

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

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

March 10, 2021

## **Abstract**

TODO

---

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

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>User commands</b>	<b>3</b>
<b>3</b>	<b>Implementation</b>	<b>3</b>
3.1	sTeX base . . . . .	4
3.2	Paths and URIs . . . . .	4
3.3	Modules . . . . .	15
3.4	Inheritance . . . . .	21
3.5	Symbols/Notations/Verbalizations . . . . .	31
3.6	Term References . . . . .	44
3.7	sref . . . . .	46
3.8	smultiling . . . . .	48
3.9	smglom . . . . .	49
3.10	mathhub . . . . .	49
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 \edef\old@newlinechar{\the\newlinechar}
7 \newlinechar=-1
8 \let\ex\expandafter
9 % TODO
10 \newif\if@stex@debugmode\@stex@debugmodefalse
11 \DeclareOption{debug}{\@stex@debugmodetrue}
12 \def\stex@debug#1{\if@stex@debugmode\message{^^J#1^^J}\fi}
13 % Modules:
14 \newif\ifmod@show\mod@showfalse
15 \DeclareOption{showmods}{\mod@showtrue}
16 % sref:
17 \newif\ifextrefs\extrefsfalse
18 \DeclareOption{extrefs}{\extrefstrue}
19 %
20 \ProcessOptions
```

A conditional for LaTeXML:

```

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

```

### 3.1 sTeX base

The S<sub>T</sub>E<sub>X</sub> logo:

```

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

```

### 3.2 Paths and URIs

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

```

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

```

`\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/smglob/sets.

```

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

```

### 3.2.1 Path Canonicalization

We define some macros for later comparison.

```

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

```

\@cpath Canonicalizes (file) paths:

```

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

```

```

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

```

```

144         \else%
145         \edef\@cpath@path%
146             {\@cpath@path\@Slash\pathsuris@cpath@temp@a}%
147         \fi%
148         \@cpath@loop%
149     }%
150 \fi\fi%
151 }{%
152     \ifx\@cpath@path\@empty%
153         \edef\@cpath@path{\pathsuris@cpath@temp}%
154     \else%
155         \edef\@cpath@path{\@cpath@path\@Slash\pathsuris@cpath@temp}%
156     \fi%
157 }%
158 }

```

**Test:**

path	canonicalized path	expected
aaa	aaa	aaa
../.. /aaa	../.. /aaa	../.. /aaa
aaa/bbb	aaa/bbb	aaa/bbb
aaa/..		
../.. /aaa/bbb	../.. /aaa/bbb	../.. /aaa/bbb
../aaa/.. /bbb	../bbb	../bbb
../aaa/bbb	../aaa/bbb	../aaa/bbb
aaa/bbb/.. /ddd	aaa/ddd	aaa/ddd
aaa/bbb/.. /ddd	aaa/bbb/ddd	aaa/bbb/ddd
./		
aaa/bbb/.. /..		

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

```

159 \newcommand\cpath@print[1]{%
160     \@cpath{#1}%
161     \@cpath@path%
162 }

```

\path@filename

```

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

```

```
173 }
```

**Test:**

Path: /foo/bar/baz.tex

Filename: baz.tex

`\path@filename@noext`

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

**Test:**

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:

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

**Test:**

We are on windows: no.

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

```
188 \newif\if@windowstopath@inpath@
189 \def\windows@to@path#1{%
190   \@windowstopath@inpath@false%
191   \def\windows@temp{}%
192   \edef\windows@path{#1}%
193   \ifx\windows@path\empty\else%
194     \ex\windows@path@loop\windows@path\windows@path@end%
195   \fi%
196   \let#1\windows@temp%
197 }
198 \def\windows@path@loop#1#2\windows@path@end{%
199   \def\windows@temp@b{#2}%
200   \ifx\windows@temp@b\empty%
201     \def\windows@continue{}%
202   \else%
```



```

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

```

#### Test:

Input: C:\foo \bar .baz

Output: /C/foo/bar.baz

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

```

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

```

```

247         \@windowstopath@inpath@true%
248     \else%
249         \edef\windows@temp{\windows@temp#1}%
250     \fi%
251 \fi%
252 \windows@continue%
253 }

```

**Test:**

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:

```

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

```

**Test:**

>foo bar<

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

```

268 %\if@latexml\else
269 \def\@kpsewhich#1#2{\begingroup%
270     \edef\kpsewhich@cmd{"|kpsewhich #2"%
271     \everyeof{\noexpand}%
272     \catcode'\=12%
273     \edef#1{\@input\kpsewhich@cmd\@Space}%
274     \path@trimstring#1%
275     \if@iswindows@\windows@to@path#1\fi%
276     \xdef#1{\ex\detokenize\expandafter{#1}}%
277 \endgroup}
278 %\fi

```

**Test:**

/usr/share/texlive/texmf-dist/tex/latex/etoolbox/etoolbox.sty

### 3.2.4 sTeX input hooks

We determine the PWD of the current main document:

```

279 \edef\pwd@cmd{\if@iswindows@ -expand-var \@Percent%
280   CD\@Percent\else -var-value PWD\fi}
281 \@kpsewhich\stex@PWD\pwd@cmd
282 \edef\stex@mainfile{\stex@PWD\@Slash\jobname}
283 \edef\stex@mainfile{\ex\detokenize\ex{\stex@mainfile}}

```

**Test:**

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

We keep a stack of \inputed files:

```

284 \def\stex@currfile@stack{}
285
286 \def\stex@currfile@push#1{%
287   \edef\stex@temppath{#1}%
288   \edef\stex@temppath{\ex\detokenize\ex{\stex@temppath}}%
289   \edef\stex@currfile@stack{\stex@currfile%
290     \ifx\stex@currfile@stack\@empty\else,\stex@currfile@stack\fi}
291   \IfBeginWith\stex@temppath\@Slash{\@cpath{\stex@temppath}}{%
292     \@cpath{\stex@PWD\@Slash#1}%
293   }
294   \let\stex@currfile\@cpath@path%
295   \path@filename\stex@currfile\stex@currfilename%
296   \StrLen\stex@currfilename[\stex@currfile@tmp]%
297   \StrGobbleRight\stex@currfile{\the\numexpr%
298     \stex@currfile@tmp+1 }[\stex@currpath]%
299   \global\let\stex@currfile\stex@currfile%
300   \global\let\stex@currpath\stex@currpath%
301   \global\let\stex@currfilename\stex@currfilename%
302 }
303 \def\stex@currfile@pop{%
304   \ifx\stex@currfile@stack\@empty%
305     \global\let\stex@currfile\stex@mainfile%
306     \global\let\stex@currpath\stex@PWD%
307     \global\let\stex@currfilename\jobname%
308   \else%
309     \StrCut\stex@currfile@stack,\stex@currfile\stex@currfile@stack%
310     \path@filename\stex@currfile\stex@currfilename%
311     \StrLen\stex@currfilename[\stex@currfile@tmp]%
312     \StrGobbleRight\stex@currfile{\the\numexpr%
313       \stex@currfile@tmp+1 }[\stex@currpath]%
314     \global\let\stex@currfile\stex@currfile%
315     \global\let\stex@currpath\stex@currpath%
316     \global\let\stex@currfilename\stex@currfilename%
317   \fi%
318 }

```

**\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:

```

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

```

**Test:**

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:

```

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

```

**Test:**

</home/jazzpirate/work/MathHub>

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

```

343 \def\mathhub@findmanifest#1{%
344   \@cpath{#1}%
345   \ifx\@cpath@path\@Slash%
346     \def\manifest@mf{%
347 \else\ifx\@cpath@path\empty%
348     \def\manifest@mf{%
349 \else%
350   \edef\@findmanifest@path{\@cpath@path/MANIFEST.MF}%
351   \if@iswindows@\path@to@windows\@findmanifest@path\fi%
352   \IfFileExists\@findmanifest@path{%
353     \edef\manifest@mf{\@findmanifest@path}%

```

```

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

```

#### Test:

[In /home/jazzpirate/work/MathHub/smglob/mv/source:](#)  
[/home/jazzpirate/work/MathHub/smglob/mv/META-INF/MANIFEST.MF](#)

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

```

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

```

the next helper function iterates over lines in MANIFEST.MF

```

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

```

```

396 \IfStrEq\manifest@key{source-base}{%
397 \xdef\manifest@mf@ns{\manifest@line}%
398 }{%
399 \IfStrEq\manifest@key{ns}{%
400 \xdef\manifest@mf@ns{\manifest@line}%
401 }{%
402 % dependencies
403 \IfStrEq\manifest@key{dependencies}{%
404 \xdef\manifest@mf@deps{\manifest@line}%
405 }{%
406 }}}}
407 \parse@manifest@loop%
408 \fi%
409 }

```

`\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.

```

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

```

#### Test:

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.

