

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

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

March 23, 2021

## Abstract

TODO

---

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

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>User commands</b>	<b>3</b>
<b>3</b>	<b>Implementation</b>	<b>3</b>
3.1	sTeX base . . . . .	4
3.2	Paths and URIs . . . . .	4
3.3	Modules . . . . .	16
3.4	Inheritance . . . . .	21
3.5	Symbols/Notations/Verbalizations . . . . .	32
3.6	Verbalizations . . . . .	45
3.7	Term References . . . . .	47
3.8	sref . . . . .	50
3.9	smultiling . . . . .	52
3.10	smglom . . . . .	52
3.11	mathhub . . . . .	53
3.12	omdoc/omgroup . . . . .	54
3.13	omtext . . . . .	56
<b>4</b>	<b>Things to deprecate</b>	<b>62</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[border=1px,varwidth]{standalone}
3 \setlength\textwidth{15cm}
4 \g@addto@macro{\@parboxrestore}{\setlength\parskip{\baselineskip}}
5 \RequirePackage{stex}
6 </cls>
7 <*package>
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 \ifcsname if@latexml\endcsname\else
22   \ex\newif\csname if@latexml\endcsname\@latexmlfalse
23 \fi
```

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

```
24 \protected\long\def\latexml@annotate#1#2#3{%
25   \def\latexml@annotate@bodyarg{#3}%
26   \if@latexml\ifmmode\latexml@annotate@math{#1}{#2}{\ifx\latexml@annotate@bodyarg\@empty\ \else
27 }
28 \protected\long\def\latexml@annotate@text#1#2#3{}
29 \protected\long\def\latexml@annotate@math#1#2#3{}
30 \newenvironment{latexml@annotateenv}[2]{\{}}{\}
31 \protected\long\def\latexml@annotate@invisible#1#2#3{}
32 \RequirePackage{xspace}
33 \RequirePackage{standalone}
34 \RequirePackageWithOptions{stex-metakeys}
35 \if@latexml\else\RequirePackage{xstring}\fi
36 \RequirePackage{etoolbox}
```

### 3.1 sTeX base

The sTeX logo:

```
37 \protected\def\stex{%
38   \@ifundefined{texorpdfstring}%
39   {\let\texorpdfstring\@firstoftwo}%
40   }%
41   \texorpdfstring{\raisebox{-.5ex}{S}\kern-.5ex\TeX}{sTeX}\xspace%
42 }
43 \def\sTeX{\stex}
```

### 3.2 Paths and URIs

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

```
44 \def\pathsuris@setcatcodes{%
45   \edef\pathsuris@oldcatcode@hash{\the\catcode'\#}%
46   \catcode'\#=12\relax%
47   \edef\pathsuris@oldcatcode@slash{\the\catcode'\/%}%
48   \catcode'\/=12\relax%
49   \edef\pathsuris@oldcatcode@colon{\the\catcode'\:%}%
50   \catcode'\:=12\relax%
51   \edef\pathsuris@oldcatcode@qm{\the\catcode'\?}%
52   \catcode'\?=12\relax%
53 }
54 \def\pathsuris@resetcatcodes{%
55   \catcode'\#\pathsuris@oldcatcode@hash\relax%
```

```

56 \catcode'\pathsuris@oldcatcode@slash\relax%
57 \catcode'\pathsuris@oldcatcode@colon\relax%
58 \catcode'\pathsuris@oldcatcode@qm\relax%
59 }

```

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

```
\defpath{MathHub}{/path/to/localmh/MathHub}
```

then we can use `\MathHub` to form other paths, for example,

```
\MathHub{source/smgglom/sets}
```

will generate `/path/to/localmh/MathHub/source/smgglom/sets`.

```

60 \def\namespace@read#1{%
61   \edef\namespace@read@path{#1}%
62   \edef\namespace@read@path{\ex\detokenize\ex\namespace@read@path}}%
63   \namespace@continue%
64 }
65 \def\namespace@continue{%
66   \pathsuris@resetcatcodes%
67   \ex\edef\csname\namespace@macroname\endcsname##1{%
68     \namespace@read@path\@Slash##1%
69   }%
70 }
71 \protected\def\namespace#1{%
72   \def\namespace@macroname{#1}%
73   \pathsuris@setcatcodes%
74   \namespace@read%
75 }
76 \let\defpath\namespace

```

### 3.2.1 Path Canonicalization

We define some macros for later comparison.

```

77 \pathsuris@setcatcodes
78 \def\@ToTop{..}
79 \def\@Slash{/}
80 \def\@Colon{:}
81 \def\@Space{ }
82 \def\@QuestionMark{?}
83 \def\@Dot{.}
84 \catcode'\&=12
85 \def\@Ampersand{&}
86 \catcode'\&=4
87 \def\@Fragment{#}
88 \pathsuris@resetcatcodes
89 \catcode'\.=0
90 .catcode'\.=12

```

```

91 .let.\@BackSlash\
92 .catcode'\.=0
93 \catcode'\.=12
94 \edef\old@percent@catcode{\the\catcode'\}%
95 \catcode'\%=12
96 \let\@Percent%
97 \catcode'\%=\old@percent@catcode

```

\@cpath Canonicalizes (file) paths:

```

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

```

```

139         \edef\pathsuris@cpath@temp%
140         {\@cpath@path\pathsuris@cpath@temp}%
141     }{%
142         \ifx\@cpath@path\@empty\else%
143             \edef\pathsuris@cpath@temp%
144             {\@cpath@path\@Slash\pathsuris@cpath@temp}%
145         \fi%
146     }%
147     \def\@cpath@path{}%
148     \@cpath@loop%
149 }{%
150     \ifx\@cpath@path\@empty%
151         \edef\@cpath@path{\pathsuris@cpath@temp@a}%
152     \else%
153         \edef\@cpath@path%
154         {\@cpath@path\@Slash\pathsuris@cpath@temp@a}%
155     \fi%
156     \@cpath@loop%
157 }%
158 \fi\fi%
159 }{%
160     \ifx\@cpath@path\@empty%
161         \edef\@cpath@path{\pathsuris@cpath@temp}%
162     \else%
163         \edef\@cpath@path{\@cpath@path\@Slash\pathsuris@cpath@temp}%
164     \fi%
165 }%
166 }

```

### Test 1:

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

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

```

167 \newcommand\cpath@print[1]{%
168     \@cpath{#1}%
169     \@cpath@path%

```

```
170 }
```

`\path@filename`

```
171 \def\path@filename#1#2{%
172   \edef\filename@oldpath{#1}%
173   \StrCount\filename@oldpath\@Slash[\filename@lastslash]%
174   \ifnum\filename@lastslash>0%
175     \StrBehind[\filename@lastslash]\filename@oldpath%
176     \@Slash[\filename@oldpath]%
177     \edef#2{\filename@oldpath}%
178   \else%
179     \edef#2{\filename@oldpath}%
180   \fi%
181 }
```

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

Filename: baz.tex

`\path@filename@noext`

```
182 \def\path@filename@noext#1#2{%
183   \path@filename{#1}{#2}%
184   \edef\filename@oldpath{#2}%
185   \StrCount\filename@oldpath\@Dot[\filename@lastdot]%
186   \ifnum\filename@lastdot>0%
187     \StrBefore[\filename@lastdot]\filename@oldpath%
188     \@Dot[\filename@oldpath]%
189     \edef#2{\filename@oldpath}%
190   \else%
191     \edef#2{\filename@oldpath}%
192   \fi%
193 }
```

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

Filename: baz

### 3.2.2 Windows

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

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

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

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

```
196 \newif\if@windowstopath@inpath@
197 \def\windows@to@path#1{%
```



```

198     \@windowstopath@inpath@false%
199     \def\windows@temp{}%
200     \edef\windows@path{#1}%
201     \ifx\windows@path\@empty\else%
202         \ex\windows@path@loop\windows@path\windows@path@end%
203     \fi%
204     \let#1\windows@temp%
205 }
206 \def\windows@path@loop#1#2\windows@path@end{%
207     \def\windows@temp@b{#2}%
208     \ifx\windows@temp@b\@empty%
209         \def\windows@continue{}%
210     \else%
211         \def\windows@continue{\windows@path@loop#2\windows@path@end}%
212     \fi%
213     \if@windowstopath@inpath@%
214         \ifx#1\@BackSlash%
215             \edef\windows@temp{\windows@temp\@Slash}%
216         \else%
217             \edef\windows@temp{\windows@temp#1}%
218         \fi%
219     \else%
220         \ifx#1:%
221             \edef\windows@temp{\@Slash\windows@temp}%
222             \@windowstopath@inpath@true%
223         \else%
224             \edef\windows@temp{\windows@temp#1}%
225         \fi%
226     \fi%
227     \windows@continue%
228 }

```

**Test 5:** Input: C:\foo \bar .baz  
Output: /C/foo/bar.baz

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

```

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

```

```

242     \def\windows@continue{}%
243 \else%
244     \def\windows@continue{\path@windows@loop#2\windows@path@end}%
245 \fi%
246 \if@windowstopath@inpath%
247     \ifx#1/%
248         \edef\windows@temp{\windows@temp\@BackSlash}%
249     \else%
250         \edef\windows@temp{\windows@temp#1}%
251     \fi%
252 \else%
253     \ifx#1/%
254         \edef\windows@temp{\windows@temp:\@BackSlash}%
255         \@windowstopath@inpath@true%
256     \else%
257         \edef\windows@temp{\windows@temp#1}%
258     \fi%
259 \fi%
260 \windows@continue%
261 }

```

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

Output: C:\foo\bar.baz

### 3.2.3 Auxiliary methods

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

```

262 \def\path@trimstring#1{%
263     \edef\pathsuris@trim@temp{#1}%
264     \IfBeginWith\pathsuris@trim@temp\@Space{%
265         \StrGobbleLeft\pathsuris@trim@temp1[#1]%
266         \path@trimstring{#1}%
267     }{%
268         \IfEndWith\pathsuris@trim@temp\@Space{%
269             \StrGobbleRight\pathsuris@trim@temp1[#1]%
270             \path@trimstring{#1}%
271         }{%
272             \edef#1{\pathsuris@trim@temp}%
273         }%
274     }%
275 }

```

**Test 7:** `>foo bar<`

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

```

276 %\if@latexml\else
277 \def\@kpsewhich#1#2{\begingroup%
278     \edef\kpsewhich@cmd{"|kpsewhich #2"}%
279     \everyeof{\noexpand}%

```

```

280 \catcode'\=12%
281 \edef#1{\input\kpsewhich@cmd\@Space}%
282 \path@trimstring#1%
283 \if@iswindows@ \windows@to@path#1\fi%
284 \xdef#1{\ex\detokenize\expandafter{#1}}%
285 \endgroup}
286 %\fi

```

**Test 8:** [/usr/share/texlive/texmf-dist/tex/latex/etoolbox/etoolbox.sty](#)

### 3.2.4 sTeX input hooks

We determine the PWD of the current main document:

```

287 \edef\pwd@cmd{\if@iswindows@ -expand-var \@Percent%
288   CD\@Percent\else -var-value PWD\fi}
289 \@kpsewhich\stex@PWD\pwd@cmd
290 \edef\stex@mainfile{\stex@PWD\@Slash\jobname}
291 \edef\stex@mainfile{\ex\detokenize\ex{\stex@mainfile}}

```

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

We keep a stack of \inputed files:

```

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

```

```

319 \StrLen\stex@currfilename[\stex@currfile@tmp]%
320 \StrGobbleRight\stex@currfile{\the\numexpr%
321 \stex@currfile@tmp+1 }[\stex@currpath]%
322 \global\let\stex@currfile\stex@currfile%
323 \global\let\stex@currpath\stex@currpath%
324 \global\let\stex@currfilename\stex@currfilename%
325 \fi%
326 }

```

`\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:

```

327 \def\stexinput#1{%
328 \stex@iffileexists{#1}{%
329 \stex@currfile@push\stex@temp@path%
330 \input{\stex@currfile}%
331 \stex@currfile@pop%
332 }%
333 {%
334 \PackageError{stex}{File does not exist %
335 (#1): \stex@temp@path}{}%
336 }%
337 }
338 \def\stex@iffileexists#1#2#3{%
339 \edef\stex@temp@path{#1}%
340 \if@iswindows@\path@to@windows\stex@temp@path\fi%
341 \IfFileExists\stex@temp@path{#2}{#3}%
342 }
343 \stex@currfile@pop

```

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

### 3.2.5 MathHub repositories

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

```

344 \@kpsewhich\mathhub@path{--var-value MATHHUB}
345 \if@iswindows@\windows@to@path\mathhub@path\fi
346 \ifx\mathhub@path\empty
347 \PackageWarning{stex}{MATHHUB system variable not %
348 found or wrongly set}{%
349 \defpath{MathHub}{%
350 \else\defpath{MathHub}\mathhub@path\fi

```

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

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

```

351 \def\mathhub@findmanifest#1{%
352   \@cpath{#1}%
353   \ifx\@cpath@path\@Slash%
354     \def\manifest@mf{%
355   \else\ifx\@cpath@path\@empty%
356     \def\manifest@mf{%
357   \else%
358     \edef\@findmanifest@path{\@cpath@path/MANIFEST.MF}%
359     \if@iswindows\@path@to@windows\@findmanifest@path\fi%
360     \IfFileExists{\@findmanifest@path}{%
361       \edef\manifest@mf{\@findmanifest@path}%
362       \xdef\temp@archive@dir{\ex\detokenize\ex{\@cpath@path}}%
363     }{%
364       \edef\@findmanifest@path{\@cpath@path/META-INF/MANIFEST.MF}%
365       \if@iswindows\@path@to@windows\@findmanifest@path\fi%
366       \IfFileExists{\@findmanifest@path}{%
367         \edef\manifest@mf{\@findmanifest@path}%
368         \xdef\temp@archive@dir{\ex\detokenize\ex{\@cpath@path}}%
369       }{%
370         \edef\@findmanifest@path{\@cpath@path/meta-inf/MANIFEST.MF}%
371         \if@iswindows\@path@to@windows\@findmanifest@path\fi%
372         \IfFileExists{\@findmanifest@path}{%
373           \edef\manifest@mf{\@findmanifest@path}%
374           \xdef\temp@archive@dir{\ex\detokenize\ex{\@cpath@path}}%
375         }{%
376           \mathhub@findmanifest{\@cpath@path/..}%
377         }}}}
378 \fi\fi%
379 }

```

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

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

```

380 \def\split@manifest@key{%
381   \IfSubStr{\manifest@line}{\@Colon}{%
382     \StrBefore{\manifest@line}{\@Colon}[\manifest@key]%
383     \StrBehind{\manifest@line}{\@Colon}[\manifest@line]%
384     \path@trimstring\manifest@line%
385     \path@trimstring\manifest@key%
386   }{%
387     \def\manifest@key{%
388   }%
389 }

```

the next helper function iterates over lines in MANIFEST.MF

```

390 \def\parse@manifest@loop{%
391   \ifeof\@manifest%
392   \else%

```

```

393 \read\@manifest to \manifest@line\relax%
394 \split\manifest@key%
395 % id
396 \IfStrEq\manifest@key{id}{%
397 \xdef\manifest@mf{id}{\manifest@line}%
398 }{%
399 % narration-base
400 \IfStrEq\manifest@key{narration-base}{%
401 \xdef\manifest@mf@narr{\manifest@line}%
402 }{%
403 % namespace
404 \IfStrEq\manifest@key{source-base}{%
405 \xdef\manifest@mf@ns{\manifest@line}%
406 }{%
407 \IfStrEq\manifest@key{ns}{%
408 \xdef\manifest@mf@ns{\manifest@line}%
409 }{%
410 % dependencies
411 \IfStrEq\manifest@key{dependencies}{%
412 \xdef\manifest@mf@deps{\manifest@line}%
413 }{%
414 }}}}
415 \parse\manifest@loop%
416 \fi%
417 }

```

`\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.

