

stex.sty: \TeX 2.0*

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

March 24, 2021

Abstract

TODO

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

Contents

1	Introduction	3
2	User commands	3
3	Implementation	3
3.1	sTeX base	4
3.2	Paths and URIs	4
3.3	Modules	16
3.4	Inheritance	22
3.5	Symbols/Notations/Verbalizations	32
3.6	Verbalizations	46
3.7	Term References	49
3.8	sref	51
3.9	smultiling	53
3.10	smglom	53
3.11	mathhub	54
3.12	omdoc/omgroup	55
3.13	omtext	57
4	Things to deprecate	63

1 Introduction

TODO

2 User commands

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

3 Implementation

```
1 <*cls>
2 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{stex}}
3 \ProcessOptions
4 \LoadClass[border=1px,varwidth]{standalone}
5 \setlength\textwidth{15cm}
6 \g@addto@macro{\@parboxrestore}{\setlength\parskip{\baselineskip}}
7 \usepackage{stex}
8 </cls>
9 <*package>
10 \RequirePackage{amsfonts}
11
12 \let\ex\expandafter
13 % TODO
14 \newif\ifstex@debugmode\stex@debugmodefalse
15 \DeclareOption{debug}{\stex@debugmodetrue}
16 \def\stex@debug#1{\ifstex@debugmode\message{^^J#1^^J}\fi}
17 % Modules:
18 \newif\ifmod@show\mod@showfalse
19 \DeclareOption{showmods}{\mod@showtrue}
20 % sref:
```

```

21 \newif\ifextrefs\extrefsfalse
22 \DeclareOption{extrefs}{\extrefstrue}
23 %
24 \ProcessOptions
25
26 \ifstex@debugmode\stex@debug{sTeX debug mode on}\fi

```

A conditional for LaTeXML:

```

27 \ifcename if@latexml\endcename\else
28 \ex\newif\cename if@latexml\endcename\@latexmlfalse
29 \fi

```

The following macro and environment generate LaTeXML annotations as a `` 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.

```

30 \protected\long\def\latexml@annotate#1#2#3{%
31   \def\latexml@annotate@bodyarg{#3}%
32   \if@latexml\ifmmode\latexml@annotate@math{#1}{#2}{\ifx\latexml@annotate@bodyarg\@empty\ \else
33 }
34 \protected\long\def\latexml@annotate@text#1#2#3{}
35 \protected\long\def\latexml@annotate@math#1#2#3{}
36 \newenvironment{latexml@annotateenv}[2]{\{}}{\}
37 \protected\long\def\latexml@annotate@invisible#1#2#3{}
38 \RequirePackage{xspace}
39 \RequirePackage{standalone}
40 \RequirePackageWithOptions{stex-metakeys}
41 \if@latexml\else\RequirePackage{xstring}\fi
42 \RequirePackage{etoolbox}

```

3.1 sTeX base

The sTeX logo:

```

43 \protected\def\stex{%
44   \ifundefined{texorpdfstring}%
45   {\let\texorpdfstring\@firstoftwo}%
46   }%
47   \texorpdfstring{\raisebox{-.5ex}{S}\kern-.5ex\TeX}{sTeX}\xspace%
48 }
49 \def\sTeX{\stex}

```

3.2 Paths and URIs

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

```

50 \def\pathsuris@setcatcodes{%
51   \edef\pathsuris@oldcatcode@hash{\the\catcode'\#}%
52   \catcode'\#=12\relax%
53   \edef\pathsuris@oldcatcode@slash{\the\catcode'\}%
54   \catcode'\/=12\relax%

```

```

55 \edef\pathsuris@oldcatcode@colon{\the\catcode'\:%}
56 \catcode'\:=12\relax%
57 \edef\pathsuris@oldcatcode@qm{\the\catcode'\?}%
58 \catcode'\?=12\relax%
59 }
60 \def\pathsuris@resetcatcodes{%
61 \catcode'\#\pathsuris@oldcatcode@hash\relax%
62 \catcode'\/\pathsuris@oldcatcode@slash\relax%
63 \catcode'\:\pathsuris@oldcatcode@colon\relax%
64 \catcode'\?\pathsuris@oldcatcode@qm\relax%
65 }

```

`\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`.

```

66 \def\namespace@read#1{%
67 \edef\namespace@read@path{#1}%
68 \edef\namespace@read@path{\ex\detokenize\ex\namespace@read@path}}%
69 \namespace@continue%
70 }
71 \def\namespace@continue{%
72 \pathsuris@resetcatcodes%
73 \ex\edef\csname\namespace@macroname\endcsname##1{%
74 \namespace@read@path\@Slash##1%
75 }%
76 }
77 \protected\def\namespace#1{%
78 \def\namespace@macroname{#1}%
79 \pathsuris@setcatcodes%
80 \namespace@read%
81 }
82 \let\defpath\namespace