```

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

```

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

```

469 \newif\if@inmhrepos\@inmhreposfalse
470 \ifcsvoid{stex@PWD}{}{
471   \mathhub@parsemanifest{currentrepos@}\stex@PWD
472   \@setcurrentreposinfo
473   \ifcsvoid{currentrepos@dir}{\message{sTeX: Not currently in a MathHub repository}}{%
474     \message{Current sTeX repository: \mh@currentrepos}
475   }
476 }

```

### 3.3 Modules

```

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

    Aux:
478 %\def\ignorespacesandpars{\begingroup\catcode13=10%
479 % \@ifnextchar\relax{\endgroup}{\endgroup}}

    and more adapted from http://tex.stackexchange.com/questions/179016/ignore-spaces-and-pars-after-an-environment
480 %\def\ignorespacesandparsafterend#1\ignorespaces\fi{#1%
481 % \fi\ignorespacesandpars}
482 %\def\ignorespacesandpars{\ifhmode\unskip\fi\@ifnextchar\par%
483 % {\ex\ignorespacesandpars\@gobble}{}}

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

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

Test:
Module 3.1[Test]: Foo

module Finally, we define the begin module command for the module environment. Much
of the work has already been done in the keyval bindings, so this is quite simple.
500 \newenvironment{module}[1][]{%
501 \begin{@module}[#1]%
502 \module@heading% make the headings
503 %\ignorespacesandpars
504 \parsemodule@maybesetcodes}%
505 \end{@module}%
506 \ignorespacesafterend%
507 }%
508 \ifmod@show\surroundwithmdframed{module@om@common}\fi%

    Some auxiliary methods:
509 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}
510 \def\addto@thismodule#1{%
511 \@ifundefined{this@module}{-}{%

```



```

512 \expandafter\g@addto@macro@safe\this@module{#1}%
513 }%
514 }
515 \def\addto@thismodule#1{%
516 \ifundefined{this@module}{}%
517 \edef\addto@thismodule@exp{#1}%
518 \expandafter\expandafter\expandafter\g@addto@macro@safe%
519 \expandafter\this@module\expandafter{\addto@thismodule@exp}%
520 }}

```

**@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`.

```

521 \newif\ifarchive@ns@empty@archive@ns@empty@false
522 \def\set@default@ns{%
523 \edef\@module@ns@temp{\stex@currpath}%
524 \if@iswindows@windows@to@path\@module@ns@temp\fi%
525 \archive@ns@empty@false%
526 \stex@debug{Generate new namespace^^J Filepath: \@module@ns@temp}%
527 \ifcvoid{mh@currentrepos}{\archive@ns@empty@true}%
528 {\ex\ifx\cname mathhub@ns@mh@currentrepos\endcsname\@empty\archive@ns@empty@true\fi%
529 }%
530 \stex@debug{ \ifarchive@ns@empty@ Namespace empty\else Namespace not empty\fi}%
531 \ifarchive@ns@empty%
532 \edef\@module@ns@tempuri{file\@Colon\@Slash\@Slash\@module@ns@temp}%
533 \else%
534 \edef\@module@filepath@temppath{\@module@ns@temp}%
535 \edef\@module@ns@tempuri{\cname mathhub@ns@mh@currentrepos\endcsname}%
536 \edef\@module@archivedirpath{\cname mathhub@dir@mh@currentrepos\endcsname\@Slash source}%
537 \edef\@module@archivedirpath{\ex\detokenize\ex{\@module@archivedirpath}}%
538 \IfBeginWith\@module@filepath@temppath\@module@archivedirpath{%
539 \StrLen\@module@archivedirpath[\ns@temp@length]%
540 \StrGobbleLeft\@module@filepath@temppath\ns@temp@length[\@module@filepath@temprest]%
541 \edef\@module@ns@tempuri{\@module@ns@tempuri\@module@filepath@temprest}%
542 }{}%
543 \fi%
544 \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]}
545 \setkeys{module}{ns=\@module@ns@tempuri}%
546 }

```

**Test:**

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

If the module is not given a name, `\set@next@moduleid` computes one by enumeration via the filename, e.g. `stex`, `stex1`, etc.

```

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

### Test:

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

```

559 \newenvironment{@module}[1][[]]{%
560   \metasetkeys{module}{#1}%
561   \ifcvoid{module@name}{\let\module@name\module@id}{}} % TODO deprecate
562   \ifcvoid{module@name}{\set@next@moduleid}{}%
563   \let\module@id\module@name % TODO deprecate
```

```

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

```

```

614 \fi%
615 }%
616 % For LaTeXML bindings
617 \newenvironment{latexml@module}[1]{}{}

Test:
Module 3.2[Foo]:
Name: Foo
URI: file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master?Foo
this@module: macro:->

Test:
Faking a MathHub archive Foo/Bar with URI http://foo.bar/baz:
Module 3.3[Foo2]:
Name: Foo2
URI: http://foo.bar/baz?Foo2
this@module: macro:->\edef\mh@old@repos@http://foo.bar/baz?Foo2{\mh@currentrepos}
\mathhub@setcurrentreposinfo{Foo/Bar}

Test:
Removing the /home/jazzpirate/work/MathHub/ system variable first:
Module 3.4[Foo]:
Name: Foo
URI: file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master?Foo
this@module: macro:->

Test:
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.

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

```

## 3.4 Inheritance

### 3.4.1 Selective Inclusion

The next great goal is to establish the `\requiremodules` macro, which reads an  $\TeX$  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 environments as part of a module signature.

```
627 \newif\if@smsmode\@smsmodefalse
628 \def\parsemodule@allow#1{%
629   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#1\endcsname}%
630 }
631 \def\parsemodule@allowenv#1{%
632   \ex\def\csname parsemodule@allowedenv@#1\endcsname{#1}%
633 }
634 \def\parsemodule@replacemacro#1#2{%
635   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#2\endcsname}%
636 }
637 \def\parsemodule@replaceenv#1#2{%
638   \ex\def\csname parsemodule@allowedenv@#1\endcsname{#2}%
639 }
640 \def\parsemodule@escapechar@beginstring{begin}
641 \def\parsemodule@escapechar@endstring{end}
```

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

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

```

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

```

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

```

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

`\set@parsemodule@catcodes`

```

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

```

`\reset@parsemodule@catcodes`

```

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

```

`\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.

```

733 \def\parsemodule@maybesetcodes{%
734   \if@smsmode\set@parsemodule@catcodes\fi%
735 }

```

`\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@currucs`

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:

```

736
737 \def\parsemodule@escapechar{%
738   \def\parsemodule@escape@currcls{}%
739   \parsemodule@escape@parse@nextchar@%
740 }%

```

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`.

```

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

```

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.

```

755 \def\parsemodule@escapechar@checkcls{%
756   \ifx\parsemodule@escape@currcls\parsemodule@escapechar@beginstring%
757     \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@last@char}%
758   \else%
759     \ifx\parsemodule@escape@currcls\parsemodule@escapechar@endstring%
760       \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@last@char}%
761     \else%

```



```

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

```

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.

```

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

```

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.

```

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

```

```

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

```

`\@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.

```

810 \newrobustcmd\@requiremodules[1]{%
811   \if@tempswa\requiremodules{#1}\fi%
812 }%

```

`\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.

```

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

```

`\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.

```

819 \newbox\modules@import@tempbox
820 \def\requiremodules@smsmode#1{%
821   \setbox\modules@import@tempbox\vbox{%
822     \@smsmodetrue%
823     \set@parsemodule@catcodes%
824     \hbadness=100000\relax%
825     \hfuzz=10000pt\relax%
826     \vbadness=100000\relax%
827     \vfuzz=10000pt\relax%
828     \stexinput{#1.tex}%
829     \reset@parsemodule@catcodes%
830   }%
831   \parsemodule@maybesetcodes%
832 }

```

**Test:**

parsing `F00/testmodule.tex`

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

### 3.4.2 importmodule

\importmodule@bookkeeping

```

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

\importmodule

```

871 %\srefaddidkey{importmodule}
872 \addmetakey{importmodule}{mhrepos}
873 \newcommand\importmodule[2][\@importmodule{#1}{#2}{export}]
```

```

874 \newcommand\@importmodule[3][]{%
875   \importmodule@bookkeeping{#1}{#2}{%
876     \@importmodule[\importmodule@dir]\importmodule@module@name{#3}%
877   }%
878 }

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

879 \newcommand\@importmodule[3][]{%
880   {%
881     \edef\@load{#1}%
882     \edef\@importmodule@name{#2}%
883     \stex@debug{Loading #1}%
884     \if@smsmode\else\ifcsvoid{stex@module@\@importmodule@name}{% TODO check this
885       \stex@iffileexists\@load{
886         \stex@debug{Exists: #1}%
887         \requiremodules\@load}{%
888         \stex@debug{Does not exist: #1}~JTrying \@load\@Slash\@importmodule@name}%
889         \requiremodules{\@load\@Slash\@importmodule@name}%
890       }%
891     }{} \fi%
892     \ifx\@load\@empty\else%
893       {% TODO
894       %   \edef\@path{\csname module@#2@path\endcsname}%
895       %   \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do no
896       %   {\PackageError{stex}% else signal an error
897       %     {Module Name Clash\MessageBreak%
898       %       A module with name #2 was already loaded under the path "\@path"\MessageBreak%
899       %       The imported path "\@load" is probably a different module with the\MessageBreak%
900       %       same name; this is dangerous -- not importing}%
901       %     {Check whether the Module name is correct}%
902       %   }%
903     }%
904     \fi%
905     \global\let\@importmodule@load\@load%
906   }%
907   \edef\@export{#3}\def\@@export{export}%prepare comparison
908   %\ifx\@export\@@export\export@defs{#2}\fi% export the module
909   \ifx\@export\@@export\addto@thismodulelex{%
910     \noexpand\@importmodule[\@importmodule@load]{#2}{noexport}%
911   }%
912   \if@smsmode\else
913   \ifcsvoid{this@module}{}{%

```

```

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

```

#### Test:

```

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

```

#### Test:

```

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

```

#### Test:

```

macro:->\edef\mh@old@repos@http://mathhub.info/FoMID/Core/foundations/types?type.en
{\mh@currentrepos}\mathhub@setcurrentreposinfo{FoMID/Core}\ifcsvoid{stex@symbol@type}{\edef
\stex@symbol@type{http://mathhub.info/FoMID/Core/foundations/types?type.en?type}}{\def
\stex@symbol@type{ambiguous}}\def\http://mathhub.info/FoMID/Core/foundations/types?type.en?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}}\ifcsvoid
{stex@symbol@hastype}{\edef\stex@symbol@hastype{http://mathhub.info/FoMID/Core/foundations/
\stex@symbol@hastype{ambiguous}}\def\http://mathhub.info/FoMID/Core/foundations/types?type.en
{\@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?hastype}}\n
{\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:

```

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

```

### Test:

<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.