```

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

```

```

439     \cslet{#1dir}\temp@archive@dir%
440   }%
441 }

```

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

```

\mathhub@setcurrentreposinfo \mathhub@setcurrentreposinfo{<id>} sets the current repository to <id>, checks
                             if the MANIFEST.MF of this repository has already been read, and if not, finds it,
                             parses it and stores the values in \currentrepos@<key>@<id> for later retrieval.
442 \def\mathhub@setcurrentreposinfo#1{%
443   \edef\mh@currentrepos{#1}%
444   \ifx\mh@currentrepos\empty%
445     \edef\currentrepos@dir{\@Dot}%
446     \def\currentrepos@narr{}%
447     \def\currentrepos@ns{}%
448     \def\currentrepos@id{}%
449     \def\currentrepos@deps{}%
450   \else%
451     \ifcsdef{mathhub@dir@\mh@currentrepos}{%
452       \@inmhrepostrue
453       \ex\let\ex\currentrepos@dir\csname mathhub@dir@#1\endcsname%
454       \ex\let\ex\currentrepos@narr\csname mathhub@narr@#1\endcsname%
455       \ex\let\ex\currentrepos@ns\csname mathhub@ns@#1\endcsname%
456       \ex\let\ex\currentrepos@deps\csname mathhub@deps@#1\endcsname%
457     }{%
458       \mathhub@parsemanifest{currentrepos@}{\MathHub{#1}}%
459       \@setcurrentreposinfo%
460       \ifcsvoid{currentrepos@dir}{\PackageError{stex}{No archive with %
461         name #1 found!}{make sure that #1 is directly in your MATHHUB folder %
462         and contains a MANIFEST.MF, either directly in #1 or in a meta-inf %
463         subfolder.}}{\@inmhrepostrue}%
464     }%
465   \fi%
466 }
467
468 \def\@setcurrentreposinfo{%
469   \edef\mh@currentrepos{\currentrepos@id}%
470   \ifcsvoid{currentrepos@dir}{%
471     \csxdef{mathhub@dir@\currentrepos@id}{\currentrepos@dir}%
472     \csxdef{mathhub@narr@\currentrepos@id}{\currentrepos@narr}%
473     \csxdef{mathhub@ns@\currentrepos@id}{\currentrepos@ns}%
474     \csxdef{mathhub@deps@\currentrepos@id}{\currentrepos@deps}%
475   }%
476 }

```

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

```

477 \newif\if@inmhrepos\@inmhreposfalse

```

```

478 \ifcsvoid{stex@PWD}{}{
479 \mathhub@parsemanifest{currentrepos@}\stex@PWD
480 \@setcurrentreposinfo
481 \ifcsvoid{currentrepos@dir}{\message{sTeX: Not currently in a MathHub repository}}{%
482 \message{Current sTeX repository: \mh@currentrepos}
483 }
484 }

```

### 3.3 Modules

```

485 \ifmod@show\if@latexml\else\RequirePackage{mdframed}\fi\fi
    Aux:
486 %\def\ignorespacesandpars{\begingroup\catcode13=10%
487 % \ifnextchar\relax{\endgroup}\endgroup}}

    and more adapted from http://tex.stackexchange.com/questions/179016/ignore-spaces-and-pars-after-an-environment
488 %\def\ignorespacesandparsafterend#1\ignorespaces\fi{#1%
489 % \fi\ignorespacesandpars}
490 %\def\ignorespacesandpars{\ifhmode\unskip\fi@ifnextchar\par%
491 % {\ex\ignorespacesandpars@gobble}{}}

    Options for the module-environment:
492 \addmetakey*{module}{title}
493 \addmetakey*{module}{name}
494 \addmetakey*{module}{creators}
495 \addmetakey*{module}{contributors}
496 \addmetakey*{module}{srccite}
497 \addmetakey*{module}{ns}
498 \addmetakey*{module}{narr}

```

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

```

499 \ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%
500 \newrobustcmd\module@heading{%
501 \stepcounter{module}%
502 \ifmod@show%
503 \noindent{\textbf{Module} \thesection.\thetitle [\module@name}}%
504 \sref@label@id{Module \thesection.\thetitle [\module@name}}%
505 \ifx\module@title\empty : \quad\else\quad(\module@title)\hfill\\fi%
506 \fi%
507 }%

```

#### Test 14: Module 3.1[Test]: Foo

`module` Finally, we define the begin module command for the module environment. Much of the work has already been done in the keyval bindings, so this is quite simple.

```

508 \newenvironment{module}[1][{}]{
509 \begin{@module}[#1]%
510 \module@heading% make the headings
511 %\ignorespacesandpars

```



```

512 \parsemodule@maybesetcodes}{%
513 \end{@module}%
514 \ignorespacesafterend%
515 }%
516 \ifmod@show\surroundwithhmdframed{module@om@common}\fi%

```

Some auxiliary methods:

```

517 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}
518 \def\addto@thismodule#1{%
519 \ifundefined{this@module}{\fi}%
520 \expandafter\g@addto@macro@safe\this@module{#1}%
521 }%
522 }
523 \def\addto@thismodulex#1{%
524 \ifundefined{this@module}{\fi}%
525 \edef\addto@thismodule@exp{#1}%
526 \expandafter\expandafter\expandafter\g@addto@macro@safe%
527 \expandafter\this@module\expandafter{\addto@thismodule@exp}%
528 }}

```

**@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`.

```

529 \newif\ifarchive@ns@empty@ \archive@ns@empty@false
530 \def\set@default@ns{%
531 \edef\@module@ns@temp{\stex@currpath}%
532 \if@iswindows@\windows@to@path\@module@ns@temp\fi%
533 \archive@ns@empty@false%
534 \stex@debug{Generate new namespace^^J Filepath: \@module@ns@temp}%
535 \ifcsvoid{mh@currentrepos}{\archive@ns@empty@true}%
536 {\ex\ifx\csname mathhub@ns@\mh@currentrepos\endcsname\@empty\archive@ns@empty@true\fi%
537 }%
538 \stex@debug{ \ifarchive@ns@empty@ Namespace empty\else Namespace not empty\fi}%
539 \ifarchive@ns@empty@%
540 \edef\@module@ns@tempuri{file\@Colon\@Slash\@Slash\@module@ns@temp}%
541 \else%
542 \edef\@module@filepath@temppath{\@module@ns@temp}%
543 \edef\@module@ns@tempuri{\csname mathhub@ns@\mh@currentrepos\endcsname}%
544 \edef\@module@archivedirpath{\csname mathhub@dir@\mh@currentrepos\endcsname\@Slash source}%
545 \edef\@module@archivedirpath{\ex\detokenize\ex{\@module@archivedirpath}}%
546 \IfBeginWith\@module@filepath@temppath\@module@archivedirpath{%

```

```

547      \StrLen\@module@archivedirpath[\ns@temp@length]%
548      \StrGobbleLeft\@module@filepath@temppath\ns@temp@length[\@module@filepath@temprest]%
549      \edef\@module@ns@tempuri{\@module@ns@tempuri\@module@filepath@temprest}%
550    }{}%
551  \fi%
552  \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]}%
553  \setkeys{module}{ns=\@module@ns@tempuri}%
554 }

```

**Test 15:** <file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master>

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

```

555 \def\set@next@moduleid{%
556   \path@filename@noext\stex@currfile\stex@next@moduleid@filename%
557   \edef\set@next@moduleid@csname{namespace@\module@ns\@QuestionMark\stex@next@moduleid@filename}%
558   \unless\ifcsname\set@next@moduleid@csname\endcsname%
559     \csgdef{\set@next@moduleid@csname}{0}%
560   \fi%
561   \edef\namespace@currnum{\csname\set@next@moduleid@csname\endcsname}%
562   \edef\module@temp@setidname{\noexpand\setkeys{module}{name=%
563     \stex@next@moduleid@filename\ex\unless\ex\ifnum\namespace@currnum\set@next@moduleid@csname\endcsname=0.
564   \module@temp@setidname}%
565   \csxdef{\set@next@moduleid@csname}{\the\numexpr\namespace@currnum+1}%
566 }

```

**Test 16:** `stex`  
`stex.1`

Finally, the `@module` environment does the actual work, i.e. setting metakeys, computing namespace/id, defining `\this@module`, etc.

For a module with name  $\langle name \rangle$  (`\module@name`) and uri  $\langle uri \rangle$  (`\module@uri`), this defines the following macros:

- `\module@defs@ $\langle uri \rangle$`  that acts as a repository for semantic macros of the current module. It will be called by `\importmodule` to activate them.
- We will add the internal forms of the semantic macros whenever `\symdef` is invoked. To do this, we will need an unexpanded form `\this@module` that expands to `\module@defs@ $\langle uri \rangle$` ; we define it first and then initialize `\module@defs@ $\langle uri \rangle$`  as empty.
- `\module@names@ $\langle uri \rangle$`  will store all symbol names declared in this module.
- `\module@imports@ $\langle uri \rangle$`  will store the URIs of all modules directly included in this module
- `\mathop{\langle uri \rangle}` that expands to `\invoke@module{\mathop{\langle uri \rangle}}` (see below).
- `\stex@module@ $\langle name \rangle$`  that expands to  $\langle uri \rangle$ , if unambiguous, otherwise to ambiguous.

If we are currently in a mathhub repository, this information will also be stored in `\module@defs@<uri>`, so we can resolve includes properly when this module is activated.

```

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

```

```

613 \let\currentmodule@uri\module@uri%
614 \stex@debug{^^JNew module: \module@uri^^J}%
615 \parsemodule@maybesetcodes%
616 \begin{latexml@annotateenv}{theory}{\module@uri}%
617 }{%
618 \end{latexml@annotateenv}%
619 \if@inmhrepos%
620 \@inmhreposfalse%
621 \addto@thismodule{x\noexpand\mathhub@setcurrentreposinfo{\expandafter\noexpand\csname mh@old@
622 \fi%
623 }%
624 \newenvironment{@structural@feature}[2]{%
625 \ifcsvoid{currentmodule@uri}{%
626 \set@default@ns\let\currentmodule@ns\module@ns%
627 \set@next@moduleid\let\currentmodule@name\module@name%
628 }{%}%
629 \edef\currentmodule@name{\currentmodule@name\@Slash#2\_feature}%
630 \parsemodule@maybesetcodes%
631 \begin{latexml@annotateenv}{feature:#1}{\currentmodule@uri\QuestionMark#2}%
632 \edef\currentmodule@uri{\currentmodule@ns\@QuestionMark\currentmodule@name}%
633 \parsemodule@maybesetcodes%
634 }{%
635 \end{latexml@annotateenv}%
636 }%
637 \newcommand\structural@feature[3]{\begingroup%
638 \ifcsvoid{currentmodule@uri}{%
639 \set@default@ns\let\currentmodule@ns\module@ns%
640 \set@next@moduleid\let\currentmodule@name\module@name%
641 }{%}%
642 \edef\currentmodule@name{\currentmodule@name\@Slash#2\_feature}%
643 \parsemodule@maybesetcodes%
644 \latexml@annotate{feature:#1}{\currentmodule@uri\QuestionMark#2}{%
645 \edef\currentmodule@uri{\currentmodule@ns\@QuestionMark\currentmodule@name}%
646 #3}%
647 \endgroup}

```

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

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

this@module: »macro:->«

**Test 18: Faking a MathHub archive Foo/Bar with URI http://foo.bar/baz:**

**Module 3.3[Foo2]:**

Name: Foo2

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

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

**Test 19: Removing the \MathHub system variable first:**

**Module 3.4**[Foo]:

Name: Foo

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

this@module: »macro:->«

**Test 20:** Faking a MathHub archive Foo/Bar with URI <http://foo.bar/baz>:

**Module 3.5**[Foo2]:

Name: Foo2

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

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

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

```
648 \def\@URI{uri} % TODO check this
649 \def\@invoke@module#1#2{%
650   \ifx\@URI#2%
651     #1%
652   \else%
653     % TODO something else
654     #2%
655   \fi%
656 }
```

## 3.4 Inheritance

### 3.4.1 Selective Inclusion

The next great goal is to establish the  $\backslash\text{requiremodules}$  macro, which reads an  $\text{\LaTeX}$  file and processes all the module signature information in them, but does not produce any output. This is a tricky business, as we need to “parse” the modules and treat the module signature macros specially (we refer to this as “**sms mode**”, since it is equivalent to what the – now deprecated – **sms** utility did).

In the following we introduce a lot of auxiliary functionality before we can define  $\backslash\text{requiremodules}$ .

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

