri}
1082 }{
1083 % Check whether full URI or module?symbol or just name
1084 \StrCount\modules@getURI@name\@QuestionMark[\isuri@number]
1085 \ifnum\isuri@number=2

```

```

1086     \edef\notation@uri{\modules@getURI@name}
1087   \else
1088     \ifnum\isuri@number=1
1089       % module?name
1090       \StrCut\modules@getURI@name\@QuestionMark\isuri@mod\isuri@name
1091       \ifcsvoid{stex@module@\isuri@mod}{
1092         \PackageError{stex}{No module with name \isuri@mod\@Space loaded}{}
1093       }{
1094         \expandafter\ifx\csname stex@module@\isuri@mod\endcsname\stex@ambiguous
1095         \PackageError{stex}{Module name \isuri@mod\@Space is ambiguous}{}
1096       }
1097       \else
1098         \edef\notation@uri{\csname stex@module@\isuri@mod\endcsname\@URI\@QuestionMark\isuri@name}
1099       \fi
1100     }
1101   \else
1102     %name
1103     \ifcsvoid{stex@symbol@\modules@getURI@name}{
1104       \PackageError{stex}{No symbol with name \modules@getURI@name\@Space known}{}
1105     }{
1106       \ifcsvoid{\module@uri\@QuestionMark\modules@getURI@name}{
1107         \expandafter\ifx\csname stex@symbol@\modules@getURI@name\endcsname\stex@ambiguous
1108         % Symbol name ambiguous and not in current module
1109         \PackageError{stex}{Symbol name, URI or macroname \detokenize{#1} found!}{}%
1110       }
1111       \else
1112         % Symbol not in current module, but unambiguous
1113         \edef\notation@uri{\csname stex@symbol@\modules@getURI@name\endcsname}
1114       \fi
1115     }{ % Symbol in current module
1116       \edef\notation@uri{\module@uri\@QuestionMark\modules@getURI@name}
1117     }
1118   \fi
1119 }
1120 }

```

**\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 `\<uri>\#<variant>`, where `<variant>` contains `arity,lang` and `variant` in that order.

```

1121 \newif\if@innotation\@innotationfalse

```

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

```

1122 \providerobustcmd\notation[2] []{%
1123   \edef\notation@first{#1}%
1124   \edef\notation@second{#2}%
1125   \notation@%
1126 }
1127
1128 \newcommand\notation@[2] [0]{%

```

```

1129 \edef\notation@donext{\noexpand\notation@@[\notation@first]%
1130   {\notation@second}[#1]}%
1131 \notation@donext{#2}%
1132 }
1133

```

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 *variant*:

```

1134 \def\notation@parse@params#1#2{%
1135   \def\notation@curr@prec{%
1136     \def\notation@curr@args{%
1137       \def\notation@curr@variant{%
1138         \def\notation@curr@arityvar{%
1139           \def\notation@curr@provided@arity{#2}
1140           \def\notation@curr@lang{%
1141             \def\notation@options@temp{#1}
1142             \notation@parse@params@%
1143             \ifx\notation@curr@args\empty%
1144               \ifx\notation@curr@provided@arity\empty%
1145                 \notation@num@to@ia\notation@curr@arityvar%
1146               \else%
1147                 \notation@num@to@ia\notation@curr@provided@arity%
1148               \fi%
1149             \fi%
1150             \StrLen\notation@curr@args[\notation@curr@arity]%
1151           }
1152           \def\notation@parse@params@{%
1153             \IfSubStr\notation@options@temp,{%
1154               \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
1155               \notation@parse@param%
1156             }\def\notation@parse@params@%
1157             \ifx\notation@options@temp\empty\else%
1158               \let\notation@option@temp\notation@options@temp%
1159               \notation@parse@param%
1160             \fi}%
1161         }
1162       }
1163     }
1164     \def\notation@parse@param{%
1165       \path@trimstring\notation@option@temp%
1166       \ifx\notation@option@temp\empty\else%
1167         \IfSubStr\notation@option@temp={%
1168           \StrCut\notation@option@temp=\notation@key\notation@value%
1169           \path@trimstring\notation@key%
1170           \path@trimstring\notation@value%
1171           \IfStrEq\notation@key{prec}{%
1172             \edef\notation@curr@prec{\notation@value}%
1173           }{%
1174             \IfStrEq\notation@key{args}{%
1175               \edef\notation@curr@args{\notation@value}%

```

```

1175     }{%
1176     \IfStrEq\notation@key{lang}{%
1177         \edef\notation@curr@lang{\notation@value}%
1178     }{%
1179     \IfStrEq\notation@key{variant}{%
1180         \edef\notation@curr@variant{\notation@value}%
1181     }{%
1182     \IfStrEq\notation@key{arity}{%
1183         \edef\notation@curr@arityvar{\notation@value}%
1184     }{%
1185     }}}}%
1186     }{%
1187         \edef\notation@curr@variant{\notation@option@temp}%
1188     }%
1189     \fi%
1190 }
1191
1192 % converts an integer to a string of 'i's, e.g. 3 => iii,
1193 % and stores the result in \notation@curr@args
1194 \def\notation@num@to@ia#1{%
1195     \IfInteger{#1}{
1196         \notation@num@to@ia@#1%
1197     }{%
1198         %
1199     }%
1200 }
1201 \def\notation@num@to@ia@#1{%
1202     \ifnum#1>0%
1203         \edef\notation@curr@args{\notation@curr@args i}%
1204         \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
1205     \fi%
1206 }
1207
1208
1209 \newcount\notation@argument@counter
1210
1211 % parses the notation arguments and wraps them in
1212 % \notation@assoc and \notation@argprec for flexary arguments and precedences
1213 \def\notation@@[#1]#2[#3]#4{%
1214     \modules@getURIfromName{#2}%
1215     \notation@parse@params{#1}{#3}%
1216     \let\notation@curr@todo@args\notation@curr@args%
1217     \def\notation@temp@notation{%
1218         \ex\renewcommand\ex\notation@temp@notation\ex[\notation@curr@arity]{#4}%
1219         % precedence
1220         \let\notation@curr@precstring\notation@curr@prec%
1221         \IfSubStr\notation@curr@prec%;{%
1222             \StrCut\notation@curr@prec%;\notation@curr@prec\notation@curr@prec%
1223             \ifx\notation@curr@prec\@empty\def\notation@curr@prec{0}\fi%

```