```

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

```

```

974     }
975   }{%
976     \ifcsundef{module@\activate@defs@uri @activated}%
977     {\csname module@defs@\activate@defs@uri\endcsname}{}%
978     \namedef{module@\activate@defs@uri @activated}{true}%
979   }%
980   \def\inimport@thismodule{#1}%
981   \stex@debug{End of import #1}%
982   \ifx\inimport@thismodule\inimport@module\inimportfalse%
983     \stex@debug{Leaving import #1}%
984   \fi%
985 }%

```

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

```

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

```

**Test:**

**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/sty/stex-master?Bar?bar}

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

```

987 \def\inputref@preskip{}
988 \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).

```

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

```

**Test:**

**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.

```

997 \newif\if@symdeflocal\@symdeflocalfalse

```

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

```

998 \def\define@in@module#1#2{
999   \expandafter\edef\csname #1\endcsname{#2}%
1000   \edef\define@in@module@temp{%
1001     \def\expandafter\noexpand\csname#1\endcsname%
1002     {#2}%
1003   }%
1004   \if@symdeflocal\else%
1005     \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1006     \expandafter\endcsname\expandafter{\define@in@module@temp}%
1007   \fi%
1008 }
```

`\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.

```

1009 \addmetakey{symdecl}{name}%
1010 \addmetakey{symdecl}{type}%
1011 \addmetakey[false]{symdecl}{local}[true]%
1012
1013 \newcommand\symdecl[2][{}]{%
1014   \ifcsdef{this@module}{%
1015     \metasetkeys{symdecl}{#1}%
1016     \ifcsvoid{symdecl@name}{
1017       \edef\symdecl@name{#2}%
1018     }{}%
1019     \edef\symdecl@uri{\module@uri\@QuestionMark\symdecl@name}%
1020     \ifcsvoid{stex@symbol@\symdecl@name}{%
1021       \expandafter\edef\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}%
1022     }{}%
1023     \expandafter\def\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}}%
1024   }%
1025   \edef\symdecl@symbolmacro{%
1026     \noexpand\ifcsvoid{stex@symbol@\symdecl@name}{%
1027       \expandafter\edef\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\symd
1028     }{}%
1029     \expandafter\def\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\detok
1030   }%
1031 }%
1032 \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1033 \expandafter\endcsname\expandafter{\symdecl@symbolmacro}%
1034 \ifcsvoid{\symdecl@uri}{%
1035   \ifcsvoid{module@names@\module@uri}{%
1036     \csxdef{module@names@\module@uri}{\symdecl@name}%
1037   }{}%
1038   \csxdef{module@names@\module@uri}{\symdecl@name,%
1039     \csname module@names@\module@uri\endcsname}%
1040 }%
1041 }
```



```

1042 % not compatible with circular dependencies, e.g. test/omdoc/07-modules/smstesta.tex
1043 \PackageWarning{stex}{symbol already defined: \symdecl@uri}{%
1044     You need to pick a fresh name for your symbol%
1045 }%
1046 }%
1047 \define@in@module\symdecl@uri{\noexpand\@invoke@symbol{\symdecl@uri}}%
1048 \define@in@module{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
1049 }{%
1050 \PackageError{stex}{\detokenize{\symdecl} not in a module}{You need to be in a module%
1051 in order to declare a new symbol}
1052 }%
1053 \if@inimport\else\latexml@symdecl\symdecl@uri\fi%
1054 \if@insymdef\else\parsemodule@maybesetcodes\fi%
1055 }
1056 \def\latexml@symdecl#1{

```

#### Test:

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

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

### 3.5.1 Notations

`\modules@getURIfromName` This macro searches for the full URI given a symbol name and stores it in `\notation@uri`. Used by e.g. `\notation[...]{foo}{...}` to figure out what symbol foo refers to:

```

1057 \edef\stex@ambiguous{\detokenize{ambiguous}}
1058 \edef\stex@macrostring{\detokenize{macro:->\@invoke@symbol}}
1059 \def\modules@getURIfromName#1{%
1060     \def\notation@uri{}%
1061     \edef\modules@getURI@name{#1}%
1062     \ifcsvoid{\modules@getURI@name}{
1063         \edef\modules@temp@meaning{
1064             }{
1065                 \edef\modules@temp@meaning{\expandafter\meaning\csname\modules@getURI@name\endcsname}
1066             }
1067             \IfBeginWith\modules@temp@meaning\stex@macrostring{
1068                 % is a \@invoke@symbol macro
1069                 \StrPosition\modules@temp@meaning\@close@brace[\stex@tempnum]
1070                 \StrMid\modules@temp@meaning{26}{\the\numexpr\stex@tempnum-1\@Space}[\notation@uri]
1071             }{
1072                 % Check whether full URI or module?symbol or just name
1073                 \StrCount\modules@getURI@name\@QuestionMark[\isuri@number]
1074                 \ifnum\isuri@number=2
1075                     \edef\notation@uri{\modules@getURI@name}
1076                 \else
1077                     \ifnum\isuri@number=1
1078                         % module?name
1079                         \StrCut\modules@getURI@name\@QuestionMark\isuri@mod\isuri@name
1080                         \ifcsvoid{stex@module@\isuri@mod}{

```

```

1081         \PackageError{stex}{No module with name \isuri@mod\@Space loaded}{}
1082     }{
1083         \expandafter\ifx\csname stex@module@\isuri@mod\endcsname\stex@ambiguous
1084         \PackageError{stex}{Module name \isuri@mod\@Space is ambiguous}{}
1085     }else
1086         \edef\notation@uri{\csname stex@module@\isuri@mod\endcsname\@URI\@QuestionMark\isur
1087     \fi
1088 }
1089 \else
1090     %name
1091     \ifcsvoid{stex@symbol@\modules@getURI@name}{
1092         \PackageError{stex}{No symbol with name \modules@getURI@name\@Space known}{}
1093     }{
1094         \ifcsvoid{\module@uri\@QuestionMark\modules@getURI@name}{
1095             \expandafter\ifx\csname stex@symbol@\modules@getURI@name\endcsname\stex@ambiguous
1096             % Symbol name ambiguous and not in current module
1097             \PackageError{stex}{Symbol name, URI or macroname \detokenize{#1} found!}{}%
1098         }else
1099             % Symbol not in current module, but unambiguous
1100             \edef\notation@uri{\csname stex@symbol@\modules@getURI@name\endcsname}
1101         \fi
1102     }{ % Symbol in current module
1103         \edef\notation@uri{\module@uri\@QuestionMark\modules@getURI@name}
1104     }
1105 }
1106 \fi
1107 \fi
1108 }
1109 }

```

`\notation` Adds a new notation to a symbol foo, as in: `\notation[lang=en,arity=0,variant=op]{foo}{...}`  
`\notation[variant=bar]{foo}[2]{...}` `\notation[args=aia,prec=500;50x49x51]{foo}{#1 bla #2}`  
the actual notation is ultimately stored in `\langle uri \rangle \# \langle variant \rangle`, where `\langle variant \rangle`  
contains `arity`, `lang` and `variant` in that order.

```

1110 \newif\if@innotation\@innotationfalse

```

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

```

1111 \providerobustcmd\notation[2] []{%
1112     \edef\notation@first{#1}%
1113     \edef\notation@second{#2}%
1114     \notation@%
1115 }
1116
1117 \newcommand\notation@[2] [0]{%
1118     \edef\notation@donext{\noexpand\notation@@[\notation@first]%
1119         {\notation@second}[#1]}%
1120     \notation@donext{#2}%
1121 }
1122

```

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*:

```

1123 \def\notation@parse@params#1#2{%
1124   \def\notation@curr@prec{%}%
1125   \def\notation@curr@args{%}%
1126   \def\notation@curr@variant{%}%
1127   \def\notation@curr@arityvar{%}%
1128   \def\notation@curr@provided@arity{#2}%
1129   \def\notation@curr@lang{%}%
1130   \def\notation@options@temp{#1}%
1131   \notation@parse@params@%
1132   \ifx\notation@curr@args\@empty%
1133     \ifx\notation@curr@provided@arity\@empty%
1134       \notation@num@to@ia\notation@curr@arityvar%
1135     \else%
1136       \notation@num@to@ia\notation@curr@provided@arity%
1137     \fi%
1138   \fi%
1139   \StrLen\notation@curr@args[\notation@curr@arity]%
1140 }
1141 \def\notation@parse@params@{%
1142   \IfSubStr\notation@options@temp,{%
1143     \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
1144     \notation@parse@param%
1145     \notation@parse@params@%
1146   }{\ifx\notation@options@temp\@empty\else%
1147     \let\notation@option@temp\notation@options@temp%
1148     \notation@parse@param%
1149   \fi}%
1150 }
1151
1152 \def\notation@parse@param{%
1153   \path@trimstring\notation@option@temp%
1154   \ifx\notation@option@temp\@empty\else%
1155     \IfSubStr\notation@option@temp={%
1156       \StrCut\notation@option@temp=\notation@key\notation@value%
1157       \path@trimstring\notation@key%
1158       \path@trimstring\notation@value%
1159       \IfStrEq\notation@key{prec}{%
1160         \edef\notation@curr@prec{\notation@value}%
1161       }{%
1162         \IfStrEq\notation@key{args}{%
1163           \edef\notation@curr@args{\notation@value}%
1164         }{%
1165           \IfStrEq\notation@key{lang}{%
1166             \edef\notation@curr@lang{\notation@value}%
1167           }{%
1168             \IfStrEq\notation@key{variant}{%