```
657 \newif\if@smsmode\@smsmodefalse
658 \def\parsemodule@allow#1{%
659   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#1\endcsname}%
660 }
661 \def\parsemodule@allowenv#1{%
662   \ex\def\csname parsemodule@allowedenv@#1\endcsname{\csname#1\endcsname}%
663 }
```

```

663 }
664 \def\parsemodule@replacemacro#1#2{%
665   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#2\endcsname}%
666 }
667 \def\parsemodule@replaceenv#1#2{%
668   \ex\def\csname parsemodule@allowedenv@#1\endcsname{#2}%
669 }
670 \def\parsemodule@escapechar@beginstring{begin}
671 \def\parsemodule@escapechar@endstring{end}

```

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

```

672 \parsemodule@allow{symdef}
673 \parsemodule@allow{abbrdef}
674 \parsemodule@allow{importmodule}
675 \parsemodule@allowenv{module}
676 \parsemodule@allowenv{@module}
677 \parsemodule@allow{importmhmodule}
678 \parsemodule@allow{gimport}
679 \parsemodule@allowenv{modsig}
680 \parsemodule@allowenv{mhmodsig}
681 \parsemodule@allowenv{mhmodnl}
682 \parsemodule@allowenv{modnl}
683 \parsemodule@allowenv{@structural@feature}
684 \parsemodule@allow{symvariant}
685 \parsemodule@allow{structural@feature}
686 \parsemodule@allow{symi}
687 \parsemodule@allow{symii}
688 \parsemodule@allow{symiii}
689 \parsemodule@allow{symiv}
690 \parsemodule@allow{notation}
691 \parsemodule@allow{symdecl}
692
693 % to deprecate:
694
695 \parsemodule@allow{defi}
696 \parsemodule@allow{defii}
697 \parsemodule@allow{defiii}
698 \parsemodule@allow{defiv}
699 \parsemodule@allow{adefi}
700 \parsemodule@allow{adefii}
701 \parsemodule@allow{adefiii}
702 \parsemodule@allow{adefiv}
703 \parsemodule@allow{defis}
704 \parsemodule@allow{defiis}
705 \parsemodule@allow{defiiis}
706 \parsemodule@allow{defivs}
707 \parsemodule@allow{Defi}
708 \parsemodule@allow{Defii}
709 \parsemodule@allow{Defiii}

```

```

710 \parsemodule@allow{Defiv}
711 \parsemodule@allow{Defis}
712 \parsemodule@allow{Defiis}
713 \parsemodule@allow{Defiiis}
714 \parsemodule@allow{Defivis}

```

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*).

```

715 \catcode'\.=0
716 .catcode'\.=13
717 .def.\@active@slash{\}
718 .catcode'\.<=1
719 .catcode'\.>=2
720 .catcode'\{=12
721 .catcode'\}=12
722 .def.\@open@brace<{>
723 .def.\@close@brace<}>
724 .catcode'\.=0
725 \catcode'\.=12
726 \catcode'\{=1
727 \catcode'\}=2
728 \catcode'\<=12
729 \catcode'\>=12

```

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

`\set@parsemodule@catcodes`

```

730 \def\parsemodule@ignorepackageerrors{,inputenc,}
731 \let\parsemodule@old@PackageError\PackageError
732 \def\parsemodule@packageerror#1#2#3{%
733   \IfSubStr\parsemodule@ignorepackageerrors{,#1,}{\}%
734   \parsemodule@old@PackageError{#1}{#2}{#3}%
735   }%
736 }
737 \def\set@parsemodule@catcodes{%
738   \ifcat'\.=0%
739   \global\catcode'\.=13%
740   \global\catcode'\#=12%
741   \global\catcode'\{=12%
742   \global\catcode'\}=12%
743   \global\catcode'\$=12%$
744   \global\catcode'\^=12%
745   \global\catcode'\_ =12%
746   \global\catcode'\&=12%

```

```

747     \ex\global\ex\let\@active@slash\parsemodule@escapechar%
748     \global\let\parsemodule@old@PackageError\PackageError%
749     \global\let\PackageError\parsemodule@packageerror%
750     \fi%
751 }

```

`\reset@parsemodule@catcodes`

```

752 \def\reset@parsemodule@catcodes{%
753     \ifcat'\=13%
754     \global\catcode'\=0%
755     \global\catcode'\#=6%
756     \global\catcode'\{=1%
757     \global\catcode'\}=2%
758     \global\catcode'\$=3%$
759     \global\catcode'\~=7%
760     \global\catcode'\_ =8%
761     \global\catcode'\&=4%
762     \global\let\PackageError\parsemodule@old@PackageError%
763     \fi%
764 }

```

`\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.

```

765 \def\parsemodule@maybesetcodes{%
766     \if@smsmode\set@parsemodule@catcodes\fi%
767 }

```

`\parsemodule@escapechar`

This macro gets called whenever a `\`-character occurs in `sms` mode. It is split into several macros that parse and store characters in `\parsemodule@escape@currcls` until a character with category code  $\neq 11$  occurs (i.e. the macro name is complete), check whether the macro is allowed in `sms` mode, and then either ignore it or execute it after setting category codes back to normal. Special care needs to be taken to make sure that braces have the right category codes (1 and 2 for open and closing braces, respectively) when delimiting macro arguments.

Entry point:

```

768
769 \def\parsemodule@escapechar{%
770     \def\parsemodule@escape@currcls{}%
771     \parsemodule@escape@parse@nextchar%
772 }%

```

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`.



```

773 \long\def\parsemodule@escape@parse@nextchar@#1{%
774   \ifcat a#1\relax%
775     \edef\parsemodule@escape@currcls{\parsemodule@escape@currcls#1}%
776     \let\parsemodule@do@next\parsemodule@escape@parse@nextchar@%
777   \else%
778     \def\parsemodule@last@char{#1}%
779     \ifx\parsemodule@escape@currcls\@empty%
780       \def\parsemodule@do@next{}%
781     \else%
782       \def\parsemodule@do@next{\parsemodule@escapechar@checkcs}%
783     \fi%
784   \fi%
785   \parsemodule@do@next%
786 }

```

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.

```

787 \def\parsemodule@escapechar@checkcs{%
788   \ifx\parsemodule@escape@currcls\parsemodule@escapechar@beginstring%
789     \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@last@char}%
790   \else%
791     \ifx\parsemodule@escape@currcls\parsemodule@escapechar@endstring%
792       \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@last@char}%
793     \else%
794       \ifcvoid{parsemodule@allowedmacro@\parsemodule@escape@currcls}{%
795         \def\parsemodule@do@next{\relax\parsemodule@last@char}%
796       }{%
797         \ifx\parsemodule@last@char\@open@brace%
798           \ex\let\ex\parsemodule@do@next@ii\csname parsemodule@allowedmacro@\parsemodule@last@char\endcsname
799           \edef\parsemodule@do@next{\noexpand\parsemodule@converttoproperbraces\@open@brace}%
800         \else%
801           \reset@parsemodule@catcodes%
802           \edef\parsemodule@do@next{\ex\noexpand\csname parsemodule@allowedmacro@\parsemodule@last@char\endcsname}%
803         \fi%
804       }%
805     \fi%
806   \fi%
807   \parsemodule@do@next%
808 }

```

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.

```
809 \ex\ex\ex\def%
810 \ex\ex\ex\parsemodule@converttoproperbraces%
811 \ex\@open@brace\ex#\ex1\@close@brace{%
812   \reset@parsemodule@catcodes%
813   \parsemodule@do@next@ii{#1}%
814 }
```

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.

```
815 \ex\ex\ex\def%
816 \ex\ex\ex\parsemodule@escapechar@checkbeginenv%
817 \ex\@open@brace\ex#\ex1\@close@brace{%
818   \ifcvoid{parsemodule@allowedenv@#1}{%
819     \def\parsemodule@do@next{#1}%
820   }{%
821     \reset@parsemodule@catcodes%
822     \edef\parsemodule@envname{\csname parsemodule@allowedenv@#1\endcsname}%
823     \ex\def\ex\parsemodule@do@next\ex{%
824       \ex\begin\ex{\parsemodule@envname}%
825     }%
826   }%
827   \parsemodule@do@next%
828 }
829 \ex\ex\ex\def%
830 \ex\ex\ex\parsemodule@escapechar@checkendenv%
831 \ex\@open@brace\ex#\ex1\@close@brace{%
832   \ifcvoid{parsemodule@allowedenv@#1}{%
833     \def\parsemodule@do@next{#1}%
834   }{%
835     \edef\parsemodule@envname{\csname parsemodule@allowedenv@#1\endcsname}%
836     \ex\def\ex\parsemodule@do@next\ex{%
837       \ex\end\ex{\parsemodule@envname}%
838     }%
839   }%
840   \parsemodule@do@next%
841 }
```

`\@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.

```
842 \newrobustcmd\@requiremodules[1]{%
843   \if@tempswa\requiremodules{#1}\fi%
844 }
```

`\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.

```

845 \newrobustcmd\requiremodules[1]{%
846   \mod@showfalse%
847   \edef\mod@path{#1}%
848   \edef\mod@path{\expandafter\detokenize\expandafter{\mod@path}}%
849   \requiremodules@smsmode{#1}%
850 }%

```

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

```

851 \newbox\modules@import@tempbox
852 \def\requiremodules@smsmode#1{%
853   \setbox\modules@import@tempbox\vbox{%
854     \@smsmodetrue%
855     \set@parsemodule@catcodes%
856     \hbadness=100000\relax%
857     \hfuzz=10000pt\relax%
858     \vbadness=100000\relax%
859     \vfuzz=10000pt\relax%
860     \stexinput{#1.tex}%
861     \reset@parsemodule@catcodes%
862   }%
863   \parsemodule@maybesetcodes%
864 }

```

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

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

### 3.4.2 importmodule

`\importmodule@bookkeeping`

```

865 \newif\if@importmodule@switchrepos\@importmodule@switchreposfalse
866 \def\importmodule@bookkeeping#1#2#3{%
867   \@importmodule@switchreposfalse%
868   \stex@debug{Importmodule: #1^^J #2^^J\detokenize{#3}}%
869   \metasetkeys{importmodule}{#1}%
870   \ifcsvoid{importmodule@mhrepos}{%
871     \ifcsvoid{currentrepos@dir}{%
872       \stex@debug{Importmodule: Set importmodule@dir to \stex@PWD}%
873       \let\importmodule@dir\stex@PWD%
874     }{%
875       \stex@debug{Importmodule: Set importmodule@dir to \currentrepos@dir\@Slash source}%
876       \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
877     }%

```

```

878 }{%
879   \@importmodule@switchreposttrue%
880   \stex@debug{Importmodule: Repository switch to \@importmodule@mhrepos}%
881   \stex@debug{Importmodule: Current repos: \@mh@currentrepos}%
882   \ex\let\csname importmodule@oldrepos@#2\endcsname\mh@currentrepos%
883   \mathhub@setcurrentreposinfo\importmodule@mhrepos%
884   \stex@debug{Importmodule: New repos: \@mh@currentrepos^^J Namespace: \@currentrepos@ns}%
885   \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
886 }%
887 \StrCut{#2}\@QuestionMark\importmodule@subdir\importmodule@modulename%
888 \ifx\importmodule@modulename\empty%
889   \let\importmodule@modulename\importmodule@subdir%
890   \let\importmodule@subdir\empty%
891 \else%
892   \ifx\importmodule@subdir\empty\else%
893     \edef\importmodule@dir{\importmodule@dir\@Slash\importmodule@subdir}%
894   \fi%
895 \fi%
896 #3%
897 \if\importmodule@switchrepos%
898   \ex\mathhub@setcurrentreposinfo\csname importmodule@oldrepos@#2\endcsname%
899   \stex@debug{Importmodule: switched back to: \@mh@currentrepos}%
900 \fi%
901 %\ignorespacesandpars%
902 }

```

`\importmodule`

```

903 %\srefaddidkey{importmodule}
904 \addmetakey{importmodule}{mhrepos}
905 \newcommand\importmodule[2][\@importmodule[#1]{#2}{export}]
906 \newcommand\@importmodule[3][\@%
907   \importmodule@bookkeeping{#1}{#2}{%
908     \@importmodule[\importmodule@dir]\importmodule@modulename{#3}%
909   }%
910 }

```

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

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

```

911 \newcommand\@importmodule[3][\@%
912   {%
913     \edef\@load{#1}%
914     \edef\@importmodule@name{#2}%
915     \stex@debug{Loading #1}%

```

```

916 \if@smsmode\else\ifcsvoid{stex@module@\@importmodule@name}{% TODO check this
917 \stex@iffileexists\@load{
918 \stex@debug{Exists: #1}%
919 \requiremodules\@load}{%
920 \stex@debug{Does not exist: #1^^JTrying \@load\@Slash\@importmodule@name}%
921 \requiremodules{\@load\@Slash\@importmodule@name}%
922 }%
923 }{}\fi%
924 \ifx\@load\@empty\else%
925 {% TODO
926 % \edef\@path{\csname module@#2@path\endcsname}%
927 % \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do not
928 % {\PackageError{stex}% else signal an error
929 % {Module Name Clash\MessageBreak%
930 % A module with name #2 was already loaded under the path "\@path"\MessageBreak%
931 % The imported path "\@load" is probably a different module with the\MessageBreak%
932 % same name; this is dangerous -- not importing}%
933 % {Check whether the Module name is correct}%
934 % }%
935 }%
936 \fi%
937 \global\let\@importmodule@load\@load%
938 }%
939 \edef\@export{#3}\def\@export{export}%prepare comparison
940 %\ifx\@export\@export\export@defs{#2}\fi% export the module
941 \ifx\@export\@export\addto@thismodulex{%
942 \noexpand\@importmodule[\@importmodule@load]{#2}{noexport}%
943 }%
944 \if@smsmode\else
945 \ifcsvoid{this@module}{\fi%
946 \ifcsvoid{module@imports@\module@uri}{
947 \csxdef{module@imports@\module@uri}{%
948 \csname stex@module@#2\endcsname\@URI}% TODO check this
949 }%
950 }{\fi%
951 \csxdef{module@imports@\module@uri}{%
952 \csname stex@module@#2\endcsname\@URI,% TODO check this
953 \csname module@imports@\module@uri\endcsname%
954 }%
955 }%
956 }%
957 \fi\fi%
958 \if@smsmode\else%
959 \edef\activate@module@name{#2}%
960 \StrCount\activate@module@name\@Slash[\activate@module@lastslash]%
961 \ifnum\activate@module@lastslash>0%
962 \StrCut[\activate@module@lastslash]\activate@module@name\@Slash\activate@module@temp\activa
963 \fi%
964 \ifcsvoid{stex@lastmodule@\activate@module@name}{%
965 \PackageError{stex}{No module with name \activate@module@name found}{}%

```