```

1224 }{%
1225   \ifx\notation@curr@prec\@empty%
1226     \ifnum\notation@curr@arity=0\relax%
1227       \edef\notation@curr@prec{\infprec}%
1228     \else%
1229       \def\notation@curr@prec{0}%
1230     \fi%
1231   \else%
1232     \edef\notation@curr@prec{\notation@curr@prec}%
1233     \def\notation@curr@prec{}%
1234   \fi%
1235 }%
1236 % arguments
1237 \notation@argument@counter=0%
1238 \def\notation@curr@extargs{}%
1239 \notation@do@args%
1240 }
1241
1242 \edef\notation@ichar{\detokenize{i}}%
1243
1244 % parses additional notation components for (associative) arguments
1245 \def\notation@do@args{%
1246   \advance\notation@argument@counter by 1%
1247   \def\notation@nextarg@temp{}%
1248   \ifx\notation@curr@todo@args\@empty%
1249     \ex\notation@after%
1250   \else%
1251     % argument precedence
1252     \IfSubStr\notation@curr@prec{x}{%
1253       \StrCut\notation@curr@prec{x}\notation@curr@argprec\notation@curr@prec%
1254     }{%
1255       \edef\notation@curr@argprec{\notation@curr@prec}%
1256       \def\notation@curr@prec{}%
1257     }%
1258     \ifx\notation@curr@argprec\@empty%
1259       \let\notation@curr@argprec\notation@curr@prec%
1260     \fi%
1261     \StrChar\notation@curr@todo@args1[\notation@argchar]%
1262     \edef\notation@argchar{\ex\detokenize\ex{\notation@argchar}}%
1263     \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1264     \ifx\notation@argchar\notation@ichar%
1265       % normal argument
1266       \edef\notation@nextarg@temp{%
1267         {\stex@arg{\the\notation@argument@counter}\notation@curr@argprec}{#####\the
1268       }%
1269       \ex\g@addto@macro@safe\ex\notation@curr@extargs%
1270       \ex{\notation@nextarg@temp}%
1271       \ex\ex\ex\notation@do@args%
1272     \else%
1273       % associative argument

```

```

1274      \ex\ex\ex\notation@parse@assocarg%
1275      \fi%
1276 \fi%
1277 }
1278
1279 \def\notation@parse@assocarg#1{%
1280   \edef\notation@nextarg@temp{%
1281     {\stex@arg{\the\notation@argument@counter}}{\notation@curr@argprec}{\notation@assoc{#1}}{###}
1282   }%
1283   \ex@g@addto@macro@safe\ex\notation@curr@extargs\ex{\notation@nextarg@temp}%
1284   \notation@do@args%
1285 }
1286
1287 \protected\def\safe@newcommand#1{%
1288   \ifdefined#1\ex\renewcommand\else\ex\newcommand\fi#1%
1289 }
1290
1291 % finally creates the actual macros
1292 \def\notation@after{
1293   % \notation@curr@prec
1294   % \notation@curr@args
1295   % \notation@curr@variant
1296   % \notation@curr@arity
1297   % \notation@curr@provided@arity
1298   % \notation@curr@lang
1299   % \notation@uri
1300   \def\notation@temp@fragment{}%
1301   \ifx\notation@curr@arityvar\@empty\else%
1302     \edef\notation@temp@fragment{arity=\notation@curr@arityvar}%
1303   \fi%
1304   \ifx\notation@curr@lang\@empty\else%
1305     \ifx\notation@temp@fragment\@empty%
1306       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1307     \else%
1308       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1309     \fi%
1310   \fi%
1311   \ifx\notation@curr@variant\@empty\else%
1312     \ifx\notation@temp@fragment\@empty%
1313       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1314     \else%
1315       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1316     \fi%
1317   \fi%
1318   \ex\ex\ex\def\ex\ex\ex\notation@temp@notation\ex\ex\ex%
1319     {\ex\notation@temp@notation\notation@curr@extargs}%
1320   \ifnum\notation@curr@arity=0
1321     \edef\notation@temp@notation{\stex@oms{\notation@uri\@Fragment\notation@temp@fragment}}{\not
1322   \else
1323     \edef\notation@temp@notation{\stex@oma{\notation@uri\@Fragment\notation@temp@fragment}}{\not

```

```

1324 \fi
1325 \stex@debug{Notation \notation@uri: \meaning\notation@temp@notation}%
1326 \notation@final%
1327 \parsemodule@maybesetcodes%
1328 }
1329
1330 \def\notation@final{%
1331 \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1332 \stex@debug{Defining \notation@csname of arity \notation@curr@arity}%
1333 \ifcvoid{\notation@csname}{%
1334 \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1335 \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@curr@arity\ex]%
1336 \ex{\notation@temp@notation}%
1337 \edef\symdecl@temps{%
1338 \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@curr@ari
1339 }%
1340 \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1341 \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1342 }-%
1343 \PackageWarning{stex}{notation already defined: \notation@csname}{%
1344 Choose a different set of notation options (variant,lang,arity)%
1345 }%
1346 }%
1347 \@innotationfalse%
1348 \if@inimport\else\if@latexml%
1349 \let\notation@simarg@args\notation@curr@args%
1350 \notation@argument@counter=0%
1351 \def\notation@simargs{%
1352 \notation@simulate@arguments%
1353 \latexml\notation\notation@uri\notation@temp@fragment\notation@curr@args\notation@curr@prec
1354 {\csname\notation@csname\ex\endcsname\notation@simargs$}%
1355 \fi\fi%
1356 }
1357 \def\notation@simulate@arguments{%
1358 \ifx\notation@simarg@args\@empty\else%
1359 \advance\notation@argument@counter by 1%
1360 \IfBeginWith\notation@simarg@args{i}{%
1361 \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\the\notation@argument
1362 }-%
1363 \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\@Fragment\the\notati
1364 }%
1365 \StrGobbleLeft\notation@simarg@args1[\notation@simarg@args]%
1366 \notation@simulate@arguments%
1367 \fi%
1368 }
1369 % URI, fragment, arity, notation
1370 \def\latexml@notation#1#2#3#4#5{\latexml@annotate{stex:notation}{#1}{%
1371 \latexml@annotate{fragment}{#2}{\ }%
1372 \latexml@annotate{arity}{#3}{\ }%
1373 \latexml@annotate{precedence}{#4}{\ }%

```

```

1374 \latexml@annotate{notation}{#5}%
1375 }}

```

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