```

```

1169         \edef\notation@curr@variant{\notation@value}%
1170     }{%
1171     \IfStrEq\notation@key{arity}{%
1172         \edef\notation@curr@arityvar{\notation@value}%
1173     }{%
1174     }}}}
1175 }{%
1176     \edef\notation@curr@variant{\notation@option@temp}%
1177 }%
1178 \fi%
1179 }
1180
1181 % converts an integer to a string of 'i's, e.g. 3 => iii,
1182 % and stores the result in \notation@curr@args
1183 \def\notation@num@to@ia#1{%
1184     \IfInteger{#1}{
1185         \notation@num@to@ia@#1%
1186     }{%
1187         %
1188     }%
1189 }
1190 \def\notation@num@to@ia@#1{%
1191     \ifnum#1>0%
1192         \edef\notation@curr@args{\notation@curr@args i}%
1193         \expandafter\notation@num@to@ia@{\expandafter\the\numexpr#1-1\@Space}%
1194     \fi%
1195 }
1196
1197
1198 \newcount\notation@argument@counter
1199
1200 % parses the notation arguments and wraps them in
1201 % \notation@assoc and \notation@argprec for flexary arguments and precedences
1202 \def\notation@@[#1]#2[#3]#4{%
1203     \modules@getURIfromName{#2}%
1204     \notation@parse@params{#1}{#3}%
1205     \let\notation@curr@todo@args\notation@curr@args%
1206     \def\notation@temp@notation{}%
1207     \ex\renewcommand\ex\notation@temp@notation\ex[\notation@curr@arity]{#4}%
1208     % precedence
1209     \let\notation@curr@precstring\notation@curr@args%
1210     \IfSubStr\notation@curr@precstring\notation@curr@prec{%
1211         \StrCut\notation@curr@prec;\notation@curr@prec\notation@curr@prec%
1212         \ifx\notation@curr@prec\@empty\def\notation@curr@prec{}%
1213     }{%
1214         \ifx\notation@curr@prec\@empty%
1215             \ifnum\notation@curr@arity=0\relax%
1216                 \edef\notation@curr@prec{\infprec}%
1217             \else%

```

```

1218         \def\notation@curr@prec{0}%
1219         \fi%
1220     \else%
1221         \edef\notation@curr@prec{\notation@curr@prec}%
1222         \def\notation@curr@prec{}%
1223     \fi%
1224 }%
1225 % arguments
1226 \notation@argument@counter=0%
1227 \def\notation@curr@extargs{}%
1228 \notation@do@args%
1229 }
1230
1231 \edef\notation@ichar{\detokenize{i}}%
1232
1233 % parses additional notation components for (associative) arguments
1234 \def\notation@do@args{%
1235     \advance\notation@argument@counter by 1%
1236     \def\notation@nextarg@temp{}%
1237     \ifx\notation@curr@todo@args\empty%
1238         \ex\notation@after%
1239     \else%
1240         % argument precedence
1241         \IfSubStr\notation@curr@prec{x}{%
1242             \StrCut\notation@curr@prec{x}\notation@curr@argprec\notation@curr@prec%
1243         }{%
1244             \edef\notation@curr@argprec{\notation@curr@prec}%
1245             \def\notation@curr@prec{}%
1246         }%
1247         \ifx\notation@curr@argprec\empty%
1248             \let\notation@curr@argprec\notation@curr@prec%
1249         \fi%
1250         \StrChar\notation@curr@todo@args1[\notation@argchar]%
1251         \edef\notation@argchar{\ex\detokenize\ex{\notation@argchar}}%
1252         \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1253         \ifx\notation@argchar\notation@ichar%
1254             % normal argument
1255             \edef\notation@nextarg@temp{%
1256                 {\stex@arg{\the\notation@argument@counter}{\notation@curr@argprec}{#####\the
1257             }}%
1258             \ex\g@addto@macro@safe\ex\notation@curr@extargs%
1259             \ex{\notation@nextarg@temp}%
1260             \ex\ex\ex\notation@do@args%
1261         \else%
1262             % associative argument
1263             \ex\ex\ex\notation@parse@assocarg%
1264         \fi%
1265     \fi%
1266 }
1267

```

```

1268 \def\notation@parse@assocarg#1{%
1269   \edef\notation@nextarg@temp{%
1270     {\stex@arg{\the\notation@argument@counter}}{\notation@curr@argprec}{\notation@assoc{#1}}{####}
1271   }%
1272   \ex@g@addto@macro@safe\ex\notation@curr@extargs\ex{\notation@nextarg@temp}%
1273   \notation@do@args%
1274 }
1275
1276 \protected\def\safe@newcommand#1{%
1277   \ifdefined#1\ex\renewcommand\else\ex\newcommand\fi#1%
1278 }
1279
1280 % finally creates the actual macros
1281 \def\notation@after{
1282   % \notation@curr@prec
1283   % \notation@curr@args
1284   % \notation@curr@variant
1285   % \notation@curr@arity
1286   % \notation@curr@provided@arity
1287   % \notation@curr@lang
1288   % \notation@uri
1289   \def\notation@temp@fragment{}%
1290   \ifx\notation@curr@arityvar\@empty\else%
1291     \edef\notation@temp@fragment{arity=\notation@curr@arityvar}%
1292   \fi%
1293   \ifx\notation@curr@lang\@empty\else%
1294     \ifx\notation@temp@fragment\@empty%
1295       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1296     \else%
1297       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1298     \fi%
1299   \fi%
1300   \ifx\notation@curr@variant\@empty\else%
1301     \ifx\notation@temp@fragment\@empty%
1302       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1303     \else%
1304       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@variant}%
1305     \fi%
1306   \fi%
1307   \ex\ex\ex\def\ex\ex\ex\notation@temp@notation\ex\ex\ex%
1308   {\ex\notation@temp@notation\notation@curr@extargs}%
1309   \ifnum\notation@curr@arity=0
1310     \edef\notation@temp@notation{\stex@oms{\notation@uri\@Fragment\notation@temp@fragment}}{\notation@curr@argprec}
1311   \else
1312     \edef\notation@temp@notation{\stex@oma{\notation@uri\@Fragment\notation@temp@fragment}}{\notation@curr@argprec}
1313   \fi
1314   \stex@debug{Notation \notation@uri: \meaning\notation@temp@notation}%
1315   \notation@final%
1316   \parsemodule@maybesetcodes%
1317 }

```

```

1318
1319 \def\notation@final{%
1320   \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1321   \stex@debug{Defining \notation@csname of arity \notation@curr@arity}%
1322   \ifcsvoid{\notation@csname}{%
1323     \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1324       \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@curr@arity\ex]%
1325     \ex{\notation@temp\notation}%
1326     \edef\symdecl@temps{%
1327       \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@curr@ari
1328     }%
1329     \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1330     \ex@g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1331   }-%
1332   \PackageWarning{stex}{notation already defined: \notation@csname}{%
1333     Choose a different set of notation options (variant,lang,arity)%
1334   }%
1335 }%
1336 \@innotationfalse%
1337 \if@inimport\else\if@latexml%
1338   \let\notation@simarg@args\notation@curr@args%
1339   \notation@argument@counter=0%
1340   \def\notation@simargs{%
1341     \notation@simulate@arguments%
1342     \latexml\notation\notation@uri\notation@temp@fragment\notation@curr@args\notation@curr@prec
1343     {\csname\notation@csname\ex\endcsname\notation@simargs$}%
1344   \fi\fi%
1345 }
1346 \def\notation@simulate@arguments{%
1347   \ifx\notation@simarg@args\empty\else%
1348     \advance\notation@argument@counter by 1%
1349     \IfBeginWith\notation@simarg@args{i}{%
1350       \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\the\notation@argument
1351     }-%
1352     \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\@Fragment\the\notati
1353   }%
1354   \StrGobbleLeft\notation@simarg@args1[\notation@simarg@args]%
1355   \notation@simulate@arguments%
1356 \fi%
1357 }
1358 % URI, fragment, arity, notation
1359 \def\latexml@notation#1#2#3#4{}

```

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

```

1360 \protected\def\notation@assoc#1#2{% function, argv
1361   \let@tmpop=\relax% do not print the function the first time round
1362   \@for\@I:=#2\do{\@tmpop% print the function
1363     % write the i-th argument with locally updated precedence
1364     \@I%