```

966     }{%
967     \ex\ex\ex\activate@defs\ex\ex\ex{\csname stex@lastmodule@\activate@module@name\endcsname}%
968     }%
969     \fi% activate the module
970 }%

```

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

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

**Test 24:** `>\relax<`  
`>\@invoke@symbol {fomid:/core/foundations/types?type.en?type}<`

Default document module:

```

971 \AtBeginDocument{%
972   \set@default@ns%
973   \ifx\module@narr\@empty\setkeys{module}{narr=\module@ns}\fi%
974   \let\module@name\jobname%
975   \let\module@id\module@name % TODO deprecate
976   \edef\module@uri{\module@ns\@QuestionMark\module@name}%
977   \csgdef{module@names@\module@uri}{}%
978   \csgdef{module@imports@\module@uri}{}%
979   \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
980   \expandafter\global\expandafter\let\csname stex@module@\module@name\expandafter\endcsname\csname
981   \edef\this@module{%
982     \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
983   }%
984   \latexml@annotate@invisible{namespace}{\module@ns\@Slash\module@name}{}%
985   \csdef{module@defs@\module@uri}{}%
986   \ifcvoid{mh@currentrepos}{}%
987     \@inmhrepostrue%
988     \addto@thismodule{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\endcsname
989       {\noexpand\mh@currentrepos}}%
990     \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\mh@currentrepos}}%
991   }%
992 }

```

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

`\activate@defs` To activate the `\symdefs` from a given module  $\langle mod \rangle$ , we call the macro `\module@defs@ $\langle mod \rangle$` . But to make sure that every module is activated only once, we only activate if the macro `\module@defs@ $\langle mod \rangle$`  is undefined, and define it directly afterwards to prohibit further activations.

```

993 \newif\if@inimport\@inimportfalse
994 \def\latexml@import#1{\latexml@annotate@invisible{import}{#1}{}}%
995 \def\activate@defs#1{%
996   \stex@debug{Activating import #1}%
997   \if@inimport\else%
998     \latexml@import{#1}%
999     \def\inimport@module{#1}%
1000     \stex@debug{Entering import #1}%
1001     \@inimporttrue%
1002   \fi%
1003   \edef\activate@defs@uri{#1}%
1004   \ifcsundef{module@defs@\activate@defs@uri}{%
1005     \PackageError{stex}{No module with URI \activate@defs@uri loaded}{Probably missing an
1006       \detokenize{\importmodule} (or variant) somewhere?
1007   }
1008   }{%
1009     \ifcsundef{module@\activate@defs@uri @activated}{%
1010       {\csname module@defs@\activate@defs@uri\endcsname}{}}%
1011     \@namedef{module@\activate@defs@uri @activated}{true}%
1012   }%
1013   \def\inimport@thismodule{#1}%
1014   \stex@debug{End of import #1}%
1015   \ifx\inimport@thismodule\inimport@module\@inimportfalse%
1016     \stex@debug{Leaving import #1}%
1017   \fi%
1018 }%

```

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

```

1019 \newcommand\usemodule[2] [] {\@importmodule[#1]{#2}{noexport}}

```

**Test 26: Module 3.10[Foo]:      Module 3.11[Bar]:      »macro:->\@invoke@symbol  
{file:///home/jazzpirate/work/Software/ext/sTeX/sty/stex-master?Foo?foo}«  
Module 3.12[Baz]:      Should be undefined: »undefined«  
Should be defined: »macro:->\@invoke@symbol {file:///home/jazzpirate/work/Software/ext/sTeX**

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

```

1020 \def\inputref@preskip{}
1021 \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).

```

1022 \newrobustcmd\inputref[2] [] {%
1023   \importmodule@bookkeeping{#1}{#2}{%
1024     %\inputreftrue
1025     \inputref@preskip%
1026     \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
1027     \inputref@postskip%

```

```
1028 }%
1029 }%
```

**Test 27:**     **Module 3.13**[type.en]:     In type theory, a type is a primitive notion defined by inference rules on the judgment  $t : T$  (read:  $t$  has type  $T$ ) for a term  $t$  and a type  $T$ .

### 3.5 Symbols/Notations/Verbalizations

`\if@symdeflocal`   A flag whether a symbol declaration is local (i.e. does not get exported) or not.  
1030 `\newif\if@symdeflocal\@symdeflocalfalse`

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

```
1031 \def\define@in@module#1#2{
1032   \expandafter\edef\csname #1\endcsname{#2}%
1033   \edef\define@in@module@temp{%
1034     \def\expandafter\noexpand\csname#1\endcsname%
1035     {#2}%
1036   }%
1037   \if@symdeflocal\else%
1038     \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1039     \expandafter\endcsname\expandafter{\define@in@module@temp}%
1040   \fi%
1041 }
```

`\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.

```
1042 \addmetakey{symdecl}{name}%
1043 \addmetakey{symdecl}{type}%
1044 \addmetakey{symdecl}{args}%
1045 \addmetakey[false]{symdecl}{local}[true]%
1046
1047 \newcommand\symdecl[2][]{%
1048   \ifcsdef{this@module}{%
1049     \metasetkeys{symdecl}{#1}%
1050     \ifcsvoid{symdecl@name}{
1051       \edef\symdecl@name{#2}%
1052     }{}%
1053     \edef\symdecl@uri{\module@uri\@QuestionMark\symdecl@name}%
1054     \ifcsvoid{stex@symbol@\symdecl@name}{%
1055       \expandafter\edef\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}%
1056     }{}%
1057     \expandafter\def\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}%
1058   }%
1059   \edef\symdecl@symbolmacro{%
1060     \noexpand\ifcsvoid{stex@symbol@\symdecl@name}{%
1061       \expandafter\edef\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\symd
```



```

1062     }{%
1063     \expandafter\def\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\detokenize{
1064     }}%
1065 }%
1066 \ifcsvoid{\symdecl@type}{}%
1067 \setbox\modules@import@tempbox\hbox{$\symdecl@type$} % only to have latex check this
1068 }%
1069 \ifcsvoid{\symdecl@args}{\csgdef{\symdecl@uri\@QuestionMark args}{}}{%
1070 \IfInteger{\symdecl@args}{\notation@num@to@ia@\symdecl@args\csxdef{\symdecl@uri\@QuestionMark
1071 \ex\global\ex\let\csname\symdecl@uri\@QuestionMark args\endcsname\symdecl@args%
1072 }%
1073 }%
1074 \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1075 \expandafter\endcsname\expandafter{\symdecl@symbolmacro}%
1076 \ifcsvoid{\symdecl@uri}{%
1077 \ifcsvoid{\module@names@\module@uri}{%
1078 \csxdef{\module@names@\module@uri}{\symdecl@name}%
1079 }{%
1080 \csxdef{\module@names@\module@uri}{\symdecl@name,%
1081 \csname module@names@\module@uri\endcsname}%
1082 }%
1083 }{%
1084 % not compatible with circular dependencies, e.g. test/omdoc/07-modules/smsstest.tex
1085 \PackageWarning{stex}{symbol already defined: \symdecl@uri}{%
1086 You need to pick a fresh name for your symbol%
1087 }%
1088 }%
1089 \define@in@module\symdecl@uri{\noexpand\@invoke@symbol{\symdecl@uri}}%
1090 \IfStrEq{\symdecl@local}{false}{%
1091 \define@in@module{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
1092 }{%
1093 \csdef{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
1094 }%
1095 }{%
1096 \PackageError{stex}{\detokenize{\symdecl} not in a module}{You need to be in a module%
1097 in order to declare a new symbol}
1098 }%
1099 \if@inimport\else\if@inabbrdef\else\latexml@symdecl\symdecl@uri{$\symdecl@type$}{\csname\symd
1100 \if@insymdef\else\parsemodule@maybesetcodes\fi%
1101 }
1102 \def\latexml@symdecl#1#2#3#4#5{\latexml@annotate@invisible{\symdecl}{#1}{%
1103 \latexml@annotate{type}{#2}%
1104 \latexml@annotate{args}{#3}{%
1105 \latexml@annotate{definiens}{#4}%
1106 \latexml@annotate{macroname}{#2}{%
1107 }}

```

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

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

### 3.5.1 Notations

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

```

1108 % TODO make this work with actual macros (it doesn't)
1109 \edef\stex@ambiguous{\detokenize{ambiguous}}
1110 \edef\stex@macrostring{\detokenize{macro:->\@invoke@symbol}}
1111 \def\modules@getURIfromName#1{%
1112   \def\notation@uri{%
1113     \edef\modules@getURI@name{#1}%
1114     \ifcsvoid{\modules@getURI@name}{%
1115       \edef\modules@temp@meaning{%
1116         }{%
1117           \edef\modules@temp@meaning{\ex\meaning\csname\modules@getURI@name\endcsname}%
1118         }%
1119         \IfBeginWith\modules@temp@meaning\stex@macrostring{%
1120           % is a \@invoke@symbol macro
1121           \StrPosition\modules@temp@meaning\@close@brace[\stex@tempnum]%
1122           \StrMid\modules@temp@meaning{26}{\the\numexpr\stex@tempnum-1\@Space}[\notation@uri]%
1123         }{%
1124           % Check whether full URI or module?symbol or just name
1125           \StrCount\modules@getURI@name\@QuestionMark[\isuri@number]%
1126           \ifnum\isuri@number=2%
1127             \edef\notation@uri{\modules@getURI@name}%
1128           \else%
1129             \ifnum\isuri@number=1%
1130               % module?name
1131               \StrCut\modules@getURI@name\@QuestionMark\isuri@mod\isuri@name%
1132               \ifcsvoid{stex@module@\isuri@mod}{%
1133                 \PackageError{stex}{No module with name \isuri@mod\@Space loaded}{}%
1134               }{%
1135                 \expandafter\ifx\csname stex@module@\isuri@mod\endcsname\stex@ambiguous%
1136                 \PackageError{stex}{Module name \isuri@mod\@Space is ambiguous}{}%
1137               \else%
1138                 \edef\notation@uri{\csname stex@module@\isuri@mod\endcsname\@URI\@QuestionMark\isuri@mod}%
1139               \fi%
1140             }%
1141           \else%
1142             %name
1143             \ifcsvoid{stex@symbol@\modules@getURI@name}{%
1144               \PackageError{stex}{No symbol with name \modules@getURI@name\@Space known}{}%
1145             }{%
1146               \ifcsvoid{\module@uri\@QuestionMark\modules@getURI@name}{%
1147                 \expandafter\ifx\csname stex@symbol@\modules@getURI@name\endcsname\stex@ambiguous%
1148                 % Symbol name ambiguous and not in current module
1149                 \PackageError{stex}{Symbol name, URI or macroname \detokenize{#1} found!}{}%
1150               \else%
1151                 % Symbol not in current module, but unambiguous
1152                 \edef\notation@uri{\csname stex@symbol@\modules@getURI@name\endcsname}%

```

```

1153         \fi%
1154     }{% Symbol in current module
1155         \edef\notation@uri{\module@uri\@QuestionMark\modules@getURI@name}%
1156     }%
1157 }%
1158 \fi%
1159 \fi%
1160 }%
1161 }

\notation Adds a new notation to a symbol foo, as in: \notation[lang=en,arity=0,variant=op]{foo}{...}
\notation[variant=bar]{foo}{...} \notation[prec=500;50x49x51]{foo}{#1 bla #2 bla #3}{arg
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.

1162 \newif\if@innotation\@innotationfalse

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

1163 \def\notation@parse@params#1#2{%
1164     \def\notation@curr@prec{%
1165         \def\notation@curr@args{%
1166             \def\notation@curr@variant{%
1167                 \def\notation@curr@arityvar{%
1168                     \def\notation@curr@provided@arity{#2}
1169                     \def\notation@curr@lang{%
1170                         \def\notation@options@temp{#1}
1171                         \notation@parse@params%
1172                         \ifx\notation@curr@args\@empty%
1173                             \ifx\notation@curr@provided@arity\@empty%
1174                                 \notation@num@to@ia\notation@curr@arityvar%
1175                             \else%
1176                                 \notation@num@to@ia\notation@curr@provided@arity%
1177                             \fi%
1178                         \fi%
1179                         \StrLen\notation@curr@args[\notation@curr@arity]%
1180                     }
1181                     \def\notation@parse@params@{%
1182                         \IfSubStr\notation@options@temp,{%
1183                             \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
1184                             \notation@parse@param%
1185                             \notation@parse@params%
1186                         }{\ifx\notation@options@temp\@empty\else%
1187                             \let\notation@option@temp\notation@options@temp%
1188                             \notation@parse@param%
1189                         \fi}%
1190                 }
1191             }
1192         \def\notation@parse@param{%