```

1376 \protected\def\notation@assoc#1#2{% function, argv
1377 \let\@tmpop=\relax% do not print the function the first time round
1378 \@for\@I:=#2\do{\@tmpop% print the function
1379 % write the i-th argument with locally updated precedence
1380 \@I%
1381 \def\@tmpop{#1}%
1382 }%
1383 }%
1384
1385 \def\notation@lparen{()}
1386 \def\notation@rparen{)}
1387 \def\infprec{1000000}
1388 \def\neginfprec{-\infprec}
1389
1390 \newcount\notation@downprec
1391 \notation@downprec=\neginfprec
1392
1393 % patching displaymode
1394 \newif\if@displaymode\@displaymodefalse
1395 \ex\everydisplay\ex{\the\everydisplay\@displaymodetrue}
1396 \let\old@displaystyle\displaystyle
1397 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1398
1399 \protected\def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1400 \def\notation@innertmp{#1}%
1401 \if@displaymode%
1402 \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1403 \ex\notation@resetbrackets\ex\notation@innertmp%
1404 \ex\right\notation@rparen%
1405 \else%
1406 \ex\ex\ex\notation@lparen%
1407 \ex\notation@resetbrackets\ex\notation@innertmp%
1408 \notation@rparen%
1409 \fi%
1410 }
1411
1412 \protected\def\withbrackets#1#2#3{%
1413 \edef\notation@lparen{#1}%
1414 \edef\notation@rparen{#2}%
1415 #3%
1416 \notation@resetbrackets%
1417 }
1418
1419 \protected\def\notation@resetbrackets{%
1420 \def\notation@lparen{()}%

```



```

1421 \def\notation@rparen{}}%
1422 }
1423
1424 \protected\def\stex@oms#1#2#3{%
1425 \if@innotation%
1426 \notation@symprec{#2}{#3}%
1427 \else%
1428 \@innotationtrue%
1429 \latexml@oms{#1}{\notation@symprec{#2}{#3}}%
1430 \@innotationfalse%
1431 \fi%
1432 }
1433
1434 % for LaTeXML Bindings
1435 \def\latexml@oms#1#2{%
1436 \if@latexml\latexml@annotate{stex:OMS}{#1}{#2}\else#2\fi%
1437 }
1438
1439 \protected\def\stex@oma#1#2#3{%
1440 \if@innotation%
1441 \notation@symprec{#2}{#3}%
1442 \else%
1443 \@innotationtrue%
1444 \latexml@oma{#1}{\notation@symprec{#2}{#3}}%
1445 \@innotationfalse%
1446 \fi%
1447 }
1448
1449 % for LaTeXML Bindings
1450 \def\latexml@oma#1#2{%
1451 \if@latexml\latexml@annotate{stex:OMA}{#1}{#2}\else#2\fi%
1452 }
1453
1454 \def\notation@symprec#1#2{%
1455 \ifnum#1>\notation@downprec\relax%
1456 \notation@resetbrackets#2%
1457 \else%
1458 \ifnum\notation@downprec=\infprec\relax%
1459 \notation@resetbrackets#2%
1460 \else
1461 \if@inarray@
1462 \notation@resetbrackets#2
1463 \else\dobrackets{#2}\fi%
1464 \fi\fi%
1465 }
1466
1467 \newif\if@inarray@\@inarray@false
1468
1469
1470 \protected\def\stex@arg#1#2#3{%

```

```

1471 \innotationfalse%
1472 \latexml@arg{#1}{\notation@argprec{#2}{#3}}%
1473 \innotationtrue%
1474 }
1475
1476 % for LaTeXML Bindings
1477 \def\latexml@arg#1#2{%
1478 \if\latexml\latexml@annotate{stex:arg}{#1}{#2}\else#2\fi%
1479 }
1480
1481 \def\notation@argprec#1#2{%
1482 \def\notation@innertmp{#2}
1483 \edef\notation@downprec@temp{\number#1}%
1484 \notation@downprec=\expandafter\notation@downprec@temp%
1485 \expandafter\relax\expandafter\notation@innertmp%
1486 \expandafter\notation@downprec\expandafter=\number\notation@downprec\relax%
1487 }

```

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

```

1488 \protected\def\@invoke@symbol#1{%
1489 \def\@invoke@symbol@first{#1}%
1490 \symbol@args%
1491 }

```

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

```

1492 \newcommand\symbol@args[1][]{%
1493 \notation@parse@params{#1}{}%
1494 \def\notation@temp@fragment{}%
1495 \ifx\notation@curr@arityvar\@empty\else%
1496 \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1497 \fi%
1498 \ifx\notation@curr@lang\@empty\else%
1499 \ifx\notation@temp@fragment\@empty%
1500 \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1501 \else%
1502 \edef\notation@temp@fragment{\notation@temp@fragment\& lang=\notation@curr@lang}%
1503 \fi%
1504 \fi%
1505 \ifx\notation@curr@variant\@empty\else%
1506 \ifx\notation@temp@fragment\@empty%
1507 \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1508 \else%
1509 \edef\notation@temp@fragment{\notation@temp@fragment\& variant=\notation@curr@va
1510 \fi%
1511 \fi%
1512 %
1513 \ifmmode\def\invoke@symbol@next{\@invoke@symbol@math\@invoke@symbol@first\notation@temp@fragm
1514 \else\def\invoke@symbol@next{\@invoke@symbol@text\@invoke@symbol@first\notation@temp@fragment

```

```

1515 \invoke@symbol@next%
1516 }

```

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

```

1517 \def\@invoke@symbol@math#1#2{%
1518 \csname #1\@Ffragment#2\endcsname%
1519 }

```

TODO:

```

1520 \def\@invoke@symbol@text#1#2{%
1521 }

```

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

```

1522 \def\setstexlang#1{%
1523 \def\stex@lang{#1}%
1524 }%
1525 \setstexlang{en}
1526 \def\setstexvariant#1#2{%
1527 % TODO
1528 }
1529 \def\setstexvariants#1{%
1530 \def\stex@variants{#1}%
1531 }

```

```

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 }
```

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

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

```

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

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

### 3.6 Term References

```
\ifhref
```

```
1532 \newif\ifhref\hreffalse%
1533 \AtBeginDocument{%
1534   \ifpackageloaded{hyperref}{%
1535     \hreftrue%
1536   }{%
1537     \hreffalse%
1538   }%
1539 }
```