```

```

1365 \def\@tmpop{#1}%
1366 }%
1367 }%
1368
1369 \def\notation@lparen{()
1370 \def\notation@rparen{)}}
1371 \def\infprec{1000000}
1372 \def\neginfprec{-\infprec}
1373
1374 \newcount\notation@downprec
1375 \notation@downprec=\neginfprec
1376
1377 % patching displaymode
1378 \newif\if@displaymode\@displaymodefalse
1379 \ex\everydisplay\ex{\the\everydisplay\@displaymodetrue}
1380 \let\old@displaystyle\displaystyle
1381 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1382
1383 \protected\def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1384 \def\notation@innertmp{#1}%
1385 \if@displaymode%
1386 \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1387 \ex\notation@resetbrackets\ex\notation@innertmp%
1388 \ex\right\notation@rparen%
1389 \else%
1390 \ex\ex\ex\notation@lparen%
1391 \ex\notation@resetbrackets\ex\notation@innertmp%
1392 \notation@rparen%
1393 \fi%
1394 }
1395
1396 \protected\def\withbrackets#1#2#3{%
1397 \edef\notation@lparen{#1}%
1398 \edef\notation@rparen{#2}%
1399 #3%
1400 \notation@resetbrackets%
1401 }
1402
1403 \protected\def\notation@resetbrackets{%
1404 \def\notation@lparen{()%
1405 \def\notation@rparen{)}}%
1406 }
1407
1408 \protected\def\stex@oms#1#2#3{%
1409 \if@innotation%
1410 \notation@symprec{#2}{#3}%
1411 \else%
1412 \@innotationtrue%
1413 \latexml@oms{#1}{\notation@symprec{#2}{#3}}%
1414 \@innotationfalse%

```



```

1415 \fi%
1416 }
1417
1418 % for LaTeXML Bindings
1419 \def\latexml@oms#1#2{%
1420   #2%
1421 }
1422
1423 \protected\def\stex@oma#1#2#3{%
1424   \if@innotation%
1425     \notation@symprec{#2}{#3}%
1426   \else%
1427     \@innotationtrue%
1428     \latexml@oma{#1}{\notation@symprec{#2}{#3}}%
1429     \@innotationfalse%
1430   \fi%
1431 }
1432
1433 % for LaTeXML Bindings
1434 \def\latexml@oma#1#2{%
1435   #2%
1436 }
1437
1438 \def\notation@symprec#1#2{%
1439   \ifnum#1>\notation@downprec\relax%
1440     \notation@resetbrackets#2%
1441   \else%
1442     \ifnum\notation@downprec=\infprec\relax%
1443       \notation@resetbrackets#2%
1444     \else
1445       \if@inarray@
1446         \notation@resetbrackets#2
1447       \else\dobrackets{#2}\fi%
1448   \fi\fi%
1449 }
1450
1451 \newif\if@inarray@\@inarray@false
1452
1453
1454 \protected\def\stex@arg#1#2#3{%
1455   \@innotationfalse%
1456   \latexml@arg{#1}{\notation@argprec{#2}{#3}}%
1457   \@innotationtrue%
1458 }
1459
1460 % for LaTeXML Bindings
1461 \def\latexml@arg#1#2{%
1462   #2%
1463 }
1464

```

```

1465 \def\notation@argprec#1#2{%
1466   \def\notation@innertmp{#2}
1467   \edef\notation@downprec@temp{\number#1}%
1468   \notation@downprec=\expandafter\notation@downprec@temp%
1469   \expandafter\relax\expandafter\notation@innertmp%
1470   \expandafter\notation@downprec\expandafter=\number\notation@downprec\relax%
1471 }

```

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

```

1472 \protected\def\@invoke@symbol#1{%
1473   \def\@invoke@symbol@first{#1}%
1474   \symbol@args%
1475 }

```

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

```

1476 \newcommand\symbol@args[1][]{%
1477   \notation@parse@params{#1}{}%
1478   \def\notation@temp@fragment{}%
1479   \ifx\notation@curr@arityvar\@empty\else%
1480     \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1481     \fi%
1482   \ifx\notation@curr@lang\@empty\else%
1483     \ifx\notation@temp@fragment\@empty%
1484       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1485     \else%
1486       \edef\notation@temp@fragment{\notation@temp@fragment\& lang=\notation@curr@lang}%
1487     \fi%
1488   \fi%
1489   \ifx\notation@curr@variant\@empty\else%
1490     \ifx\notation@temp@fragment\@empty%
1491       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1492     \else%
1493       \edef\notation@temp@fragment{\notation@temp@fragment\& variant=\notation@curr@va
1494     \fi%
1495   \fi%
1496   %
1497   \ifmmode\def\invoke@symbol@next{\@invoke@symbol@math\@invoke@symbol@first\notation@temp@fragm
1498   \else\def\invoke@symbol@next{\@invoke@symbol@text\@invoke@symbol@first\notation@temp@fragment
1499   \invoke@symbol@next%
1500 }

```

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

```

1501 \def\@invoke@symbol@math#1#2{%
1502   \csname #1\@Fragment#2\endcsname%
1503 }

```

TODO:

```

1504 \def\@invoke@symbol@text#1#2{%

```

1505 }

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

```
1506 \def\setstexlang#1{%
1507   \def\stex@lang{#1}%
1508 }%
1509 \setstexlang{en}
1510 \def\setstexvariant#1#2{%
1511   % TODO
1512 }
1513 \def\setstexvariants#1{%
1514   \def\stex@variants{#1}%
1515 }
```

**Test:**

**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}}

$\bar{A}$ :  $\psi(A)$

$\bar{A}$ :  $\Psi(A)$

\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 }

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

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

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

,\plus {\vard ,\vare }}}\]:

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

### 3.6 Term References

\ifhref

```

1516 \newif\ifhref\hreffalse%
1517 \AtBeginDocument{%
1518   \ifpackageloaded{hyperref}{%
1519     \hreftrue%
1520   }{%
1521     \hreffalse%
1522   }%
1523 }
```

\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.

```

1524 \newbox\stex@targetbox
1525 \def\termref@maketarget#1#2{%
1526   % #1: symbol URI
1527   % #2: text
1528   \stex@debug{Here: #1 <> #2}%
1529   \ifhref\if@smsmode\else%
1530     \hypertarget{sref@#1@target}{#2}%
1531   \fi\fi%
1532   \stex@debug{Here!}%
1533   \expandafter\edef\csname sref@#1\endcsname##1{%
1534     \ifhref\if@smsmode\else\noexpand\hyperlink{sref@#1@target}{##1}\fi\fi%
1535   }%
1536 }
```

\@termref

```

1537 \def\@termref#1#2{%
1538   % #1: symbol URI
1539   % #2: text
1540   \ifcsvoid{#1}{%
1541     \StrCut[2]{#1}\@QuestionMark\termref@mod\termref@name%
1542     \ifcsvoid{\termref@mod}{%
1543       \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
1544     }{%
1545       \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
1546         contains no symbol with name \termref@name.%
1547     }{}%
1548   }%
1549 }{%
1550   \ifcsvoid{sref@#1}{%
```

```

1551      #2% TODO: No reference point exists!
1552    }{%
1553      \csname sref@#1\endcsname{#2}%
1554    }%
1555  }%
1556 }

\tref
1557
1558 \def\@capitalize#1{\uppercase{#1}}%
1559 \newrobustcmd\capitalize[1]{\expandafter\@capitalize #1}%
1560
1561 \newcommand\tref[2][]{%
1562   \edef\tref@name{#1}%
1563   \expandafter\modules@getURIfromName\expandafter{\tref@name}%
1564   \expandafter\@termref\expandafter{\notation@uri}{#2}%
1565 }
1566 \def\trefs#1{%
1567   \modules@getURIfromName{#1}%
1568   % TODO
1569 }
1570 \def\Tref#1{%
1571   \modules@getURIfromName{#1}%
1572   % TODO
1573 }
1574 \def\Trefs#1{%
1575   \modules@getURIfromName{#1}%
1576   % TODO
1577 }

\defi
1578 \addmetakey{defi}{name}
1579 \def\@definiendum#1#2{%
1580   \parsemodule@maybesetcodes%
1581   \stex@debug{Here: #1 | #2}%
1582   \termref@maketarget{#1}{#2}%\termref@maketarget{#1}{\defemph{#2}}%
1583 }
1584
1585 \newcommand\defi[2][]{%
1586   \metasetkeys{defi}{#1}%
1587   \ifx\defi@name\@empty%
1588     \symdecl@constructname{#2}%
1589     \let\defi@name\symdecl@name%
1590     \let\defi@verbalization\symdecl@verbalization%
1591   \else%
1592     \edef\defi@verbalization{#2}%
1593   \fi%
1594   \ifcsvoid{\module@uri\@QuestionMark\defi@name}{%
1595     \symdecl\defi@name%
1596   }{\edef\symdecl@uri{\module@uri\@QuestionMark\defi@name}}%

```

```

1597 \@definiendum\symdecl@uri\defi@verbalization%
1598 }
1599 \def\Defi#1{%
1600 \symdecl{#1}%
1601 \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization}%
1602 }
1603 \def\defis#1{%
1604 \symdecl{#1}%
1605 \@definiendum\symdecl@uri{\symdecl@verbalization s}%
1606 }
1607 \def\Defis#1{%
1608 \symdecl{#1}%
1609 \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization s}%
1610 }

```

### 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
1611 \newif\ifhref\hreffalse%
1612 \AtBeginDocument{%
1613 \ifpackageloaded{hyperref}{%
1614 \hreftrue%
1615 }{%
1616 \hreffalse%
1617 }%
1618 }%
1619 \newcommand\sref@href@ifh[2]{%
1620 \ifhref%
1621 \href{#1}{#2}%
1622 \else%
1623 #2%
1624 \fi%
1625 }%
1626 \newcommand\sref@hlink@ifh[2]{%
1627 \ifhref%
1628 \hyperlink{#1}{#2}%
1629 \else%
1630 #2%
1631 \fi%
1632 }%
1633 \newcommand\sref@target@ifh[2]{%
1634 \ifhref%
1635 \hypertarget{#1}{#2}%
1636 \else%
1637 #2%

```

```

1638 \fi%
1639 }%

```

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.