```

```

1193 \path@trimstring\notation@option@temp%
1194 \ifx\notation@option@temp\@empty\else%
1195   \IfSubStr\notation@option@temp={%
1196     \StrCut\notation@option@temp=\notation@key\notation@value%
1197     \path@trimstring\notation@key%
1198     \path@trimstring\notation@value%
1199     \IfStrEq\notation@key{prec}{%
1200       \edef\notation@curr@prec{\notation@value}%
1201     }{%
1202       \IfStrEq\notation@key{args}{%
1203         \edef\notation@curr@args{\notation@value}%
1204       }{%
1205         \IfStrEq\notation@key{lang}{%
1206           \edef\notation@curr@lang{\notation@value}%
1207         }{%
1208           \IfStrEq\notation@key{variant}{%
1209             \edef\notation@curr@variant{\notation@value}%
1210           }{%
1211             \IfStrEq\notation@key{arity}{%
1212               \edef\notation@curr@arityvar{\notation@value}%
1213             }{%
1214               }}}}%
1215     }{%
1216       \edef\notation@curr@variant{\notation@option@temp}%
1217     }%
1218   \fi%
1219 }
1220
1221 % converts an integer to a string of 'i's, e.g. 3 => iii,
1222 % and stores the result in \notation@curr@args
1223 \def\notation@num@to@ia#1{%
1224   \IfInteger{#1}{
1225     \notation@num@to@ia@#1%
1226   }{%
1227     %
1228   }%
1229 }
1230 \def\notation@num@to@ia@#1{%
1231   \ifnum#1>0%
1232     \edef\notation@curr@args{\notation@curr@args i}%
1233     \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
1234   \fi%
1235 }
1236
1237 \newcount\notation@argument@counter
1238
1239 % parses the notation arguments and wraps them in
1240 % \notation@assoc and \notation@argprec for flexary arguments and precedences
1241 \providerobustcmd\notation[3][]{%
1242   \modules@getURIfromName{#2}%

```

```

1243 \notation@parse@params{#1}{}%
1244 \def\notation@temp@notation{}%
1245 \ex\let\ex\notation@curr@args\csname\notation@uri\@QuestionMark args\endcsname%
1246 \let\notation@curr@todo@args\notation@curr@args%
1247 \StrLen\notation@curr@todo@args[\notation@curr@arity]%
1248 \ex\renewcommand\ex\notation@temp@notation\ex[\notation@curr@arity]{#3}%
1249 % precedence
1250 \let\notation@curr@precstring\notation@curr@prec%
1251 \IfSubStr\notation@curr@prec;{%
1252   \StrCut\notation@curr@prec;\notation@curr@prec\notation@curr@prec%
1253   \ifx\notation@curr@prec\@empty\def\notation@curr@prec{0}\fi%
1254 }{%
1255   \ifx\notation@curr@prec\@empty%
1256     \ifnum\notation@curr@arity=0\relax%
1257       \edef\notation@curr@prec{\infprec}%
1258     \else%
1259       \def\notation@curr@prec{0}%
1260     \fi%
1261   \else%
1262     \edef\notation@curr@prec{\notation@curr@prec}%
1263     \def\notation@curr@prec{}%
1264   \fi%
1265 }%
1266 % arguments
1267 \notation@argument@counter=0%
1268 \def\notation@curr@extargs{}%
1269 \notation@do@args%
1270 }
1271
1272 \edef\notation@ichar{\detokenize{i}}%
1273 \edef\notation@achar{\detokenize{a}}%
1274 \edef\notation@bchar{\detokenize{b}}%
1275
1276 % parses additional notation components for (associative) arguments
1277 \def\notation@do@args{%
1278   \advance\notation@argument@counter by 1%
1279   \def\notation@nextarg@temp{}%
1280   \ifx\notation@curr@todo@args\@empty%
1281     \ex\notation@after%
1282   \else%
1283     % argument precedence
1284     \IfSubStr\notation@curr@prec{x}{%
1285       \StrCut\notation@curr@prec{x}\notation@curr@argprec\notation@curr@prec%
1286     }{%
1287       \edef\notation@curr@argprec{\notation@curr@prec}%
1288       \def\notation@curr@prec{}%
1289     }%
1290     \ifx\notation@curr@argprec\@empty%
1291       \let\notation@curr@argprec\notation@curr@prec%
1292     \fi%

```

```

1293 \StrChar\notation@curr@todo@args1[\notation@argchar]%
1294 \edef\notation@argchar{\ex\detokenize\ex{\notation@argchar}}%
1295 \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1296 \ifx\notation@argchar\notation@ichar%
1297 % normal argument
1298 \edef\notation@nextarg@temp{%
1299 {\stex@arg{\the\notation@argument@counter}{\notation@curr@argprec}{#####\the\notation
1300 }%
1301 \ex\g@addto@macro@safe\ex\notation@curr@extargs%
1302 \ex{\notation@nextarg@temp}%
1303 \ex\ex\ex\notation@do@args%
1304 \else\ifx\notation@argchar\notation@bchar%
1305 % bound argument
1306 \edef\notation@nextarg@temp{%
1307 {\stex@arg{\the\notation@argument@counter}{\notation@curr@argprec}{#####\the\notation
1308 }%
1309 \ex\g@addto@macro@safe\ex\notation@curr@extargs%
1310 \ex{\notation@nextarg@temp}%
1311 \ex\ex\ex\ex\ex\ex\ex\notation@do@args%
1312 \else%
1313 % associative argument
1314 \ex\ex\ex\ex\ex\ex\ex\notation@parse@assocarg%
1315 \fi%
1316 \fi%
1317 \fi%
1318 }
1319
1320 \def\notation@parse@assocarg#1{%
1321 \def\notation@parse@assocop{#1}%
1322 \edef\notation@nextarg@temp{%
1323 {\stex@arg{\the\notation@argument@counter}{\notation@curr@argprec}{\notation@assoc{\ex\unexp
1324 {#####\the\notation@argument@counter}}}%
1325 }%
1326 \ex\g@addto@macro@safe\ex\notation@curr@extargs\ex{\notation@nextarg@temp}%
1327 \notation@do@args%
1328 }
1329
1330 \protected\def\safe@newcommand#1{%
1331 \ifdefined#1\ex\renewcommand\else\ex\newcommand\fi#1%
1332 }
1333
1334 % finally creates the actual macros
1335 \def\notation@after{
1336 % \notation@curr@prec
1337 % \notation@curr@args
1338 % \notation@curr@variant
1339 % \notation@curr@arity
1340 % \notation@curr@provided@arity
1341 % \notation@curr@lang
1342 % \notation@uri

```

```

1343 \def\notation@temp@fragment{}%
1344 \ifx\notation@curr@arityvar\@empty\else%
1345   \edef\notation@temp@fragment{arity=\notation@curr@arityvar}%
1346 \fi%
1347 \ifx\notation@curr@lang\@empty\else%
1348   \ifx\notation@temp@fragment\@empty%
1349     \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1350   \else%
1351     \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1352   \fi%
1353 \fi%
1354 \ifx\notation@curr@variant\@empty\else%
1355   \ifx\notation@temp@fragment\@empty%
1356     \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1357   \else%
1358     \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1359   \fi%
1360 \fi%
1361 \ex\ex\ex\def\ex\ex\ex\notation@temp@notation\ex\ex\ex%
1362   {\ex\notation@temp@notation\notation@curr@extargs}%
1363 \ifnum\notation@curr@arity=0%
1364   \edef\notation@temp@notation{\stex@dooms{\notation@uri}{\notation@temp@fragment}{\notation@
1365 \else%
1366   \IfSubStr\notation@curr@args\notation@bchar{%
1367     \edef\notation@temp@notation{\stex@doomb{\notation@uri}{\notation@temp@fragment}{\notation
1368   }{%
1369     \edef\notation@temp@notation{\stex@dooma{\notation@uri}{\notation@temp@fragment}{\notation
1370   }%
1371 \fi%
1372 \stex@debug{Notation \notation@uri: \meaning\notation@temp@notation}%
1373 \notation@final%
1374 \parsemodule@maybesetcodes%
1375 }
1376
1377 \def\notation@final{%
1378   \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1379   \stex@debug{Defining \notation@csname of arity \notation@curr@arity}%
1380   \ifcvoid{\notation@csname}{%
1381     \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1382     \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@curr@arity\ex]%
1383     \ex{\notation@temp@notation}%
1384     \edef\symdecl@temps{%
1385       \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@curr@ari
1386     }%
1387     \ex@g@addto@macro@safe\csname module@defs@module@uri\ex\endcsname\ex{\symdecl@temps}%
1388     \ex@g@addto@macro@safe\csname module@defs@module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1389   }{%
1390     \PackageWarning{stex}{notation already defined: \notation@csname}{%
1391       Choose a different set of notation options (variant,lang,arity)%
1392     }%

```

```

1393 }%
1394 \@innotationfalse%
1395 \if@inimport\else\if@latexml%
1396   \let\notation@simarg@args\notation@curr@args%
1397   \notation@argument@counter=0%
1398   \def\notation@simargs{}%
1399   \notation@simulate@arguments%
1400   \latexml\notation\notation@uri\notation@temp@fragment\notation@curr@args\notation@curr@prec%
1401   {$\csname\notation@csname\ex\endcsname\notation@simargs$}%
1402   \fi\fi%
1403 }
1404 \def\notation@simulate@arguments{%
1405   \ifx\notation@simarg@args\@empty\else%
1406     \advance\notation@argument@counter by 1%
1407     \IfBeginWith\notation@simarg@args{i}{%
1408       \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\the\notation@argument}}}%
1409     }{%
1410       \IfBeginWith\notation@simarg@args{b}{%
1411         \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\the\notation@argument}}}%
1412       }{%
1413         \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\@Fragment\the\notation@argument}}}%
1414       }%
1415     }%
1416     \StrGobbleLeft\notation@simarg@args1[\notation@simarg@args]%
1417     \notation@simulate@arguments%
1418   \fi%
1419 }
1420 % URI, fragment, arity, notation
1421 \def\latexml@notation#1#2#3#4#5{\latexml@annotate@invisible{notation}{#1}{%
1422   \latexml@annotate{notationfragment}{#2}{}%
1423   \latexml@annotate{args}{#3}{}%
1424   \latexml@annotate{precedence}{#4}{}%
1425   \latexml@annotate{notationcomp}{#5}%
1426 }}

```

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

```

1427 \protected\def\notation@assoc#1#2{% function, argv
1428   \let\@tmpop=\relax% do not print the function the first time round
1429   \@for\@I:=#2\do{\@tmpop% print the function
1430     % write the i-th argument with locally updated precedence
1431     \@I%
1432   \def\@tmpop{#1}%
1433   }%
1434 }%
1435
1436 \def\notation@lparen{()
1437 \def\notation@rparen{)}
1438 \def\infprec{1000000}
1439 \def\neginfprec{-\infprec}

```



```

1440
1441 \newcount\notation@downprec
1442 \notation@downprec=\neginfprec
1443
1444 % patching displaymode
1445 \newif\if@displaymode\@displaymodefalse
1446 \ex\everydisplay\ex{\the\everydisplay\@displaymodetrue}
1447 \let\old@displaystyle\displaystyle
1448 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1449
1450 \protected\def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1451   \def\notation@innertmp{#1}%
1452   \if@displaymode%
1453     \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1454     \ex\notation@resetbrackets\ex\notation@innertmp%
1455     \ex\right\notation@rparen%
1456   \else%
1457     \ex\ex\ex\notation@lparen%
1458     \ex\notation@resetbrackets\ex\notation@innertmp%
1459     \notation@rparen%
1460   \fi%
1461 }
1462
1463 \protected\def\withbrackets#1#2#3{%
1464   \edef\notation@lparen{#1}%
1465   \edef\notation@rparen{#2}%
1466   #3%
1467   \notation@resetbrackets%
1468 }
1469
1470 \protected\def\notation@resetbrackets{%
1471   \def\notation@lparen{()%
1472   \def\notation@rparen{)%}
1473 }
1474
1475 \protected\def\stex@dooms#1#2#3#4{%
1476   \if@innotation%
1477     \notation@symprec{#3}{#4}%
1478   \else%
1479     \@innotationtrue%
1480     \latexml@oms{#1}{#2}{\notation@symprec{#3}{#4}}%
1481     \@innotationfalse%
1482   \fi%
1483 }
1484
1485 \protected\def\stex@doomb#1#2#3#4{%
1486   \if@innotation%
1487     \notation@symprec{#3}{#4}%
1488   \else%
1489     \@innotationtrue%

```

```

1490 \latexml@ombind{#1}{#2}{\notation@symprec{#3}{#4}}%
1491 \@innotationfalse%
1492 \fi%
1493 }
1494
1495 \protected\def\stex@dooma#1#2#3#4{%
1496 \if@innotation%
1497 \notation@symprec{#3}{#4}%
1498 \else%
1499 \@innotationtrue%
1500 \latexml@oma{#1}{#2}{\notation@symprec{#3}{#4}}%
1501 \@innotationfalse%
1502 \fi%
1503 }
1504
1505 % for LaTeXML Bindings
1506 \protected\def\latexml@oms#1#2#3{%
1507 \latexml@annotate{OMID}{#1\@Fragment#2}{#3}%
1508 }
1509
1510 \protected\def\latexml@oma#1#2#3{%
1511 \edef\latexml@oma@uri{%
1512 \ifcsname#1\@QuestionMark args\endcsname%
1513 #1\@Fragment\csname#1\@QuestionMark args\endcsname\@Fragment#2%
1514 \else#1\@Fragment\@Fragment#2\fi%
1515 }%
1516 \latexml@annotate{OMA}{\latexml@oma@uri}{#3}%
1517 }
1518
1519 \protected\def\latexml@ombind#1#2#3{%
1520 \edef\latexml@oma@uri{%
1521 \ifcsname#1\@QuestionMark args\endcsname%
1522 #1\@Fragment\csname#1\@QuestionMark args\endcsname\@Fragment#2%
1523 \else#1\@Fragment\@Fragment#2\fi%
1524 }%
1525 \latexml@annotate{OMBIND}{\latexml@oma@uri}{#3}%
1526 }
1527
1528 \def\notation@symprec#1#2{%
1529 \ifnum#1>\notation@downprec\relax%
1530 \notation@resetbrackets#2%
1531 \else%
1532 \ifnum\notation@downprec=\infprec\relax%
1533 \notation@resetbrackets#2%
1534 \else
1535 \if@inarray@
1536 \notation@resetbrackets#2
1537 \else\dobrackets{#2}\fi%
1538 \fi\fi%
1539 }