`\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.

```
1540 \newbox\stex@targetbox
1541 \def\termref@maketarget#1#2{%
1542   % #1: symbol URI
1543   % #2: text
1544   \stex@debug{Here: #1 <> #2}%
1545   \ifhref\if@smsmode\else%
1546     \hypertarget{sref@#1@target}{#2}%
1547   \fi\fi%
1548   \stex@debug{Here!}%
1549   \expandafter\edef\csname sref@#1\endcsname##1{%
1550     \ifhref\if@smsmode\else\noexpand\hyperlink{sref@#1@target}{##1}\fi\fi%
1551   }%
1552 }
```

```
\@termref
```

```
1553 \def\@termref#1#2{%
1554   % #1: symbol URI
1555   % #2: text
1556   \ifcvoid{#1}{%
```

```

1557 \StrCut[2]{#1}\@QuestionMark\termref@mod\termref@name%
1558 \ifcsvoid{\termref@mod}{%
1559 \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
1560 }{%
1561 \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
1562 contains no symbol with name \termref@name.%
1563 }{%}
1564 }%
1565 }{%
1566 \ifcsvoid{sref@#1}{%
1567 #2% TODO: No reference point exists!
1568 }{%
1569 \csname sref@#1\endcsname{#2}%
1570 }%
1571 }%
1572 }

\tref
1573
1574 \def\@capitalize#1{\uppercase{#1}}%
1575 \newrobustcmd\capitalize[1]{\expandafter\@capitalize #1}%
1576
1577 \newcommand\tref[2][]{%
1578 \edef\tref@name{#1}%
1579 \expandafter\modules@getURIfromName\expandafter{\tref@name}%
1580 \expandafter\@termref\expandafter{\notation@uri}{#2}%
1581 }
1582 \def\trefs#1{%
1583 \modules@getURIfromName{#1}%
1584 % TODO
1585 }
1586 \def\Tref#1{%
1587 \modules@getURIfromName{#1}%
1588 % TODO
1589 }
1590 \def\Trefs#1{%
1591 \modules@getURIfromName{#1}%
1592 % TODO
1593 }

\defi
1594 \addmetakey{defi}{name}
1595 \def\@definiendum#1#2{%
1596 \parsemodule@maybesetcodes%
1597 \stex@debug{Here: #1 | #2}%
1598 \termref@maketarget{#1}{#2}\termref@maketarget{#1}{\defemph{#2}}%
1599 }
1600
1601 \newcommand\defi[2][]{%
1602 \metasetkeys{defi}{#1}%

```

```

1603 \ifx\defi@name\@empty%
1604   \symdecl@constructname{#2}%
1605   \let\defi@name\symdecl@name%
1606   \let\defi@verbalization\symdecl@verbalization%
1607 \else%
1608   \edef\defi@verbalization{#2}%
1609 \fi%
1610 \ifcvoid{\module@uri\@QuestionMark\defi@name}{%
1611   \symdecl\defi@name%
1612 }{\edef\symdecl@uri{\module@uri\@QuestionMark\defi@name}}%
1613 \@definiendum\symdecl@uri\defi@verbalization%
1614 }
1615 \def\Defi#1{%
1616   \symdecl{#1}%
1617   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization}%
1618 }
1619 \def\defis#1{%
1620   \symdecl{#1}%
1621   \@definiendum\symdecl@uri{\symdecl@verbalization s}%
1622 }
1623 \def\Defis#1{%
1624   \symdecl{#1}%
1625   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization s}%
1626 }

```

### 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
1627 \newif\ifhref\hreffalse%
1628 \AtBeginDocument{%
1629   \@ifpackageloaded{hyperref}{%
1630     \hreftrue%
1631   }{%
1632     \hreffalse%
1633   }%
1634 }%
1635 \newcommand\sref@href@ifh[2]{%
1636   \ifhref%
1637     \href{#1}{#2}%
1638   \else%
1639     #2%
1640   \fi%
1641 }%
1642 \newcommand\sref@hlink@ifh[2]{%
1643   \ifhref%

```

```

1644 \hyperlink{#1}{#2}%
1645 \else%
1646 #2%
1647 \fi%
1648 }%
1649 \newcommand\sref@target@ifh[2]{%
1650 \ifhref%
1651 \hypertarget{#1}{#2}%
1652 \else%
1653 #2%
1654 \fi%
1655 }%

```

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.

```

1656 \def\sref@target{%
1657 \ifx\sref@id\empty%
1658 \relax%
1659 \else%
1660 \edef\@target{sref@\ifcsundef{sref@part}}{\sref@part @}\sref@id @target}%
1661 \sref@target@ifh\@target}%
1662 \fi%
1663 }%

```

**\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.

```

1664 \addmetakey{srefaddidkey}{prefix}
1665 \newcommand\srefaddidkey[2][{}]{%
1666 \metasetkeys{srefaddidkey}{#1}%
1667 \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
1668 \metakeys@ext@clear@keys{#2}{id}{}%
1669 \metakeys@ext@showkeys{#2}{id}%
1670 \define@key{#2}{id}{%
1671 \edef\sref@id{\srefaddidkey@prefix ##1}%
1672 \expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
1673 \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1674 }%
1675 }%

```

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

```

1676 \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.

```
1677 \ifextrefs%
1678   \newwrite\refs@file%
1679 \else%
1680   \def\refs@file{\@auxout}%
1681 \fi%
```

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

```
1682 \newcommand\sref@def[3]{%
1683   \protected@write\refs@file{}\string\sref@def{#1}{#2}{#3}}%
1684 }%
```

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

```
1685 \newcommand\sref@label[2]{%
1686   \sref@def{\ifcsundef\sref@part}{\sref@part @}{#2}{page}{\thepage}%
1687   \sref@def{\ifcsundef\sref@part}{\sref@part @}{#2}{label}{#1}%
1688 }%
```

`\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`.

```
1689 \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.

```
1690 \def\sref@id{} % make sure that defined
1691 \newcommand\sref@label@id[1]{%
1692   \ifx\sref@id\@empty%
1693     \relax%
1694   \else%
1695     \sref@label{#1}{\sref@id}%
1696   \fi%
1697 }%
```

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

```
1698 \newcommand\sref@label@id@arg[2]{%
1699   \def\@@id{#2}
1700   \ifx\@@id\@empty%
1701     \relax%
1702   \else%
1703     \sref@label{#1}{\@@id}%
1704   \fi%
1705 }%
```



### 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`.