```

1640 \def\sref@target{%
1641   \ifx\sref@id\@empty%
1642     \relax%
1643   \else%
1644     \edef\@target{sref@\ifcsundef{sref@part}{}\sref@part @}\sref@id @target}%
1645     \sref@target@ifh\@target{}%
1646   \fi%
1647 }%

```

**\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.

```

1648 \addmetakey{srefaddidkey}{prefix}
1649 \newcommand\srefaddidkey[2][]{%
1650   \metasetkeys{srefaddidkey}{#1}%
1651   \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
1652   \metakeys@ext@clear@keys{#2}{id}{}%
1653   \metakeys@ext@showkeys{#2}{id}%
1654   \define@key{#2}{id}{%
1655     \edef\sref@id{\srefaddidkey@prefix ##1}%
1656     \%expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
1657     \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1658   }%
1659 }%

```

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

```

1660 \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.

```

1661 \ifextrefs%
1662   \newwrite\refs@file%
1663 \else%
1664   \def\refs@file{\@auxout}%
1665 \fi%

```

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

```

1666 \newcommand\sref@def[3]{%
1667   \protected@write\refs@file{}\string\sref@def{#1}{#2}{#3}}%
1668 }%

```

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

```

1669 \newcommand\sref@label[2]{%
1670   \sref@def{\ifcsundef\sref@part}{\sref@part @}{#2}{page}{\thepage}%
1671   \sref@def{\ifcsundef\sref@part}{\sref@part @}{#2}{label}{#1}%
1672 }%

```

`\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`.

```

1673 \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.

```

1674 \def\sref@id{} % make sure that defined
1675 \newcommand\sref@label@id[1]{%
1676   \ifx\sref@id\@empty%
1677     \relax%
1678   \else%
1679     \sref@label{#1}{\sref@id}%
1680   \fi%
1681 }%

```

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

```

1682 \newcommand\sref@label@id@arg[2]{%
1683   \def\@id{#2}
1684   \ifx\@id\@empty%
1685     \relax%
1686   \else%
1687     \sref@label{#1}{\@id}%
1688   \fi%
1689 }%

```

### 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`.

```

1690 \newenvironment{modsig}[2] [] {\def\@test{#1}%
1691   \ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1692   \expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
1693   %\ignorespacesandpars
1694 }
1695 {\end{module}}\ignorespacesandpars
1696 }

```



### 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`.

```
1697 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1698 \newrobustcmd\@gimport@star[2] [] {\def\@test{#1}%
1699 \edef\mh@repos{\mh@currentrepos}%
1700 \ifx\@test\@empty%
1701 \importmhmodule[conservative,mhrepos=\mh@repos,path=#2]{#2}%
1702 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1703 \mathhub@setcurrentreposinfo{\mh@repos}%
1704 %\ignorespacesandpars
1705 \parsemodule@maybesetcodes}
1706 \newrobustcmd\@gimport@nostar[2] [] {\def\@test{#1}%
1707 \edef\mh@repos{\mh@currentrepos}%
1708 \ifx\@test\@empty%
1709 \importmhmodule[mhrepos=\mh@repos,path=#2]{#2}%
1710 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1711 \mathhub@setcurrentreposinfo{\mh@repos}%
1712 %\ignorespacesandpars
1713 \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.

```
1714 \def\modules@@first#1/#2;{#1}
1715 \newcommand\libinput[1]{%
1716 \stex@debug{Libinput current repo: \meaning\mh@currentrepos}%
1717 \ifcsvoid{mh@currentrepos}{%
1718 \PackageError{stex}{current MathHub repository not found}{}%
1719 }%
1720 \edef\@mh@group{\expandafter\modules@@first\mh@currentrepos;}
1721 \let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
1722 \def\mh@inffile{\MathHub{\@mh@group/meta-inf/lib/#1}}
```

```

1723 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1724 \IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
1725 \IfFileExists\mh@inffile{}\{\IfFileExists\mh@libfile{}\{%
1726   {\PackageError{stex}
1727     {Library file missing; cannot input #1.tex\MessageBreak%
1728     Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
1729     do not exist}%
1730   {Check whether the file name is correct}}}}
1731 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1732 \let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}

```

### 3.11 omdoc/omgroup

```

1733 \newcount\section@level
1734
1735 \section@level=2
1736 \ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{}
1737 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{}
1738 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{}
1739 \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>*.

```

1740 \newcommand\omgroup@nonum[2]{%
1741 \ifx\hyper@anchor\undefined\else\phantomsection\fi%
1742 \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.

```

1743 \newcommand\omgroup@num[2]{%
1744 \edef\@@ID{\sref@id}
1745 \ifx\omgroup@short\@empty% no short title
1746 \@nameuse{#1}{#2}%
1747 \else% we have a short title
1748 \@ifundefined{rdfmata@sectioning}%
1749   {\@nameuse{#1}[\omgroup@short]{#2}}%
1750   {\@nameuse{rdfmata@#1@old}[\omgroup@short]{#2}}%
1751 \fi%
1752 \sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@@ID}

```

`omgroup`

```

1753 \def\@true{true}
1754 \def\@false{false}
1755 \srefaddidkey{omgroup}
1756 \addmetakey{omgroup}{date}
1757 \addmetakey{omgroup}{creators}

```

```

1758 \addmetakey{omgroup}{contributors}
1759 \addmetakey{omgroup}{srccite}
1760 \addmetakey{omgroup}{type}
1761 \addmetakey*{omgroup}{short}
1762 \addmetakey*{omgroup}{display}
1763 \addmetakey[false]{omgroup}{loadmodules}[true]

\at@begin@omgroup we define a switch for numbering lines and a hook for the beginning of groups:
The \at@begin@omgroup macro allows customization. It is run at the beginning
of the omgroup, i.e. after the section heading.

1764 \newif\if@mainmatter\@mainmattertrue
1765 \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.

1766 \addmetakey{omdoc@sect}{name}
1767 \addmetakey[false]{omdoc@sect}{clear}[true]
1768 \addmetakey{omdoc@sect}{ref}
1769 \addmetakey[false]{omdoc@sect}{num}[true]
1770 \newcommand\omdoc@sectioning[3] [] {\metasetkeys{omdoc@sect}{#1}%
1771 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
1772 \if@mainmatter% numbering not overridden by frontmatter, etc.
1773 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
1774 \def\current@section@level{\omdoc@sect@name}%
1775 \else\omgroup@nonum{#2}{#3}%
1776 \fi}% if@mainmatter

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

1777 \newcommand\omgroup@redefine@addtocontents[1]{%
1778 %\edef\@import{#1}%
1779 %\@for\@I:=\@import\do{%
1780 %\edef\@path{\csname module@\@I @path\endcsname}%
1781 %\@ifundefined{tf@toc}\relax%
1782 % {\protected@write\tf@toc}{\string\@requiremodules{\@path}}}%
1783 %\ifx\hyper@anchor\@undefined% hyperref.sty loaded?
1784 %\def\addcontentsline##1##2##3{%
1785 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}%
1786 %\else% hyperref.sty not loaded
1787 %\def\addcontentsline##1##2##3{%
1788 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}{\@c
1789 %\fi
1790 }% hypreref.sty loaded?

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

1791 \newcount\omgroup@level
1792 \newenvironment{omgroup}[2] [] % keys, title

```

```

1793 {\metasetkeys{omgroup}{#1}\sref@target%
1794 \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.

1795 \ifx\omgroup@loadmodules\@true%
1796 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
1797 {\@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.

1798 \advance\section@level by 1\relax%
1799 \ifcase\section@level%
1800 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
1801 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
1802 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
1803 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
1804 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%
1805 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{#2}%
1806 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{#2}%
1807 \fi% \ifcase
1808 \at@begin@omgroup[#1]\section@level{#2}}% for customization
1809 {\advance\section@level by -1\advance\omgroup@level by -1}

```

and finally, we localize the sections

```

1810 \newcommand\omdoc@part@kw{Part}
1811 \newcommand\omdoc@chapter@kw{Chapter}
1812 \newcommand\omdoc@section@kw{Section}
1813 \newcommand\omdoc@subsection@kw{Subsection}
1814 \newcommand\omdoc@subsubsection@kw{Subsubsection}
1815 \newcommand\omdoc@paragraph@kw{paragraph}
1816 \newcommand\omdoc@subparagraph@kw{subparagraph}

```

\setSGvar set a global variable

```

1817 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}

```

\useSGvar use a global variable

```

1818 \newrobustcmd\useSGvar[1]{%
1819 \@ifundefined{sTeX@Gvar@#1}
1820 {\PackageError{omdoc}
1821 {The sTeX Global variable #1 is undefined}
1822 {set it with \protect\setSGvar}}
1823 \@nameuse{sTeX@Gvar@#1}}

```

blindomgroup

```

1824 \newcommand\at@begin@blindomgroup[1]{%
1825 \newenvironment{blindomgroup}
1826 {\advance\section@level by 1\at@begin@blindomgroup\setion@level}
1827 {\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).