```

```

1540
1541 \newif\if@inpparray@\@inpparray@false
1542
1543
1544 \protected\def\stex@arg#1#2#3{%
1545   \@innotationfalse%
1546   \latexml@arg{#1}{\notation@argprec{#2}{#3}}%
1547   \@innotationtrue%
1548 }
1549
1550 % for LaTeXML Bindings
1551 \def\latexml@arg#1#2{%
1552   \latexml@annotate{arg}{#1}{#2}%
1553 }
1554
1555 \def\notation@argprec#1#2{%
1556   \def\notation@innertmp{#2}
1557   \edef\notation@downprec@temp{\number#1}%
1558   \notation@downprec=\ex\notation@downprec@temp%
1559   \ex\relax\ex\notation@innertmp%
1560   \ex\notation@downprec\ex=\number\notation@downprec\relax%
1561 }

```

Macros for introducing OMSs and OMAs manually

```

1562 \protected\def\stex@oms#1#2{\modules@getURIfromName{#1}\latexml@oms{\notation@uri}{#2}}
1563 \protected\def\stex@oma#1#2{\modules@getURIfromName{#1}\latexml@oma{\notation@uri}{#2}}
1564 \protected\def\stex@ombind#1#2{\modules@getURIfromName{#1}\latexml@ombind{\notation@uri}{#2}}

```

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

```

1565 \protected\def\@invoke@symbol#1{%
1566   \ifmmode%
1567     \def\@invoke@symbol@first{#1}%
1568     \let\invoke@symbol@next\invoke@symbol@math%
1569   \else%
1570     \def\invoke@symbol@next{\invoke@symbol@text{#1}}%
1571   \fi%
1572   \invoke@symbol@next%
1573 }

```

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

```

1574 \newcommand\invoke@symbol@math[1][]{%
1575   \notation@parse@params{#1}{}%
1576   \def\notation@temp@fragment{}%
1577   \ifx\notation@curr@arityvar\@empty\else%
1578     \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1579   \fi%
1580   \ifx\notation@curr@lang\@empty\else%
1581     \ifx\notation@temp@fragment\@empty%

```



`\[ \times \{\frac \vara \varb , \plus \{\frac \vara \{\frac \vara \varb \}, \times \{\varc , \plus \{\vard , \vare , 2\}\}\} \}`:

$$\frac{a}{b} \cdot \left( \frac{a}{b} + c \cdot (d + e + 2) \right)$$

`\abbrdef` The `\abbrdef` macro is a variant of `\symdecl` that does the same on the L<sup>A</sup>T<sub>E</sub>X level, and adds a definiens on the OMDoc level.

```

1606 \newif\if@inabbrdef\@inabbrdeffalse
1607 \def\abbrdef@definiens{}
1608 \newcommand\abbrdef[3][\%
1609   \@inabbrdeftrue\symdecl[#1]{#2}%
1610   \@inabbrdeffalse%
1611   \ex\let\ex\abbrdef@args\csname\symdecl@uri\@QuestionMark args\endcsname%
1612   \StrLen\abbrdef@args[\abbrdef@arity]
1613   \ex\renewcommand\ex\abbrdef@definiens\ex[\abbrdef@arity]{\unexpanded{#3}}%
1614   \if@inimport\else\if@latexml%
1615     \let\notation@simarg@args\abbrdef@args%
1616     \notation@argument@counter=0%
1617     \def\notation@simargs{}%
1618     \notation@simulate@arguments%
1619     \latexml\symdecl\symdecl@uri{\$ \symdecl@type$}{\csname\symdecl@uri\@QuestionMark args\endcsname
1620       {\$ \ex\abbrdef@definiens\notation@simargs$}{#2}%
1621   \fi\fi%
1622 }
```

**Test 30:** `\symdecl {foo}`  
`\notation {foo}{\psi }`

`$_{foo}$`  
`\abbrdef {lftype}{\stex@oms {http://cds.omdoc.org/urtheories?Typed?type}{}}`  
`\notation {lftype}{\noexpand \mathtt {type}}`  
`$_{lftype}$`  
*type*

### 3.6 Verbalizations

```

1623 \newif\if@inoms
1624 \def\invoke@symbol@text#1{%
1625   \edef\invoke@symbol@uri{#1}%
1626   \def\invoke@symbol@return{}%
1627   \notation@argument@counter=0%
1628   \edef\invoke@symbol@arity{\csname #1\@QuestionMark args\endcsname}%
1629   \ifx\invoke@symbol@arity\empty\@inomsfalse\fi%
1630   \invoke@symbol@text@args%
1631 }
1632
```

```

1633 \edef\notation@Xchar{\detokenize{X}}%
1634
1635 \protected\def\opref#1{%
1636   \modules@getURIfromName{#1}%
1637   \let\invoke@symbol@uri\notation@uri%
1638   \def\invoke@symbol@return{}%
1639   \notation@argument@counter=0%
1640   \def\invoke@symbol@arity{}%
1641   \@inometrue%
1642   \invoke@symbol@text@args%
1643 }
1644
1645 \def\invoke@symbol@text@args{%
1646   \advance\notation@argument@counter by 1%
1647   \edef\notation@charnum{\the\notation@argument@counter}%
1648   \StrChar\invoke@symbol@arity{\the\notation@argument@counter}[\invoke@symbol@nextchar]%
1649   \ifx\invoke@symbol@nextchar\notation@Xchar%
1650     \ex\invoke@symbol@text@args%
1651   \else%
1652     \ifx\invoke@symbol@nextchar\@empty%
1653       \let\invoke@symbol@nextstep\invoke@symbol@text@finally%
1654       \ex\ex\ex\invoke@symbol@maybesqbracket%
1655     \else%
1656       \let\invoke@symbol@nextstep\invoke@symbol@normalarg%
1657       \ex\ex\ex\invoke@symbol@maybestarI%
1658     \fi%
1659   \fi%
1660 }
1661
1662 \def\invoke@symbol@maybestarI{%
1663   \@ifnextchar*{%
1664     \@ifnextchar[{%
1665       \invoke@symbol@switchnum%
1666     }{%
1667       \invoke@symbol@invisible%
1668     }%
1669   }{%
1670     \invoke@symbol@maybesqbracket%
1671   }%
1672 }
1673
1674 \def\invoke@symbol@maybesqbracket{%
1675   \@ifnextchar[{\invoke@symbol@verbcomp}{\invoke@symbol@nextstep}%
1676 }
1677
1678 \def\invoke@symbol@verbcomp[#1]{%
1679   \ex\def\ex\invoke@symbol@return\ex{\invoke@symbol@return #1}%
1680   \invoke@symbol@nextstep%
1681 }
1682

```

```

1683 \def\invoke@symbol@invisible*#1{% TODO a-args
1684   \edef\invoke@symbol@frame{\noexpand\latexml@annotate@invisible{arg}{\notation@charnum}}}%
1685   \ex\ex\ex\def\ex\ex\ex\invoke@symbol@return\ex\ex\ex{\ex\invoke@symbol@return\invoke@symbol@f
1686   \invoke@symbol@text@args%
1687 }
1688
1689 \def\invoke@symbol@normalarg#1{% TODO a-args
1690   \edef\invoke@symbol@frame{\noexpand\latexml@annotate{arg}{\notation@charnum}}}%
1691   \ex\ex\ex\def\ex\ex\ex\invoke@symbol@return\ex\ex\ex{\ex\invoke@symbol@return\invoke@symbol@f
1692   \invoke@symbol@text@args%
1693 }
1694
1695 \def\invoke@symbol@switchnum*[#1]{%
1696   \advance\notation@argument@counter by -1%
1697   \edef\notation@charnum[#1]%
1698   \StrChar\invoke@symbol@arity\notation@charnum[\invoke@symbol@nextchar]%
1699   \ifx\invoke@symbol@nextchar\notation@ichar%
1700     \StrLeft\invoke@symbol@arity{\numexpr\notation@charnum-1}[\invoke@symbol@newarityLeft]%
1701     \StrGobbleLeft\invoke@symbol@arity\notation@charnum[\invoke@symbol@newarity]%
1702     \edef\invoke@symbol@newarity{\invoke@symbol@newarityLeft\notation@Xchar\invoke@symbol@newar
1703   \else% TODO
1704   \fi%
1705   \invoke@symbol@maybestarII%
1706 }
1707
1708 \def\invoke@symbol@maybestarII{%
1709   \@ifnextchar*{%
1710     \invoke@symbol@invisible%
1711   }{%
1712     \invoke@symbol@normalarg%
1713   }%
1714 }
1715
1716 \def\invoke@symbol@text@finally{%
1717   \stex@debug{HERE! \meaning\invoke@symbol@return}%
1718   \if@inoms\latexml@oms{\invoke@symbol@uri}{\invoke@symbol@return}%
1719   \else\latexml@oma{\invoke@symbol@uri}{\invoke@symbol@return}%
1720   \fi%
1721 }

```

**Test 31: Module 3.16[FooBarVerbs]:** `\symdecl [args=ii]{plus}`  
`\symdecl {someprime}`  
`\plus [The sum of ]{\someprime [$p$]}[ and ]{$2$}`: “The sum of  $p$  and 2”  
plus and + and +.

### 3.7 Term References

```

\ifhref
1722 \newif\ifhref\hreffalse%
1723 \AtBeginDocument{%

```

```

1724 \@ifpackageloaded{hyperref}{%
1725   \hreftrue%
1726 }{%
1727   \hreffalse%
1728 }%
1729 }

\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.
1730 \newbox\stex@targetbox
1731 \def\termref@maketarget#1#2{%
1732   % #1: symbol URI
1733   % #2: text
1734   \stex@debug{Here: #1 <> #2}%
1735   \ifhref\if@smsmode\else%
1736     \hypertarget{sref@#1@target}{#2}%
1737   \fi\fi%
1738   \stex@debug{Here!}%
1739   \expandafter\edef\csname sref@#1\endcsname##1{%
1740     \ifhref\if@smsmode\else\noexpand\hyperlink{sref@#1@target}{##1}\fi\fi%
1741   }%
1742 }

\@termref
1743 \def\@termref#1#2{%
1744   % #1: symbol URI
1745   % #2: text
1746   \ifcvoid{#1}{%
1747     \StrCut[2]{#1}\@QuestionMark\termref@mod\termref@name%
1748     \ifcvoid{\termref@mod}{%
1749       \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
1750     }{%
1751       \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
1752         contains no symbol with name \termref@name.%
1753     }{}%
1754   }%
1755 }{%
1756   \ifcvoid{sref@#1}{%
1757     #2% TODO: No reference point exists!
1758   }{%
1759     \csname sref@#1\endcsname{#2}%
1760   }%
1761 }%
1762 }

\tref
1763
1764 \def\@capitalize#1{\uppercase{#1}}%
1765 \newrobustcmd\capitalize[1]{\expandafter\@capitalize #1}%

```



```

1766
1767 \newcommand\tref[2] []{%
1768   \edef\tref@name{#1}%
1769   \expandafter\modules@getURIfromName\expandafter{\tref@name}%
1770   \expandafter\@termref\expandafter{\notation@uri}{#2}%
1771 }
1772 \def\trefs#1{%
1773   \modules@getURIfromName{#1}%
1774   % TODO
1775 }
1776 \def\Tref#1{%
1777   \modules@getURIfromName{#1}%
1778   % TODO
1779 }
1780 \def\Trefs#1{%
1781   \modules@getURIfromName{#1}%
1782   % TODO
1783 }

\defi
1784 \addmetakey{defi}{name}
1785 \def\@definiendum#1#2{%
1786   \parsemodule@maybesetcodes%
1787   \stex@debug{Here: #1 | #2}%
1788   \termref@maketarget{#1}{#2}%\termref@maketarget{#1}{\defemph{#2}}%
1789 }
1790
1791 \newcommand\defi[2] []{%
1792   \metasetkeys{defi}{#1}%
1793   \ifx\defi@name\@empty%
1794     \symdecl@constructname{#2}%
1795     \let\defi@name\symdecl@name%
1796     \let\defi@verbalization\symdecl@verbalization%
1797   \else%
1798     \edef\defi@verbalization{#2}%
1799   \fi%
1800   \ifcsvoid{\module@uri\@QuestionMark\defi@name}{%
1801     \symdecl\defi@name%
1802   }{\edef\symdecl@uri{\module@uri\@QuestionMark\defi@name}}%
1803   \@definiendum\symdecl@uri\defi@verbalization%
1804 }
1805 \def\Defi#1{%
1806   \symdecl{#1}%
1807   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization}%
1808 }
1809 \def\defis#1{%
1810   \symdecl{#1}%
1811   \@definiendum\symdecl@uri{\symdecl@verbalization s}%
1812 }
1813 \def\Defis#1{%

```

```

1814 \symdecl{#1}%
1815 \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization s}%
1816 }

```

### 3.8 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
1817 \newif\ifhref\hreffalse%
1818 \AtBeginDocument{%
1819   \ifpackageloaded{hyperref}{%
1820     \hreftrue%
1821   }{%
1822     \hreffalse%
1823   }%
1824 }%
1825 \newcommand\sref@href@ifh[2]{%
1826   \ifhref%
1827     \href{#1}{#2}%
1828   \else%
1829     #2%
1830   \fi%
1831 }%
1832 \newcommand\sref@hlink@ifh[2]{%
1833   \ifhref%
1834     \hyperlink{#1}{#2}%
1835   \else%
1836     #2%
1837   \fi%
1838 }%
1839 \newcommand\sref@target@ifh[2]{%
1840   \ifhref%
1841     \hypertarget{#1}{#2}%
1842   \else%
1843     #2%
1844   \fi%
1845 }%

```

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

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

```

1846 \def\sref@target{%
1847   \ifx\sref@id\@empty%
1848     \relax%
1849   \else%

```

```

1850 \edef\@target{sref@ifcsundef{sref@part}}{\sref@part @}\sref@id @target}%
1851 \sref@target@ifh\@target{}%
1852 \fi%
1853 }%