```
1706 \newenvironment{modsig}[2][\def\@test{#1}%
1707 \ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1708 \expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
1709 %\ignorespacesandpars
1710 }
1711 {\end{module}}\ignorespacesandpars
1712 }
```

### 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`.

```
1713 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1714 \newrobustcmd\@gimport@star[2][\def\@test{#1}%
1715 \edef\mh@repos{\mh@currentrepos}%
1716 \ifx\@test\@empty%
1717 \importmhmodule[conservative,mhrepos=\mh@repos,path=#2]{#2}%
1718 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1719 \mathhub@setcurrentreposinfo{\mh@repos}%
1720 %\ignorespacesandpars
1721 \parsemodule@maybesetcodes}
1722 \newrobustcmd\@gimport@nostar[2][\def\@test{#1}%
1723 \edef\mh@repos{\mh@currentrepos}%
1724 \ifx\@test\@empty%
1725 \importmhmodule[mhrepos=\mh@repos,path=#2]{#2}%
1726 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1727 \mathhub@setcurrentreposinfo{\mh@repos}%
1728 %\ignorespacesandpars
1729 \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.

```

1730 \def\modules@@first#1/#2;{#1}
1731 \newcommand\libinput[1]{%
1732 \stex@debug{Libinput current repo: \meaning\mh@currentrepos}%
1733 \ifcsvoid{mh@currentrepos}{%
1734 \PackageError{stex}{current MathHub repository not found}{}}%
1735 {}
1736 \edef\mh@group{\expandafter\modules@@first\mh@currentrepos;}
1737 \let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
1738 \def\mh@inffile{\MathHub{\mh@group/meta-inf/lib/#1}}
1739 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1740 \IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
1741 \IfFileExists\mh@inffile{\IfFileExists\mh@libfile}{%
1742 {\PackageError{stex}
1743 {Library file missing; cannot input #1.tex\MessageBreak%
1744 Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
1745 do not exist}%
1746 {Check whether the file name is correct}}}}
1747 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1748 \let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}

```

### 3.11 omdoc/omgroup

```

1749 \newcount\section@level
1750
1751 \section@level=2
1752 \ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{}
1753 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{}
1754 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{}
1755 \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>*.

```

1756 \newcommand\omgroup@nonum[2]{%
1757 \ifx\hyper@anchor\@undefined\else\phantomsection\fi%
1758 \addcontentsline{toc}{#1}{#2}\@nameuse{#1}*{#2}}

```

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

```

1759 \newcommand\omgroup@num[2]{%
1760 \edef\@ID{\sref@id}

```

```

1761 \ifx\omgroup@short\@empty% no short title
1762 \@nameuse{#1}{#2}%
1763 \else% we have a short title
1764 \@ifundefined{rdfmeta@sectioning}%
1765   {\@nameuse{#1}[\omgroup@short]{#2}}%
1766   {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
1767 \fi%
1768 \sref@label{id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@ID}

```

**omgroup**

```

1769 \def\@true{true}
1770 \def\@false{false}
1771 \srefaddidkey{omgroup}
1772 \addmetakey{omgroup}{date}
1773 \addmetakey{omgroup}{creators}
1774 \addmetakey{omgroup}{contributors}
1775 \addmetakey{omgroup}{srccite}
1776 \addmetakey{omgroup}{type}
1777 \addmetakey*{omgroup}{short}
1778 \addmetakey*{omgroup}{display}
1779 \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.

```

1780 \newif\if@mainmatter\@mainmattertrue
1781 \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.

```

1782 \addmetakey{omdoc@sect}{name}
1783 \addmetakey[false]{omdoc@sect}{clear}[true]
1784 \addmetakey{omdoc@sect}{ref}
1785 \addmetakey[false]{omdoc@sect}{num}[true]
1786 \newcommand\omdoc@sectioning[3][]{\metasetkeys{omdoc@sect}{#1}%
1787 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
1788 \if@mainmatter% numbering not overridden by frontmatter, etc.
1789 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
1790 \def\current@section@level{\omdoc@sect@name}%
1791 \else\omgroup@nonum{#2}{#3}%
1792 \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.

```

1793 \newcommand\omgroup@redefine@addtocontents[1]{%
1794 %\edef\@import{#1}%
1795 %\@for\@I:=\@import\do{%
1796 %\edef\@path{\csname module@\@I @path\endcsname}%
1797 %\@ifundefined{tf@toc}\relax%
1798 %    {\protected@write\tf@toc}{\string\@requiremodules{\@path}}}}

```

```

1799 %\ifx\hyper@anchor\@undefined% hyperref.sty loaded?
1800 %\def\addcontentsline##1##2##3{%
1801 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##3}}{\thepage}}
1802 %\else% hyperref.sty not loaded
1803 %\def\addcontentsline##1##2##3{%
1804 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##3}}{\thepage}{\c
1805 %\fi
1806 }% hypreref.sty loaded?

```

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

```

1807 \newcount\omgroup@level
1808 \newenvironment{omgroup}[2][ ]% keys, title
1809 {\metasetkeys{omgroup}{##1}\sref@target%
1810 \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.

```

1811 \ifx\omgroup@loadmodules\@true%
1812 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
1813 {\@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`.

```

1814 \advance\section@level by 1\relax%
1815 \ifcase\section@level%
1816 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{##2}%
1817 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{##2}%
1818 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{##2}%
1819 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{##2}%
1820 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{##2}%
1821 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{##2}%
1822 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{##2}%
1823 \fi% \ifcase
1824 \at@begin@omgroup[##1]\section@level{##2}}% for customization
1825 {\advance\section@level by -1\advance\omgroup@level by -1}

```

and finally, we localize the sections

```

1826 \newcommand\omdoc@part@kw{Part}
1827 \newcommand\omdoc@chapter@kw{Chapter}
1828 \newcommand\omdoc@section@kw{Section}
1829 \newcommand\omdoc@subsection@kw{Subsection}
1830 \newcommand\omdoc@subsubsection@kw{Subsubsection}
1831 \newcommand\omdoc@paragraph@kw{paragraph}
1832 \newcommand\omdoc@subparagraph@kw{subparagraph}

```

`\setSGvar` set a global variable

```

1833 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@##1}}

```

```

\useSGvar use a global variable
1834 \newrobustcmd\useSGvar[1]{%
1835   \@ifundefined{sTeX@Gvar@#1}
1836   {\PackageError{omdoc}
1837     {The sTeX Global variable #1 is undefined}
1838     {set it with \protect\setSGvar}}
1839 \@nameuse{sTeX@Gvar@#1}}