```
1828 \srefaddidkey{omtext}
1829 \addmetakey[] {omtext} {functions}
1830 \addmetakey* {omtext} {display}
1831 \addmetakey {omtext} {for}
1832 \addmetakey {omtext} {from}
1833 \addmetakey {omtext} {type}
1834 \addmetakey* {omtext} {title}
1835 \addmetakey* {omtext} {start}
1836 \addmetakey {omtext} {theory}
1837 \addmetakey {omtext} {continues}
1838 \addmetakey {omtext} {verbalizes}
1839 \addmetakey {omtext} {subject}
```

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

```
1840 \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.

```
1841 \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.

```
1842 \def\omtext@pre@skip{\smallskip}
1843 \def\omtext@post@skip{}
1844 \newenvironment{omtext}[1] [] {\@in@omtexttrue%
1845   \bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
1846   \def\lec##1{\@lec{##1}}%
1847   \omtext@pre@skip\par\noindent%
1848   \ifx\omtext@title\@empty%
1849     \ifx\omtext@start\@empty\else%
1850       \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
1851     \fi% end omtext@start empty
1852   \else\stDMemph{\omtext@title}:\enspace%
1853   \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
1854   \fi% end omtext@title empty
1855   %\ignorespacesandpars
1856 }
1857 {\egroup\omtext@post@skip\@in@omtextfalse%\ignorespacesandpars
1858 }
```

### 3.12.2 Phrase-level Markup

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

```

1859 \srefaddidkey{phrase}
1860 \addmetakey{phrase}{style}
1861 \addmetakey{phrase}{class}
1862 \addmetakey{phrase}{index}
1863 \addmetakey{phrase}{verbalizes}
1864 \addmetakey{phrase}{type}
1865 \addmetakey{phrase}{only}
1866 \newcommand\phrase[2] [] {\metasetkeys{phrase}{#1}%
1867 \ifx\phrase@only\empty\only<\phrase@only>{#2}\else #2\fi}

```

`\coref*`

```

1868 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}
1869 \newcommand\corefs[2]{#1\textsubscript{#2}}
1870 \newcommand\coreft[2]{#1\textsuperscript{#2}}

```

`\n*lex`

```

1871 \newcommand\nlex[1]{\green{\sl{#1}}}
1872 \newcommand\nlcex[1]{*\green{\sl{#1}}}

```

`sinlinequote`

```

1873 \def\@sinlinequote#1{‘‘{\sl{#1}}’’}
1874 \def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
1875 \newcommand\sinlinequote[2] []
1876 {\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.

```

1877 \newcommand\vdec[2] [] {#2}
1878 \newcommand\vrest[2] [] {#2}
1879 \newcommand\vcond[2] [] {#2}

```

EdN:1

`\strucdec` <sup>1</sup>

```

1880 \newcommand\strucdec[2] [] {#2}

```

EdN:2

`\impdec` <sup>2</sup>

```

1881 \newcommand\impdec[2] [] {#2}

```

---

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

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

### 3.12.4 Block-Level Markup

sblockquote

```

1882 \def\begin@sblockquote{\begin{quote}\sl}
1883 \def\end@sblockquote{\end{quote}}
1884 \def\begin@#1{\begin@sblockquote}
1885 \def\end@#1{\def@@lec##1{\textrm{##1}}\@lec{#1}\end@sblockquote}
1886 \newenvironment{sblockquote}[1] []
1887   {\def\@opt{#1}\ifx\@opt\empty\begin@sblockquote\else\begin@sblockquote\@opt\fi}
1888   {\ifx\@opt\empty\end@sblockquote\else\end@sblockquote\@opt\fi}

```

sboxquote

```

1889 \newenvironment{sboxquote}[1] []
1890 {\def@@@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
1891 {\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.

```

1892 \providecommand{\@@lec}[1]{\{#1\}}
1893 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@@lec{#1}}
1894 \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.

```

1895 \addmetakey{omdoc@index}{at}
1896 \addmetakey[false]{omdoc@index}{loadmodules}[true]
1897 \newcommand\omdoc@indexi[2] [] {\ifindex%
1898 \metasetkeys{omdoc@index}{#1}%
1899 \@bsphack\begingroup\@sanitize%
1900 \protected@write\@indexfile{\string\indexentry%
1901 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1902 \ifx\omdoc@index@loadmodules\@true%
1903 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
1904 \else #2\fi% loadmodules
1905 }\@thepage}}%
1906 \endgroup\@esphack\fi}%ifindex
1907 \newcommand\omdoc@indexii[3] [] {\ifindex%
1908 \metasetkeys{omdoc@index}{#1}%

```

```

1909 \@bsphack\beginngroup\@sanitize%
1910 \protected@write\@indexfile{}\string\indexentry%
1911 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1912 \ifx\omdoc@index@loadmodules\@true%
1913 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1914 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
1915 \else #2!#3\fi% loadmodules
1916 }\the page}}%
1917 \endgroup\@esphack\fi}%ifindex
1918 \newcommand\omdoc@indexiii[4] [] {\ifindex%
1919 \metasetkeys{omdoc@index}{#1}%
1920 \@bsphack\beginngroup\@sanitize%
1921 \protected@write\@indexfile{}\string\indexentry%
1922 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1923 \ifx\omdoc@index@loadmodules\@true%
1924 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1925 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1926 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1927 \else #2!#3!#4\fi% loadmodules
1928 }\the page}}%
1929 \endgroup\@esphack\fi}%ifindex
1930 \newcommand\omdoc@indexiv[5] [] {\ifindex%
1931 \metasetkeys{omdoc@index}{#1}%
1932 \@bsphack\beginngroup\@sanitize%
1933 \protected@write\@indexfile{}\string\indexentry%
1934 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1935 \ifx\omdoc@index@loadmodules\@true%
1936 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1937 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1938 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1939 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
1940 \else #2!#3!#4!#5\fi% loadmodules
1941 }\the page}}%
1942 \endgroup\@esphack\fi}%ifindex

```

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

```

\*indi*

1943 \newcommand\aindi[3] [] {{#2}\omdoc@indexi[#1]{#3}}
1944 \newcommand\indi[2] [] {{#2}\omdoc@indexi[#1]{#2}}
1945 \newcommand\indis[2] [] {{#2}\omdoc@indexi[#1]{#2s}}
1946 \newcommand\Indi[2] [] {{\captitalize{#2}}\omdoc@indexi[#1]{#2}}
1947 \newcommand\Indis[2] [] {{\capitalize{#2}}\omdoc@indexi[#1]{#2s}}
1948
1949 \newcommand\@indii[3] [] {\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
1950 \newcommand\aindii[4] [] {{#2}\@indii[#1]{#3}{#4}}
1951 \newcommand\indii[3] [] {{#2 #3}\@indii[#1]{#2}{#3}}
1952 \newcommand\indiis[3] [] {{#2 #3s}\@indii[#1]{#2}{#3}}
1953 \newcommand\Indii[3] [] {{\captitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1954 \newcommand\Indiis[3] [] {{\capitalize{#2 #3}}\@indii[#1]{#2}{#3}}

```



```

1955
1956 \newcommand\@indiii[4] [] {\@omdoc@indexiii[#1]{#2}{#3}{#4}\@omdoc@indexii[#1]{#3}{#2}{#4}}
1957 \newcommand\aindiii[5] [] {\@#2\@indiii[#1]{#3}{#4}{#5}}
1958 \newcommand\indiii[4] [] {\@#2 #3 #4\@indiii[#1]{#2}{#3}{#4}}
1959 \newcommand\indiiis[4] [] {\@#2 #3 #4s\@indiii[#1]{#2}{#3}{#4}}
1960 \newcommand\Indiii[4] [] {\@capitalizet{#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
1961 \newcommand\Indiiis[4] [] {\@capitalizet{#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}
1962
1963 \newcommand\@indiv[5] [] {\@omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
1964 \newcommand\aindiv[6] [] {\@#2\@indiv[#1]{#3}{#4}{#5}{#6}}
1965 \newcommand\indiv[5] [] {\@#2 #3 #4 #5\@indiv[#1]{#2}{#3}{#4}{#5}}
1966 \newcommand\indivs[5] [] {\@#2 #3 #4 #5s\@indiv[#1]{#2}{#3}{#4}{#5}}
1967 \newcommand\Indiv[5] [] {\@capitalizet{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
1968 \newcommand\Indivs[5] [] {\@capitalizet{#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.