\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.
1854 \addmetakey{srefaddidkey}{prefix}
1855 \newcommand\srefaddidkey[2][]{%
1856 \metasetkeys{srefaddidkey}{#1}%
1857 \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
1858 \metakeys@ext@clear@keys{#2}{id}{}%
1859 \metakeys@ext@showkeys{#2}{id}%
1860 \define@key{#2}{id}{%
1861 \edef\sref@id{\srefaddidkey@prefix ##1}%
1862 %\expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
1863 \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1864 }%
1865 }%

\@sref@def This macro stores the value of its last argument in a custom macro for reference.
1866 \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.
1867 \ifextrefs%
1868 \newwrite\refs@file%
1869 \else%
1870 \def\refs@file{\@auxout}%
1871 \fi%

\sref@def This macro writes an \@sref@def command to the current aux file and also exe-
cutes it.
1872 \newcommand\sref@def[3]{%
1873 \protected@write\refs@file{}{\string\@sref@def{#1}{#2}{#3}}%
1874 }%

\sref@label The \sref@label macro writes a label definition to the auxfile.
1875 \newcommand\sref@label[2]{%
1876 \sref@def{\ifcsundef{sref@part}}{\sref@part @}{#2}{page}{\thepage}%
1877 \sref@def{\ifcsundef{sref@part}}{\sref@part @}{#2}{label}{#1}%
1878 }%

\sreflabel The \sreflabel macro is a semantic version of \label, it combines the catego-
rization given in the first argument with LATEX's \@currentlabel.
1879 \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.

```
1880 \def\sref@id{} % make sure that defined
1881 \newcommand\sref@label@id[1]{%
1882   \ifx\sref@id\@empty%
1883     \relax%
1884   \else%
1885     \sref@label{#1}{\sref@id}%
1886   \fi%
1887 }%
```

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

```
1888 \newcommand\sref@label@id@arg[2]{%
1889   \def\@id{#2}
1890   \ifx\@id\@empty%
1891     \relax%
1892   \else%
1893     \sref@label{#1}{\@id}%
1894   \fi%
1895 }%
```

### 3.9 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`.

```
1896 \newenvironment{modsig}[2][]{\def\@test{#1}%
1897 \ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1898 \expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
1899 %\ignorespacesandpars
1900 }
1901 {\end{module}}\ignorespacesandpars
1902 }
```

### 3.10 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`.

```

1903 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1904 \newrobustcmd\@gimport@star[2][\def\@test{#1}%
1905 \edef\mh@crepos{\mh@currentrepos}%
1906 \ifx\@test\@empty%
1907 \importmhmodule[conservative,mhrepos=\mh@crepos,path=#2]{#2}%
1908 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1909 \mathhub@setcurrentreposinfo{\mh@crepos}%
1910 %\ignorespacesandpars
1911 \parsemodule@maybesetcodes}
1912 \newrobustcmd\@gimport@nostar[2][\def\@test{#1}%
1913 \edef\mh@crepos{\mh@currentrepos}%
1914 \ifx\@test\@empty%
1915 \importmhmodule[mhrepos=\mh@crepos,path=#2]{#2}%
1916 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1917 \mathhub@setcurrentreposinfo{\mh@crepos}%
1918 %\ignorespacesandpars
1919 \parsemodule@maybesetcodes}

```

### 3.11 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.

```

1920 \def\modules@@first#1/#2;{#1}
1921 \newcommand\libinput[1]{%
1922 \stex@debug{Libinput current repo: \meaning\mh@currentrepos}%
1923 \ifcsvoid{mh@currentrepos}{%
1924 \PackageError{stex}{current MathHub repository not found}{}}%
1925 {}
1926 \edef\@mh@group{\expandafter\modules@@first\mh@currentrepos;}
1927 \let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
1928 \def\mh@inffile{\MathHub{\@mh@group/meta-inf/lib/#1}}
1929 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1930 \IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
1931 \IfFileExists\mh@inffile{\IfFileExists\mh@libfile}{%
1932 {\PackageError{stex}
1933 {Library file missing; cannot input #1.tex\MessageBreak%
1934 Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
1935 do not exist}%
1936 {Check whether the file name is correct}}}}
1937 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1938 \let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}

```

### 3.12 omdoc/omgroup

```

1939 \newcount\section@level
1940
1941 \section@level=2
1942 \ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{}
1943 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{}
1944 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{}
1945 \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>.

1946 \newcommand\omgroup@nonum[2]{%
1947 \ifx\hyper@anchor\@undefined\else\phantomsection\fi%
1948 \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 omgrou environment and – if it is use it. But how to do that depends on whether the rdfmeta package has been loaded. In the end we call \sref@label@id to enable crossreferencing.

1949 \newcommand\omgroup@num[2]{%
1950 \edef\@@ID{\sref@id}
1951 \ifx\omgroup@short\@empty% no short title
1952 \@nameuse{#1}{#2}%
1953 \else% we have a short title
1954 \@ifundefined{rdfmeta@sectioning}%
1955   {\@nameuse{#1}[\omgroup@short]{#2}}%
1956   {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
1957 \fi%
1958 \sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@@ID}

omgroup
1959 \def\@true{true}
1960 \def\@false{false}
1961 \srefaddidkey{omgroup}
1962 \addmetakey{omgroup}{date}
1963 \addmetakey{omgroup}{creators}
1964 \addmetakey{omgroup}{contributors}
1965 \addmetakey{omgroup}{srccite}
1966 \addmetakey{omgroup}{type}
1967 \addmetakey*{omgroup}{short}
1968 \addmetakey*{omgroup}{display}
1969 \addmetakey[false]{omgroup}{loadmodules}[true]

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

1970 \newif\if@mainmatter\@mainmattertrue
1971 \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.

```

1972 \addmetakey{omdoc@sect}{name}
1973 \addmetakey[false]{omdoc@sect}{clear}[true]
1974 \addmetakey{omdoc@sect}{ref}
1975 \addmetakey[false]{omdoc@sect}{num}[true]
1976 \newcommand\omdoc@sectioning[3][\metasetkeys{omdoc@sect}{#1}%
1977 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
1978 \if@mainmatter% numbering not overridden by frontmatter, etc.
1979 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
1980 \def\current@section@level{\omdoc@sect@name}%
1981 \else\omgroup@nonum{#2}{#3}%
1982 \fi}% if@mainmatter

```

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

```

1983 \newcommand\omgroup@redefine@addtocontents[1]{%
1984 %\edef\@import{#1}%
1985 %\@for\@I:=\@import\do{%
1986 %\edef\@path{\csname module@\@I @path\endcsname}%
1987 %\@ifundefined{tf@toc}\relax%
1988 % {\protected@write\tf@toc}{\string\@requiremodules{\@path}}}%
1989 %\ifx\hyper@anchor\@undefined% hyperref.sty loaded?
1990 %\def\addcontentsline##1##2##3{%
1991 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}%
1992 %\else% hyperref.sty not loaded
1993 %\def\addcontentsline##1##2##3{%
1994 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}{\@c
1995 %\fi
1996 }% hypreref.sty loaded?

```

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

```

1997 \newcount\omgroup@level
1998 \newenvironment{omgroup}[2][\keys, title
1999 {\metasetkeys{omgroup}{#1}\sref@target%
2000 \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.

```

2001 \ifx\omgroup@loadmodules\@true%
2002 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
2003 {\@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`.

```

2004 \advance\section@level by 1\relax%

```

```

2005 \ifcase\section@level%
2006 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
2007 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
2008 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
2009 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
2010 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%
2011 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{#2}%
2012 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{#2}%
2013 \fi% \ifcase
2014 \at@begin@omgroup[#1]\section@level{#2}}% for customization
2015 {\advance\section@level by -1\advance\omgroup@level by -1}

```

and finally, we localize the sections

```

2016 \newcommand\omdoc@part@kw{Part}
2017 \newcommand\omdoc@chapter@kw{Chapter}
2018 \newcommand\omdoc@section@kw{Section}
2019 \newcommand\omdoc@subsection@kw{Subsection}
2020 \newcommand\omdoc@subsubsection@kw{Subsubsection}
2021 \newcommand\omdoc@paragraph@kw{paragraph}
2022 \newcommand\omdoc@subparagraph@kw{subparagraph}

```

`\setSGvar` set a global variable

```

2023 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}

```

`\useSGvar` use a global variable

```

2024 \newrobustcmd\useSGvar[1]{%
2025   \@ifundefined{sTeX@Gvar@#1}
2026   {\PackageError{omdoc}
2027     {The sTeX Global variable #1 is undefined}
2028     {set it with \protect\setSGvar}}
2029 \@nameuse{sTeX@Gvar@#1}}

```

`blindomgroup`

```

2030 \newcommand\at@begin@blindomgroup[1]{%
2031 \newenvironment{blindomgroup}
2032 {\advance\section@level by 1\at@begin@blindomgroup\setion@level}
2033 {\advance\section@level by -1}

```

### 3.13 omtext

#### 3.13.1 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).

```

2034 \srefaddidkey{omtext}
2035 \addmetakey[] {omtext} {functions}
2036 \addmetakey* {omtext} {display}

```



```

2037 \addmetakey{omtext}{for}
2038 \addmetakey{omtext}{from}
2039 \addmetakey{omtext}{type}
2040 \addmetakey*{omtext}{title}
2041 \addmetakey*{omtext}{start}
2042 \addmetakey{omtext}{theory}
2043 \addmetakey{omtext}{continues}
2044 \addmetakey{omtext}{verbalizes}
2045 \addmetakey{omtext}{subject}

\st@flow We define this macro, so that we can test whether the display key has the value
flow
2046 \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.

2047 \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.

2048 \def\omtext@pre@skip{\smallskip}
2049 \def\omtext@post@skip{}
2050 \newenvironment{omtext}[1][\@in@omtexttrue%
2051 \bgroup\metasetkeys{omtext}{#1}\sref@label{id}{this paragraph}%
2052 \def\lec##1{\@lec{##1}}%
2053 \omtext@pre@skip\par\noindent%
2054 \ifx\omtext@title\@empty%
2055 \ifx\omtext@start\@empty\else%
2056 \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
2057 \fi% end omtext@start empty
2058 \else\stDMemph{\omtext@title}:\enspace%
2059 \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
2060 \fi% end omtext@title empty
2061 %\ignorespacesandpars
2062 }
2063 {\egroup\omtext@post@skip\@in@omtextfalse%\ignorespacesandpars
2064 }
```

### 3.13.2 Phrase-level Markup

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

```

2065 \srefaddidkey{phrase}
2066 \addmetakey{phrase}{style}
2067 \addmetakey{phrase}{class}
2068 \addmetakey{phrase}{index}
2069 \addmetakey{phrase}{verbalizes}
2070 \addmetakey{phrase}{type}
2071 \addmetakey{phrase}{only}
```

```

2072 \newcommand\phrase[2] [] {\metasetkeys{phrase}{#1}%
2073 \ifx\prhase@only\@empty\only<\phrase@only>{#2}\else #2\fi}

\coref*
2074 \providecommand\textsubscript[1]{\ensuremath{_{\text{#1}}}}
2075 \newcommand\corefs[2]{#1\textsubscript{#2}}
2076 \newcommand\coreft[2]{#1\textsuperscript{#2}}

\n*lex
2077 \newcommand\nlex[1]{\green{\sl{#1}}}
2078 \newcommand\nlcex[1]{*\green{\sl{#1}}}

sinlinequote
2079 \def\@sinlinequote#1{‘‘{\sl{#1}}’’}
2080 \def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
2081 \newcommand\sinlinequote[2] []
2082 {\def\@opt{#1}\ifx\@opt\@empty\@sinlinequote{#2}\else\@@sinlinequote\@opt{#2}\fi}

```

### 3.13.3 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.

```

2083 \newcommand\vdec[2] [] {#2}
2084 \newcommand\vrest[2] [] {#2}
2085 \newcommand\vcond[2] [] {#2}

```

EdN:1            \strucdec    1

```

2086 \newcommand\strucdec[2] [] {#2}

```

EdN:2            \impdec    2

```

2087 \newcommand\impdec[2] [] {#2}

```

### 3.13.4 Block-Level Markup

```

sblockquote
2088 \def\begin@sblockquote{\begin{quote}\sl}
2089 \def\end@sblockquote{\end{quote}}
2090 \def\begin@@sblockquote#1{\begin@sblockquote}
2091 \def\end@@sblockquote#1{\def\@lec##1{\textrm{##1}}\@lec{#1}\end@sblockquote}
2092 \newenvironment{sblockquote}[1] []
2093 {\def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
2094 {\ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}

```

---

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

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

sboxquote

```
2095 \newenvironment{sboxquote}[1] []
2096 {\def\@@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
2097 {\@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.

```
2098 \providecommand{\@@lec}[1]{( #1 )}
2099 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@@lec{#1}}
2100 \def\lec#1{\@lec{#1}\par}
```