```

**blindomgroup**

```

1840 \newcommand\at@begin@blindomgroup[1]{%
1841 \newenvironment{blindomgroup}
1842 {\advance\section@level by 1\at@begin@blindomgroup\section@level}
1843 {\advance\section@level by -1}}

```

## 3.12 omtex

### 3.12.1 Mathematical Text

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

```

1844 \srefaddidkey{omtext}
1845 \addmetakey[] {omtext} {functions}
1846 \addmetakey* {omtext} {display}
1847 \addmetakey {omtext} {for}
1848 \addmetakey {omtext} {from}
1849 \addmetakey {omtext} {type}
1850 \addmetakey* {omtext} {title}
1851 \addmetakey* {omtext} {start}
1852 \addmetakey {omtext} {theory}
1853 \addmetakey {omtext} {continues}
1854 \addmetakey {omtext} {verbalizes}
1855 \addmetakey {omtext} {subject}

```

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

```

1856 \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.

```

1857 \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.

```

1858 \def\omtext@pre@skip{\smallskip}
1859 \def\omtext@post@skip{}

```

```

1860 \newenvironment{omtext}[1][\@in@omtexttrue%
1861   \bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
1862   \def\lec##1{\@lec{##1}}%
1863   \omtext@pre@skip\par\noindent%
1864   \ifx\omtext@title\@empty%
1865     \ifx\omtext@start\@empty\else%
1866       \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
1867     \fi% end omtext@start empty
1868   \else\stDMemph{\omtext@title}:\enspace%
1869   \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
1870 \fi% end omtext@title empty
1871 %\ignorespacesandpars
1872 }
1873 {\egroup\omtext@post@skip\@in@omtextfalse%\ignorespacesandpars
1874 }

```

### 3.12.2 Phrase-level Markup

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

```

1875 \srefaddidkey{phrase}
1876 \addmetakey{phrase}{style}
1877 \addmetakey{phrase}{class}
1878 \addmetakey{phrase}{index}
1879 \addmetakey{phrase}{verbalizes}
1880 \addmetakey{phrase}{type}
1881 \addmetakey{phrase}{only}
1882 \newcommand\phrase[2][\metasetkeys{phrase}{#1}%
1883 \ifx\phrase@only\@empty\only<\phrase@only>{#2}\else #2\fi}

```

`\coref*`

```

1884 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}
1885 \newcommand\corefs[2]{#1\textsubscript{#2}}
1886 \newcommand\coreft[2]{#1\textsuperscript{#2}}

```

`\n*lex`

```

1887 \newcommand\nlex[1]{\green{\sl{#1}}}
1888 \newcommand\nlcex[1]{*\green{\sl{#1}}}

```

`sinlinequote`

```

1889 \def\@sinlinequote#1{‘‘{\sl{#1}}’’}
1890 \def\@sinlinequote#1#2{\@sinlinequote{#2}~#1}
1891 \newcommand\sinlinequote[2][\@sinlinequote{#2}]
1892 {\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.

```

1893 \newcommand\vdec[2] [] {#2}
1894 \newcommand\vest[2] [] {#2}
1895 \newcommand\vcond[2] [] {#2}

```

EdN:1

```

\strucdec 1
1896 \newcommand\strucdec[2] [] {#2}

```

EdN:2

```

\impdec 2
1897 \newcommand\impdec[2] [] {#2}

```

### 3.12.4 Block-Level Markup

`sblockquote`

```

1898 \def\begin@sblockquote{\begin{quote}\sl}
1899 \def\end@sblockquote{\end{quote}}
1900 \def\begin@@sblockquote#1{\begin@sblockquote}
1901 \def\end@@sblockquote#1{\def\@lec#1{\textrm{#1}}\@lec{#1}\end@sblockquote}
1902 \newenvironment{sblockquote}[1] []
1903   {\def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
1904   {\ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}

```

`sboxquote`

```

1905 \newenvironment{sboxquote}[1] []
1906 {\def\@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
1907 {\@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.

```

1908 \providecommand{\@lec}[1]{( #1 )}
1909 \def\@lec#1{\strut\hfil\strut\hfill\@lec{#1}}
1910 \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.

```

1911 \addmetakey{omdoc@index}{at}

```

---

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

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

```

1912 \addmetakey[false]{omdoc@index}{loadmodules}[true]
1913 \newcommand\omdoc@indexi[2][]{\ifindex%
1914 \metasetkeys{omdoc@index}{#1}%
1915 \@bsphack\beginingroup\@sanitize%
1916 \protected@write\@indexfile{}\string\indexentry%
1917 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1918 \ifx\omdoc@index@loadmodules\@true%
1919 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
1920 \else #2\fi% loadmodules
1921 }\thepage}}%
1922 \endgroup\@esphack\fi}%ifindex
1923 \newcommand\omdoc@indexii[3][]{\ifindex%
1924 \metasetkeys{omdoc@index}{#1}%
1925 \@bsphack\beginingroup\@sanitize%
1926 \protected@write\@indexfile{}\string\indexentry%
1927 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1928 \ifx\omdoc@index@loadmodules\@true%
1929 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1930 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
1931 \else #2!#3\fi% loadmodules
1932 }\thepage}}%
1933 \endgroup\@esphack\fi}%ifindex
1934 \newcommand\omdoc@indexiii[4][]{\ifindex%
1935 \metasetkeys{omdoc@index}{#1}%
1936 \@bsphack\beginingroup\@sanitize%
1937 \protected@write\@indexfile{}\string\indexentry%
1938 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1939 \ifx\omdoc@index@loadmodules\@true%
1940 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1941 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1942 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1943 \else #2!#3!#4\fi% loadmodules
1944 }\thepage}}%
1945 \endgroup\@esphack\fi}%ifindex
1946 \newcommand\omdoc@indexiv[5][]{\ifindex%
1947 \metasetkeys{omdoc@index}{#1}%
1948 \@bsphack\beginingroup\@sanitize%
1949 \protected@write\@indexfile{}\string\indexentry%
1950 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1951 \ifx\omdoc@index@loadmodules\@true%
1952 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1953 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1954 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1955 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
1956 \else #2!#3!#4!#5\fi% loadmodules
1957 }\thepage}}%
1958 \endgroup\@esphack\fi}%ifindex

```

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



`\*indi*`