```

1969 \newcommand\hateq{\ensuremath{\widehat{=}}\xspace}
1970 \newcommand\hatequiv{\ensuremath{\widehat{equiv}}\xspace}
1971 \@ifundefined{ergo}%
1972 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1973 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1974 \newcommand{\reflect@squig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%
1975 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
1976 \newcommand\notergo{\ensuremath{\not\leadsto}}
1977 \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*
1978 \newcommand\indextoo[2] [] {\@indi[#1]{#2}}%
1979 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}%
1980 \newcommand\indexalt[2] [] {\@aindi[#1]{#2}}%
1981 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead}%
1982 \newcommand\twintoo[3] [] {\@indii[#1]{#2}{#3}}%
1983 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}%
1984 \newcommand\twinalt[3] [] {\@aindii[#1]{#2}{#3}}%
1985 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead}%
1986 \newcommand\atwintoo[4] [] {\@indiii[#1]{#2}{#3}{#4}}%
1987 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead}%
1988 \newcommand\atwinalt[4] [] {\@aindii[#1]{#2}{#3}{#4}}%
1989 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instead}%

```

\my\*graphics

```

1990 \newcommand\mygraphics[2] [] {\includegraphics[#1]{#2}}%
1991 \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics}
1992 \newcommand\mycgraphics[2] [] {\begin{center}\mygraphics[#1]{#2}\end{center}}%
1993 \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics}
1994 \newcommand\mybgraphics[2] [] {\fbox{\mygraphics[#1]{#2}}}%
1995 \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics}
1996 \newcommand\mycbgraphics[2] [] {\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}%
1997 \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphics}

```

## 4 Things to deprecate

Module options:

```

1998 \addmetakey*{module}{id} % TODO: deprecate properly
1999 \addmetakey*{module}{load}
2000 \addmetakey*{module}{path}
2001 \addmetakey*{module}{dir}
2002 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
2003 \addmetakey*{module}{noalign}[true]
2004
2005 \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.

```

2006 %\srefaddidkey{symdef}% what does this do?
2007 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
2008 \define@key{symdef}{noverb}[all]{}%
2009 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
2010 \define@key{symdef}{specializes}{}%
2011 \addmetakey*{symdef}{noalign}[true]
2012 \define@key{symdef}{primary}[true]{}%
2013 \define@key{symdef}{assocarg}{}%
2014 \define@key{symdef}{bvars}{}%
2015 \define@key{symdef}{bargs}{}%
2016 \addmetakey{symdef}{lang}%
2017 \addmetakey{symdef}{prec}%
2018 \addmetakey{symdef}{arity}%
2019 \addmetakey{symdef}{variant}%
2020 \addmetakey{symdef}{ns}%
2021 \addmetakey{symdef}{args}%
2022 \addmetakey{symdef}{name}%
2023 \addmetakey*{symdef}{title}%
2024 \addmetakey*{symdef}{description}%

```

```

2025 \addmetakey{symdef}{subject}%
2026 \addmetakey*{symdef}{display}%
2027 \addmetakey*{symdef}{gfc}%

```

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

```

2028 \def\symdef{\@ifnextchar[{\@symdef}{\@symdef[]}}%
2029 \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.

```

2030 \def\@@symdef[#1]#2[#3]{%
2031   \@insymdef@true%
2032   \metasetkeys{symdef}{#1}%
2033   \edef\symdef@tmp@optpars{\ifcsvoid{symdef@name}{[]}{[name=\symdef@name]}}%
2034   \expandafter\symdecl\symdef@tmp@optpars{#2}%
2035   \@insymdef@false%
2036   \notation[#1]{#2}[#3]%
2037 }% mod@show
2038 \def\symdef@type{Symbol}%
2039 \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`.

```

2040 \def\symvariant#1{%
2041   \@ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}}%
2042 }%
2043 \def\@symvariant#1[#2]#3#4{%
2044   \notation[#3]{#1}[#2]{#4}%
2045 %\ignorespacesandpars
2046 }%

```

`\abbrdef` The `\abbrdef` macro is a variant of `\symdef` that does the same on the  $\text{\LaTeX}$  level.

```

2047 \let\abbrdef\symdef%

```

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

```

2048 \newif\if@importing\@importingfalse
2049 \define@key{symi}{noverb}[all]{}%
2050 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
2051 \define@key{symi}{specializes}{}%
2052 \define@key{symi}{gfc}{}%
2053 \define@key{symi}{noalign}[true]{}%

```

---

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

```

2054 \newcommand\symi{\@ifstar\@symi@star\@symi}
2055 \newcommand\@symi[2] [] {\metasetkeys{symi}{#1}%
2056   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces
2057 }
2058 \newcommand\@symi@star[2] [] {\metasetkeys{symi}{#1}%
2059   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignorespaces
2060 }
2061 \newcommand\symii{\@ifstar\@symii@star\@symii}
2062 \newcommand\@symii[3] [] {\metasetkeys{symi}{#1}%
2063   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespaces
2064 }
2065 \newcommand\@symii@star[3] [] {\metasetkeys{symi}{#1}%
2066   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\ignorespaces
2067 }
2068 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
2069 \newcommand\@symiii[4] [] {\metasetkeys{symi}{#1}%
2070   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignorespaces
2071 }
2072 \newcommand\@symiii@star[4] [] {\metasetkeys{symi}{#1}%
2073   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi\ignorespaces
2074 }
2075 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
2076 \newcommand\@symiv[5] [] {\metasetkeys{symi}{#1}%
2077   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces
2078 }
2079 \newcommand\@symiv@star[5] [] {\metasetkeys{symi}{#1}%
2080   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces
2081 }

```

`\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`.

```

2082 %\srefaddidkey{importmhmodule}%
2083 \addmetakey{importmhmodule}{mhrepos}%
2084 \addmetakey{importmhmodule}{path}%
2085 \addmetakey{importmhmodule}{ext}% why does this exist?
2086 \addmetakey{importmhmodule}{dir}%
2087 \addmetakey[false]{importmhmodule}{conservative}[true]%
2088 \newcommand\importmhmodule[2] [] {%
2089   \parsemodule@maybesetcodes
2090   \metasetkeys{importmhmodule}{#1}%
2091   \ifx\importmhmodule@dir\@empty%
2092     \edef\@path{\importmhmodule@path}%
2093   \else\edef\@path{\importmhmodule@dir/#2}\fi%
2094   \ifx\@path\@empty% if module name is not set
2095     \importmodule[] {#2}{export}%

```

```

2096 \else%
2097 \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
2098 \ifx\importmhmodule@mhrepos\empty% if in the same repos
2099 \relax% no need to change mh@currentrepos, i.e, current directory.
2100 \else%
2101 \mathhub@setcurrentreposinfo\importmhmodule@mhrepos% change it.
2102 \addto@thismodule{x\noexpand\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}}%
2103 \fi%
2104 \@importmodule[\MathHub{\mh@currentrepos/source/\@path}]{#2}{export}%
2105 \mathhub@setcurrentreposinfo\mh@@repos% after importing, reset to old value
2106 \addto@thismodule{x\noexpand\mathhub@setcurrentreposinfo{\mh@@repos}}%
2107 \fi%
2108 %\ignorespacesandpars%
2109 }

\usemhmodule

2110 \addmetakey{importmhmodule}{load}
2111 \addmetakey{importmhmodule}{id}
2112 \addmetakey{importmhmodule}{dir}
2113 \addmetakey{importmhmodule}{mhrepos}
2114
2115 \addmetakey{importmodule}{load}
2116 \addmetakey{importmodule}{id}
2117
2118 \newcommand\usemhmodule[2][{}]{%
2119 \metasetkeys{importmhmodule}{#1}%
2120 \ifx\importmhmodule@dir\empty%
2121 \edef\@path{\importmhmodule@path}%
2122 \else\edef\@path{\importmhmodule@dir/#2}\fi%
2123 \ifx\@path\empty%
2124 \usemodule[id=\importmhmodule@id]{#2}%
2125 \else%
2126 \edef\mh@@repos{\mh@currentrepos}%
2127 \ifx\importmhmodule@mhrepos\empty%
2128 \else\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
2129 \usemodule{\@path\@QuestionMark#2}%
2130 %\usemodule[load=\MathHub{\mh@currentrepos/source/\@path},
2131 % id=\importmhmodule@id]{#2}%
2132 \mathhub@setcurrentreposinfo\mh@@repos%
2133 \fi%
2134 %\ignorespacesandpars
2135 }

\mhinputref

2136 \newcommand\mhinputref[2][{}]{%
2137 \edef\mhinputref@first{#1}%
2138 \ifx\mhinputref@first\empty%
2139 \inputref{#2}%
2140 \else%
2141 \inputref[mhrepos=\mhinputref@first]{#2}%

```

```

2142 \fi%
2143 }

\trefi*
2144 \newcommand\trefi[2] [] {%
2145 \edef\trefi@mod{#1}%
2146 \ifx\trefi@mod\empty\tref{#2}\else\tref{#1\@QuestionMark#2}\fi%
2147 }
2148 \newcommand\trefii[3] [] {%
2149 \edef\trefii@mod{#1}%
2150 \ifx\trefii@mod\empty\tref{#2-#3}\else\tref{#1\@QuestionMark#2-#3}\fi%
2151 }

\defi*
2152 \def\defii#1#2{\defi{#1!#2}}
2153 \def\Defii#1#2{\Defi{#1!#2}}
2154 \def\defiis#1#2{\defis{#1!#2}}
2155 \def\Defiis#1#2{\Defis{#1!#2}}
2156 \def\defiii#1#2#3{\defi{#1!#2!#3}}
2157 \def\Defiii#1#2#3{\Defi{#1!#2!#3}}
2158 \def\defiiis#1#2#3{\defis{#1!#2!#3}}
2159 \def\Defiiis#1#2#3{\Defis{#1!#2!#3}}
2160 \def\defiv#1#2#3#4{\defi{#1!#2!#3!#4}}
2161 \def\Defiv#1#2#3#4{\Defi{#1!#2!#3!#4}}
2162 \def\defivs#1#2#3#4{\defis{#1!#2!#3!#4}}
2163 \def\Defivs#1#2#3#4{\Defis{#1!#2!#3!#4}}
2164 \def\adefi#1#2{\defi[name=#2]{#1}}
2165 \def\adefii#1#2#3{\defi[name=#2-#3]{#1}}
2166 \def\adefiii#1#2#3#4{\defi[name=#2-#3-#4]{#1}}
2167 \def\adefiv#1#2#3#4#5{\defi[name=#2-#3-#4-#5]{#1}}

2168 \newlinechar=\old@newlinechar

```