### 3.13.5 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.

```
2101 \addmetakey{omdoc@index}{at}
2102 \addmetakey[false]{omdoc@index}{loadmodules}[true]
2103 \newcommand\omdoc@indexi[2] [] {\ifindex%
2104 \metasetkeys{omdoc@index}{#1}%
2105 \@bsphack\begingroup\@sanitize%
2106 \protected@write\@indexfile{\string\indexentry%
2107 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
2108 \ifx\omdoc@index@loadmodules\@true%
2109 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
2110 \else #2\fi% loadmodules
2111 }\thepage}}%
2112 \endgroup\@esphack\fi}%ifindex
2113 \newcommand\omdoc@indexii[3] [] {\ifindex%
2114 \metasetkeys{omdoc@index}{#1}%
2115 \@bsphack\begingroup\@sanitize%
2116 \protected@write\@indexfile{\string\indexentry%
2117 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
2118 \ifx\omdoc@index@loadmodules\@true%
2119 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
2120 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
2121 \else #2!#3\fi% loadmodules
2122 }\thepage}}%
2123 \endgroup\@esphack\fi}%ifindex
2124 \newcommand\omdoc@indexiii[4] [] {\ifindex%
2125 \metasetkeys{omdoc@index}{#1}%
2126 \@bsphack\begingroup\@sanitize%
2127 \protected@write\@indexfile{\string\indexentry%
2128 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
2129 \ifx\omdoc@index@loadmodules\@true%
2130 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
2131 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
2132 \else #2!#3\fi% loadmodules
2133 }\thepage}}%
2134 \endgroup\@esphack\fi}%ifindex
```

```

2126 \@bsphack\beginngroup\@sanitize%
2127 \protected@write\@indexfile{}\string\indexentry%
2128 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
2129 \ifx\omdoc@index@loadmodules\@true%
2130 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
2131 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
2132 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
2133 \else #2!#3!#4\fi% loadmodules
2134 }\the page}}%
2135 \endgroup\@esphack\fi}%ifindex
2136 \newcommand\omdoc@indexiv[5][]{\ifindex%
2137 \metasetkeys{omdoc@index}{#1}%
2138 \@bsphack\beginngroup\@sanitize%
2139 \protected@write\@indexfile{}\string\indexentry%
2140 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
2141 \ifx\omdoc@index@loadmodules\@true%
2142 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
2143 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
2144 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
2145 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
2146 \else #2!#3!#4!#5\fi% loadmodules
2147 }\the page}}%
2148 \endgroup\@esphack\fi}%ifindex

```

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

\\*indi\*

```

2149 \newcommand\aindi[3][]{\{#2\}\omdoc@indexi[#1]{#3}}
2150 \newcommand\indi[2][]{\{#2\}\omdoc@indexi[#1]{#2}}
2151 \newcommand\indis[2][]{\{#2\}\omdoc@indexi[#1]{#2s}}
2152 \newcommand\Indi[2][]{\{\captitalize{#2}\}\omdoc@indexi[#1]{#2}}
2153 \newcommand\Indis[2][]{\{\capitalize{#2}\}\omdoc@indexi[#1]{#2s}}
2154
2155 \newcommand\@indii[3][]{\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
2156 \newcommand\aindii[4][]{\{#2\}\@indii[#1]{#3}{#4}}
2157 \newcommand\indii[3][]{\{#2 #3\}\@indii[#1]{#2}{#3}}
2158 \newcommand\indiis[3][]{\{#2 #3s\}\@indii[#1]{#2}{#3}}
2159 \newcommand\Indii[3][]{\{\captitalize{#2 #3}\}\@indii[#1]{#2}{#3}}
2160 \newcommand\Indiis[3][]{\{\capitalize{#2 #3}\}\@indii[#1]{#2}{#3}}
2161
2162 \newcommand\@indiii[4][]{\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexiii[#1]{#3}{#2}{#4}}
2163 \newcommand\aindiii[5][]{\{#2\}\@indiii[#1]{#3}{#4}{#5}}
2164 \newcommand\indiii[4][]{\{#2 #3 #4\}\@indiii[#1]{#2}{#3}{#4}}
2165 \newcommand\indiis[4][]{\{#2 #3 #4s\}\@indiii[#1]{#2}{#3}{#4}}
2166 \newcommand\Indiii[4][]{\{\captitalize{#2 #3 #4}\}\@indiii[#1]{#2}{#3}{#4}}
2167 \newcommand\Indiis[4][]{\{\capitalize{#2 #3 #4s}\}\@indiii[#1]{#2}{#3}{#4}}
2168
2169 \newcommand\@indiv[5][]{\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
2170 \newcommand\aindiv[6][]{\{#2\}\@indiv[#1]{#3}{#4}{#5}{#6}}
2171 \newcommand\indiv[5][]{\{#2 #3 #4 #5\}\@indiv[#1]{#2}{#3}{#4}{#5}}

```

```

2172 \newcommand\indivs[5] []{{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
2173 \newcommand\Indiv[5] []{\capitalizе{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
2174 \newcommand\Indivs[5] []{\capitalizе{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}

```

### 3.13.6 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.

```

2175 \newcommand\hateq{\ensuremath{\widehat{=}}\xspace}
2176 \newcommand\hatequiv{\ensuremath{\widehat{equiv}}\xspace}
2177 \@ifundefined{ergo}%
2178 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
2179 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
2180 \newcommand{\reflect@squig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%
2181 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
2182 \newcommand\notergo{\ensuremath{\not\leadsto}}
2183 \newcommand\notogre{\ensuremath{\not\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%

```

### 3.13.7 Deprecated Functionality

In this section we centralize old interfaces that are only partially supported any more.

\\*def\*

```

2184 \newcommand\indextoo[2] []{\indi[#1]{#2}}%
2185 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}%
2186 \newcommand\indexalt[2] []{\aindi[#1]{#2}}%
2187 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead}%
2188 \newcommand\twintoo[3] []{\indii[#1]{#2}{#3}}%
2189 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}%
2190 \newcommand\twinalt[3] []{\aindii[#1]{#2}{#3}}%
2191 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead}%
2192 \newcommand\atwintoo[4] []{\indiii[#1]{#2}{#3}{#4}}%
2193 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead}%
2194 \newcommand\atwinalt[4] []{\aindiii[#1]{#2}{#3}{#4}}%
2195 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instead}%

```

\my\*graphics

```

2196 \newcommand\mygraphics[2] []{\includegraphics[#1]{#2}}%
2197 \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics}%
2198 \newcommand\mycgraphics[2] []{\begin{center}\mygraphics[#1]{#2}\end{center}}%
2199 \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics}%
2200 \newcommand\mybgraphics[2] []{\fbox{\mygraphics[#1]{#2}}}%
2201 \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics}%
2202 \newcommand\mycbgraphics[2] []{\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}%
2203 \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphics}%

```

## 4 Things to deprecate

Module options:

```

2204 \addmetakey*{module}{id} % TODO: deprecate properly
2205 \addmetakey*{module}{load}
2206 \addmetakey*{module}{path}
2207 \addmetakey*{module}{dir}
2208 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
2209 \addmetakey*{module}{noalign}[true]
2210
2211 \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.

```

2212 %\srefaddidkey{symdef}% what does this do?
2213 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
2214 \define@key{symdef}{noverb}[all]{}%
2215 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
2216 \define@key{symdef}{specializes}{}%
2217 \addmetakey*{symdef}{noalign}[true]
2218 \define@key{symdef}{primary}[true]{}%
2219 \define@key{symdef}{assocarg}{}%
2220 \define@key{symdef}{bvars}{}%
2221 \define@key{symdef}{bargs}{}%
2222 \addmetakey{symdef}{lang}%
2223 \addmetakey{symdef}{prec}%
2224 \addmetakey{symdef}{arity}%
2225 \addmetakey{symdef}{variant}%
2226 \addmetakey{symdef}{ns}%
2227 \addmetakey{symdef}{args}%
2228 \addmetakey{symdef}{name}%
2229 \addmetakey*{symdef}{title}%
2230 \addmetakey*{symdef}{description}%
2231 \addmetakey{symdef}{subject}%
2232 \addmetakey*{symdef}{display}%
2233 \addmetakey*{symdef}{gfc}%

```

EdN:3

3

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

```

2234 \def\symdef{\@ifnextchar[{\@symdef}{\@symdef []}}%
2235 \def\@symdef[#1]#2{\@ifnextchar[{\@@symdef[#1]{#2}}{\@@symdef[#1]{#2}[0]}}%

```

---

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

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

```

2236 \def\@@symdef[#1]#2[#3]{%
2237   \@insymdef@true%
2238   \metasetkeys{symdef}{#1}%
2239   \edef\symdef@tmp@optpars{\ifcsvoid{symdef@name}{[]}{[name=\symdef@name]}}%
2240   \expandafter\symdecl\symdef@tmp@optpars{#2}%
2241   \@insymdef@false%
2242   \notation[#1]{#2}[#3]%
2243 }% mod@show
2244 \def\symdef@type{Symbol}%
2245 \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`.

```

2246 \def\symvariant#1{%
2247   \@ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}%
2248 }%
2249 \def\@symvariant#1[#2]#3#4{%
2250   \notation[#3]{#1}[#2]{#4}%
2251 }% \ignorespacesandpars
2252 }%

```

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

```

2253 \newif\if@importing\@importingfalse
2254 \define@key{symi}{noverb}[all]{}%
2255 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
2256 \define@key{symi}{specializes}{}%
2257 \define@key{symi}{gfc}{}%
2258 \define@key{symi}{noalign}[true]{}%
2259 \newcommand\symi{\@ifstar\@symi@star\@symi}
2260 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
2261   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces
2262 }
2263 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
2264   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignorespaces
2265 }
2266 \newcommand\symii{\@ifstar\@symii@star\@symii}
2267 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
2268   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespaces
2269 }
2270 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
2271   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\ignorespaces
2272 }
2273 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
2274 \newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%

```

```

2275 \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi%\ignorespacesandpars%
2276 }
2277 \newcommand\@symiii@star[4][\metasetkeys{symi}{#1}%
2278 \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi%\ignorespacesandpars%
2279 }
2280 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
2281 \newcommand\@symiv[5][\metasetkeys{symi}{#1}%
2282 \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi%\ignorespacesandpars%
2283 }
2284 \newcommand\@symiv@star[5][\metasetkeys{symi}{#1}%
2285 \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4-#5}\fi%\ignorespacesandpars%
2286 }

```

`\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`.

```

2287 %\srefaddidkey{importmhmodule}%
2288 \addmetakey{importmhmodule}{mhrepos}%
2289 \addmetakey{importmhmodule}{path}%
2290 \addmetakey{importmhmodule}{ext}% why does this exist?
2291 \addmetakey{importmhmodule}{dir}%
2292 \addmetakey[false]{importmhmodule}{conservative}[true]%
2293 \newcommand\importmhmodule[2][\%
2294 \parsemodule@maybesetcodes
2295 \metasetkeys{importmhmodule}{#1}%
2296 \ifx\importmhmodule@dir\@empty%
2297 \edef\@path{\importmhmodule@path}%
2298 \else\edef\@path{\importmhmodule@dir/#2}\fi%
2299 \ifx\@path\@empty% if module name is not set
2300 \importmodule[#2]{export}%
2301 \else%
2302 \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
2303 \ifx\importmhmodule@mhrepos\@empty% if in the same repos
2304 \relax% no need to change mh@currentrepos, i.e., current directory.
2305 \else%
2306 \mathhub@setcurrentreposinfo\importmhmodule@mhrepos% change it.
2307 \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}}%
2308 \fi%
2309 \@importmodule[\MathHub{\mh@currentrepos/source/\@path}]{#2}{export}%
2310 \mathhub@setcurrentreposinfo\mh@@repos% after importing, reset to old value
2311 \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\mh@@repos}}%
2312 \fi%
2313 %\ignorespacesandpars%
2314 }

```

`\usemhmodule`



```

2315 \addmetakey{importmhmodule}{load}
2316 \addmetakey{importmhmodule}{id}
2317 \addmetakey{importmhmodule}{dir}
2318 \addmetakey{importmhmodule}{mhrepos}
2319
2320 \addmetakey{importmodule}{load}
2321 \addmetakey{importmodule}{id}
2322
2323 \newcommand\usemhmodule[2][]{%
2324 \metasetkeys{importmhmodule}{#1}%
2325 \ifx\importmhmodule@dir\empty%
2326 \edef\@path{\importmhmodule@path}%
2327 \else\edef\@path{\importmhmodule@dir/#2}\fi%
2328 \ifx\@path\empty%
2329 \usemodule[id=\importmhmodule@id]{#2}%
2330 \else%
2331 \edef\mh@@repos{\mh@currentrepos}%
2332 \ifx\importmhmodule@mhrepos\empty%
2333 \else\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
2334 \usemodule{\@path\@QuestionMark#2}%
2335 \usemodule[load=\MathHub{\mh@currentrepos/source/\@path},
2336 % id=\importmhmodule@id]{#2}%
2337 \mathhub@setcurrentreposinfo\mh@@repos%
2338 \fi%
2339 %\ignorespacesandpars
2340 }

\mhinputref
2341 \newcommand\mhinputref[2][]{%
2342 \edef\mhinputref@first{#1}%
2343 \ifx\mhinputref@first\empty%
2344 \inputref{#2}%
2345 \else%
2346 \inputref[mhrepos=\mhinputref@first]{#2}%
2347 \fi%
2348 }

\trefi*
2349 \newcommand\trefi[2][]{%
2350 \edef\trefi@mod{#1}%
2351 \ifx\trefi@mod\empty\tref{#2}\else\tref{#1\@QuestionMark#2}\fi%
2352 }
2353 \newcommand\trefii[3][]{%
2354 \edef\trefi@mod{#1}%
2355 \ifx\trefi@mod\empty\tref{#2-#3}\else\tref{#1\@QuestionMark#2-#3}\fi%
2356 }

\defi*
2357 \def\defii#1#2{\defi{#1!#2}}

```

```

2358 \def\Defii#1#2{\Defi{#1!#2}}
2359 \def\defiis#1#2{\defis{#1!#2}}
2360 \def\Defiis#1#2{\Defis{#1!#2}}
2361 \def\defiii#1#2#3{\defi{#1!#2!#3}}
2362 \def\Defiii#1#2#3{\Defi{#1!#2!#3}}
2363 \def\defiis#1#2#3{\defis{#1!#2!#3}}
2364 \def\Defiis#1#2#3{\Defis{#1!#2!#3}}
2365 \def\defiv#1#2#3#4{\defi{#1!#2!#3!#4}}
2366 \def\Defiv#1#2#3#4{\Defi{#1!#2!#3!#4}}
2367 \def\defivs#1#2#3#4{\defis{#1!#2!#3!#4}}
2368 \def\Defivs#1#2#3#4{\Defis{#1!#2!#3!#4}}
2369 \def\adefi#1#2{\defi[name=#2]{#1}}
2370 \def\adefi#1#2#3{\defi[name=#2-#3]{#1}}
2371 \def\adefiii#1#2#3#4{\defi[name=#2-#3-#4]{#1}}
2372 \def\adefiv#1#2#3#4#5{\defi[name=#2-#3-#4-#5]{#1}}

```