```

1959 \newcommand\aindi[3] []{{#2}\omdoc@indexi[#1]{#3}}
1960 \newcommand\indi[2] []{{#2}\omdoc@indexi[#1]{#2}}
1961 \newcommand\indis[2] []{{#2}\omdoc@indexi[#1]{#2s}}
1962 \newcommand\Indi[2] []{{\captitalize{#2}}\omdoc@indexi[#1]{#2}}
1963 \newcommand\Indis[2] []{{\capitalize{#2}}\omdoc@indexi[#1]{#2s}}
1964
1965 \newcommand\@indii[3] []{\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
1966 \newcommand\aindii[4] []{{#2}\@indii[#1]{#3}{#4}}
1967 \newcommand\indii[3] []{{#2 #3}\@indii[#1]{#2}{#3}}
1968 \newcommand\indiis[3] []{{#2 #3s}\@indii[#1]{#2}{#3}}
1969 \newcommand\Indii[3] []{{\captitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1970 \newcommand\Indiis[3] []{{\capitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1971
1972 \newcommand\@indiii[4] []{\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexii[#1]{#3}{#2 (#4)}}
1973 \newcommand\aindiii[5] []{{#2}\@indiii[#1]{#3}{#4}{#5}}
1974 \newcommand\indiii[4] []{{#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
1975 \newcommand\indiis[4] []{{#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}
1976 \newcommand\Indiii[4] []{{\captitalize{#2 #3 #4}}\@indiii[#1]{#2}{#3}{#4}}
1977 \newcommand\Indiis[4] []{{\capitalize{#2 #3 #4s}}\@indiii[#1]{#2}{#3}{#4}}
1978
1979 \newcommand\@indiv[5] []{\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
1980 \newcommand\aindiv[6] []{{#2}\@indiv[#1]{#3}{#4}{#5}{#6}}
1981 \newcommand\indiv[5] []{{#2 #3 #4 #5}\@indiv[#1]{#2}{#3}{#4}{#5}}
1982 \newcommand\indivs[5] []{{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
1983 \newcommand\Indiv[5] []{{\capitalize{#2 #3 #4 #5s}}\@indiv[#1]{#2}{#3}{#4}{#5}}
1984 \newcommand\Indivs[5] []{{\capitalize{#2 #3 #4 #5s}}\@indiv[#1]{#2}{#3}{#4}{#5}}

```

### 3.12.6 Miscellaneous

Some shortcuts that use math symbols but are not mathematical at all; in particular, they should not be translated by L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>L.

```

1985 \newcommand\hateq{\ensuremath{\widehat{=}}\xspace}
1986 \newcommand\hatequiv{\ensuremath{\widehat{equiv}}\xspace}
1987 \@ifundefined{ergo}%
1988 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1989 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1990 \newcommand{\reflect@squig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%
1991 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
1992 \newcommand\notergo{\ensuremath{\not\leadsto}}
1993 \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*`

```

1994 \newcommand\indextoo[2] []{\indi[#1]{#2}}%

```

```

1995 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}%
1996 \newcommand\indexalt[2][]{\aindi[#1]{#2}}%
1997 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead}%
1998 \newcommand\twintoo[3][]{\indii[#1]{#2}{#3}}%
1999 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}%
2000 \newcommand\twinalt[3][]{\aindii[#1]{#2}{#3}}%
2001 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead}%
2002 \newcommand\atwintoo[4][]{\indiii[#1]{#2}{#3}{#4}}%
2003 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead}%
2004 \newcommand\atwinalt[4][]{\aindii[#1]{#2}{#3}{#4}}%
2005 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instead}%

```

`\my*graphics`

```

2006 \newcommand\mygraphics[2][]{\includegraphics[#1]{#2}}%
2007 \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics}%
2008 \newcommand\mycgraphics[2][]{\begin{center}\mygraphics[#1]{#2}\end{center}}%
2009 \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics}%
2010 \newcommand\mybgraphics[2][]{\fbox{\mygraphics[#1]{#2}}}%
2011 \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics}%
2012 \newcommand\mycbgraphics[2][]{\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}%
2013 \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphics}%

```

## 4 Things to deprecate

Module options:

```

2014 \addmetakey*{module}{id} % TODO: deprecate properly
2015 \addmetakey*{module}{load}
2016 \addmetakey*{module}{path}
2017 \addmetakey*{module}{dir}
2018 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
2019 \addmetakey*{module}{noalign}[true]
2020
2021 \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.

```

2022 %\srefaddidkey{symdef}% what does this do?
2023 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
2024 \define@key{symdef}{noverb}[all]{}%
2025 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
2026 \define@key{symdef}{specializes}{}%
2027 \addmetakey*{symdef}{noalign}[true]

```

```

2028 \define@key{symdef}{primary}[true]{}%
2029 \define@key{symdef}{assocarg}{}%
2030 \define@key{symdef}{bvars}{}%
2031 \define@key{symdef}{bargs}{}%
2032 \addmetakey{symdef}{lang}%
2033 \addmetakey{symdef}{prec}%
2034 \addmetakey{symdef}{arity}%
2035 \addmetakey{symdef}{variant}%
2036 \addmetakey{symdef}{ns}%
2037 \addmetakey{symdef}{args}%
2038 \addmetakey{symdef}{name}%
2039 \addmetakey*{symdef}{title}%
2040 \addmetakey*{symdef}{description}%
2041 \addmetakey{symdef}{subject}%
2042 \addmetakey*{symdef}{display}%
2043 \addmetakey*{symdef}{gfc}%

```

EdN:3

3

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

```

2044 \def\symdef{\ifnextchar[{\@symdef}{\@symdef []}}%
2045 \def\@symdef[#1]#2{\ifnextchar[{\@@symdef[#1]{#2}}{\@@symdef[#1]{#2}[0]}}%

```

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

```

2046 \def\@@symdef[#1]#2[#3]{%
2047   \@insymdef@true%
2048   \metasetkeys{symdef}{#1}%
2049   \edef\symdef@tmp@optpars{\ifcvoid{symdef@name}{[]}{[name=\symdef@name]}}%
2050   \expandafter\symdecl\symdef@tmp@optpars{#2}%
2051   \@insymdef@false%
2052   \notation[#1]{#2}[#3]%
2053 }% mod@show
2054 \def\symdef@type{Symbol}%
2055 \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`.

```

2056 \def\symvariant#1{%
2057   \ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}}%
2058 }%
2059 \def\@symvariant#1[#2]#3#4{%
2060   \notation[#3]{#1}[#2]{#4}%
2061 %\ignorespacesandpars
2062 }%

```

---

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

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

```

2063 \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.

```

2064 \newif\if@importing\@importingfalse
2065 \define@key{symi}{noverb}[all]{}%
2066 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
2067 \define@key{symi}{specializes}{}%
2068 \define@key{symi}{gfc}{}%
2069 \define@key{symi}{noalign}[true]{}%
2070 \newcommand\symi{\@ifstar\@symi@star\@symi}
2071 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
2072   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces
2073 }
2074 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
2075   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignorespaces
2076 }
2077 \newcommand\symii{\@ifstar\@symii@star\@symii}
2078 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
2079   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespaces
2080 }
2081 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
2082   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\ignorespaces
2083 }
2084 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
2085 \newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%
2086   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignorespaces
2087 }
2088 \newcommand\@symiii@star[4][]{\metasetkeys{symi}{#1}%
2089   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi\ignorespaces
2090 }
2091 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
2092 \newcommand\@symiv[5][]{\metasetkeys{symi}{#1}%
2093   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces
2094 }
2095 \newcommand\@symiv@star[5][]{\metasetkeys{symi}{#1}%
2096   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces
2097 }

```

`\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`.

```

2098 %\srefaddidkey{importmhmodule}%
2099 \addmetakey{importmhmodule}{mhrepos}%

```

```

2100 \addmetakey{importmhmodule}{path}%
2101 \addmetakey{importmhmodule}{ext}% why does this exist?
2102 \addmetakey{importmhmodule}{dir}%
2103 \addmetakey[false]{importmhmodule}{conservative}[true]%
2104 \newcommand\importmhmodule[2] []{%
2105   \parsemodule@maybesetcodes
2106   \metasetkeys{importmhmodule}{#1}%
2107   \ifx\importmhmodule@dir\empty%
2108     \edef\@path{\importmhmodule@path}%
2109   \else\edef\@path{\importmhmodule@dir/#2}\fi%
2110   \ifx\@path\empty% if module name is not set
2111     \@importmodule[] {#2}{export}%
2112   \else%
2113     \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
2114     \ifx\importmhmodule@mhrepos\empty% if in the same repos
2115       \relax% no need to change mh@currentrepos, i.e, current directory.
2116     \else%
2117       \mathhub@setcurrentreposinfo\importmhmodule@mhrepos% change it.
2118       \addto@thismodule{x}\noexpand\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}}%
2119     \fi%
2120     \@importmodule[\MathHub{\mh@currentrepos/source/\@path}]{#2}{export}%
2121     \mathhub@setcurrentreposinfo\mh@@repos% after importing, reset to old value
2122     \addto@thismodule{x}\noexpand\mathhub@setcurrentreposinfo{\mh@@repos}}%
2123   \fi%
2124   %\ignorespacesandpars%
2125 }

```

\usemhmodule

```

2126 \addmetakey{importmhmodule}{load}
2127 \addmetakey{importmhmodule}{id}
2128 \addmetakey{importmhmodule}{dir}
2129 \addmetakey{importmhmodule}{mhrepos}
2130
2131 \addmetakey{importmodule}{load}
2132 \addmetakey{importmodule}{id}
2133
2134 \newcommand\usemhmodule[2] []{%
2135   \metasetkeys{importmhmodule}{#1}%
2136   \ifx\importmhmodule@dir\empty%
2137     \edef\@path{\importmhmodule@path}%
2138   \else\edef\@path{\importmhmodule@dir/#2}\fi%
2139   \ifx\@path\empty%
2140     \usemodule[id=\importmhmodule@id]{#2}%
2141   \else%
2142     \edef\mh@@repos{\mh@currentrepos}%
2143     \ifx\importmhmodule@mhrepos\empty%
2144     \else\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
2145     \usemodule{\@path\@QuestionMark#2}%
2146     %\usemodule[load=\MathHub{\mh@currentrepos/source/\@path},
2147     %                                     id=\importmhmodule@id]{#2}%

```

```

2148 \mathhub@setcurrentreposinfo\mh@crepos%
2149 \fi%
2150 %\ignorespacesandpars
2151 }

\mhinputref
2152 \newcommand\mhinputref[2] [] {%
2153   \edef\mhinputref@first{#1}%
2154   \ifx\mhinputref@first\@empty%
2155     \inputref{#2}%
2156   \else%
2157     \inputref[mhrepos=\mhinputref@first]{#2}%
2158   \fi%
2159 }

\trefi*
2160 \newcommand\trefi[2] [] {%
2161   \edef\trefi@mod{#1}%
2162   \ifx\trefi@mod\empty\tref{#2}\else\tref{#1\@QuestionMark#2}\fi%
2163 }
2164 \newcommand\trefii[3] [] {%
2165   \edef\trefi@mod{#1}%
2166   \ifx\trefi@mod\empty\tref{#2-#3}\else\tref{#1\@QuestionMark#2-#3}\fi%
2167 }

\defi*
2168 \def\defii#1#2{\defi{#1!#2}}
2169 \def\Defii#1#2{\Defi{#1!#2}}
2170 \def\defiis#1#2{\defis{#1!#2}}
2171 \def\Defiis#1#2{\Defis{#1!#2}}
2172 \def\defiii#1#2#3{\defi{#1!#2!#3}}
2173 \def\Defiii#1#2#3{\Defi{#1!#2!#3}}
2174 \def\defiiis#1#2#3{\defis{#1!#2!#3}}
2175 \def\Defiiis#1#2#3{\Defis{#1!#2!#3}}
2176 \def\defiv#1#2#3#4{\defi{#1!#2!#3!#4}}
2177 \def\Defiv#1#2#3#4{\Defi{#1!#2!#3!#4}}
2178 \def\defivs#1#2#3#4{\defis{#1!#2!#3!#4}}
2179 \def\Defivs#1#2#3#4{\Defis{#1!#2!#3!#4}}
2180 \def\adefi#1#2{\defi[name=#2]{#1}}
2181 \def\adefii#1#2#3{\defi[name=#2-#3]{#1}}
2182 \def\adefiii#1#2#3#4{\defi[name=#2-#3-#4]{#1}}
2183 \def\adefiv#1#2#3#4#5{\defi[name=#2-#3-#4-#5]{#1}}

```