```

3.2.1 Path Canonicalization

We define some macros for later comparison.

```

83 \pathsuris@setcatcodes
84 \def\@ToTop{..}
85 \def\@Slash{/}
86 \def\@Colon{:}
87 \def\@Space{ }
88 \def\@QuestionMark{?}
89 \def\@Dot{.}

```

```

90 \catcode'\&=12
91 \def\@Ampersand{&}
92 \catcode'\&=4
93 \def\@Fragment{#}
94 \pathsuris@resetcatcodes
95 \catcode'\.=0
96 .catcode'\.=12
97 .let.\@BackSlash\
98 .catcode'\.=0
99 \catcode'\.=12
100 \edef\old@percent@catcode{\the\catcode'\%}
101 \catcode'\%=12
102 \let\@Percent%
103 \catcode'\%=\old@percent@catcode

```

\@cpath Canonicalizes (file) paths:

```

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

```

```

138     \ifx\pathsuris@cpath@temp@a\@Dot%
139         \@cpath@loop%
140     \else%
141     \IfBeginWith\pathsuris@cpath@temp\@ToTop{%
142         \StrBehind{\pathsuris@cpath@temp}{\@ToTop}%
143         [\pathsuris@cpath@temp]%
144         \IfBeginWith\pathsuris@cpath@temp\@Slash{%
145             \edef\pathsuris@cpath@temp%
146                 {\@cpath@path\pathsuris@cpath@temp}%
147             }{%
148                 \ifx\@cpath@path\@empty\else%
149                     \edef\pathsuris@cpath@temp%
150                         {\@cpath@path\@Slash\pathsuris@cpath@temp}%
151                 \fi%
152             }%
153             \def\@cpath@path{}%
154             \@cpath@loop%
155         }{%
156             \ifx\@cpath@path\@empty%
157                 \edef\@cpath@path{\pathsuris@cpath@temp@a}%
158             \else%
159                 \edef\@cpath@path%
160                     {\@cpath@path\@Slash\pathsuris@cpath@temp@a}%
161             \fi%
162             \@cpath@loop%
163         }%
164     \fi\fi%
165 }{%
166     \ifx\@cpath@path\@empty%
167         \edef\@cpath@path{\pathsuris@cpath@temp}%
168     \else%
169         \edef\@cpath@path{\@cpath@path\@Slash\pathsuris@cpath@temp}%
170     \fi%
171 }%
172 }

```

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.

```

173 \newcommand\cpath@print[1]{%
174   \cpath{#1}%
175   \cpath@path%
176 }
```

`\path@filename`

```

177 \def\path@filename#1#2{%
178   \edef\filename@oldpath{#1}%
179   \StrCount\filename@oldpath\@Slash[\filename@lastslash]%
180   \ifnum\filename@lastslash>0%
181     \StrBehind[\filename@lastslash]\filename@oldpath%
182     \@Slash[\filename@oldpath]%
183     \edef#2{\filename@oldpath}%
184   \else%
185     \edef#2{\filename@oldpath}%
186   \fi%
187 }
```

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

`\path@filename@noext`

```

188 \def\path@filename@noext#1#2{%
189   \path@filename{#1}{#2}%
190   \edef\filename@oldpath{#2}%
191   \StrCount\filename@oldpath\@Dot[\filename@lastdot]%
192   \ifnum\filename@lastdot>0%
193     \StrBefore[\filename@lastdot]\filename@oldpath%
194     \@Dot[\filename@oldpath]%
195     \edef#2{\filename@oldpath}%
196   \else%
197     \edef#2{\filename@oldpath}%

```



```

198     \fi%
199 }

```

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:

```

202 \newif\if@iswindows@\@iswindows@false
201 \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:

```

202 \newif\if@windowstopath@inpath@
203 \def\windows@to@path#1{%
204     \@windowstopath@inpath@false%
205     \def\windows@temp{}%
206     \edef\windows@path{#1}%
207     \ifx\windows@path\empty\else%
208         \ex\windows@path@loop\windows@path\windows@path@end%
209     \fi%
210     \let#1\windows@temp%
211 }
212 \def\windows@path@loop#1#2\windows@path@end{%
213     \def\windows@temp@b{#2}%
214     \ifx\windows@temp@b\empty%
215         \def\windows@continue{}%
216     \else%
217         \def\windows@continue{\windows@path@loop#2\windows@path@end}%
218     \fi%
219     \if@windowstopath@inpath@%
220         \ifx#1\@BackSlash%
221             \edef\windows@temp{\windows@temp\@Slash}%
222         \else%
223             \edef\windows@temp{\windows@temp#1}%
224         \fi%
225     \else%
226         \ifx#1:%
227             \edef\windows@temp{\@Slash\windows@temp}%
228             \@windowstopath@inpath@true%
229         \else%
230             \edef\windows@temp{\windows@temp#1}%
231         \fi%
232     \fi%
233     \windows@continue%
234 }

```

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:

```
235 \def\path@to@windows#1{%
236     \@windowstopath@inpath@false%
237     \def\windows@temp{%
238         \edef\windows@path{#1}%
239         \edef\windows@path{\expandafter\@gobble\windows@path}%
240         \ifx\windows@path\empty\else%
241             \expandafter\path@windows@loop\windows@path\windows@path@end%
242         \fi%
243         \let#1\windows@temp%
244     }
245 \def\path@windows@loop#1#2\windows@path@end{%
246     \def\windows@temp@b{#2}%
247     \ifx\windows@temp@b\empty%
248         \def\windows@continue{%
249             \else%
250                 \def\windows@continue{\path@windows@loop#2\windows@path@end}%
251             \fi%
252             \if@windowstopath@inpath@%
253                 \ifx#1/%
254                     \edef\windows@temp{\windows@temp\@BackSlash}%
255                 \else%
256                     \edef\windows@temp{\windows@temp#1}%
257                 \fi%
258             \else%
259                 \ifx#1/%
260                     \edef\windows@temp{\windows@temp:\@BackSlash}%
261                     \@windowstopath@inpath@true%
262                 \else%
263                     \edef\windows@temp{\windows@temp#1}%
264                 \fi%
265             \fi%
266             \windows@continue%
267     }
```

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:

```
268 \def\path@trimstring#1{%
269     \edef\pathsuris@trim@temp{#1}%
270     \IfBeginWith\pathsuris@trim@temp\@Space{%
271         \StrGobbleLeft\pathsuris@trim@temp1[#1]%
272         \path@trimstring{#1}%
273     }
```

```

273     }{%
274         \IfEndWith\pathsuris@trim@temp\@Space{%
275             \StrGobbleRight\pathsuris@trim@temp1[#1]%
276             \path@trimstring{#1}%
277         }{%
278             \edef#1{\pathsuris@trim@temp}%
279         }%
280     }%
281 }

```

Test 7: `>foo bar<`

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

```

282 %\if@latexml\else
283 \def\@kpsewhich#1#2{\begingroup%
284     \edef\kpsewhich@cmd{"|kpsewhich #2"}%
285     \everyeof{\noexpand}%
286     \catcode'\=12%
287     \edef#1{\@input\kpsewhich@cmd\@Space}%
288     \path@trimstring#1%
289     \if@iswindows@\windows@to@path#1\fi%
290     \xdef#1{\ex\detokenize\expandafter{#1}}%
291 \endgroup}
292 %\fi

```

Test 8: `/usr/share/texlive/texmf-dist/tex/latex/etoolbox/etoolbox.sty`

3.2.4 \TeX input hooks

We determine the PWD of the current main document:

```

293 \edef\pwd@cmd{\if@iswindows@ -expand-var \@Percent%
294     CD\@Percent\else -var-value PWD\fi}
295 \@kpsewhich\stex@PWD\pwd@cmd
296 \edef\stex@mainfile{\stex@PWD\@Slash\jobname}
297 \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:

```

298 \def\stex@currfile@stack{}
299
300 \def\stex@currfile@push#1{%
301     \edef\stex@temppath{#1}%
302     \edef\stex@temppath{\ex\detokenize\ex{\stex@temppath}}%
303     \edef\stex@currfile@stack{\stex@currfile%
304         \ifx\stex@currfile@stack\@empty\else,\stex@currfile@stack\fi}
305     \IfBeginWith\stex@temppath\@Slash{\@cpath{\stex@temppath}}{%
306         \@cpath{\stex@PWD\@Slash#1}%
307     }

```

```

308 \let\stex@currfile\@cpath@path%
309 \path@filename\stex@currfile\stex@currfilename%
310 \StrLen\stex@currfilename[\stex@currfile@tmp]%
311 \StrGobbleRight\stex@currfile{\the\numexpr%
312   \stex@currfile@tmp+1 }[\stex@currpath]%
313 \global\let\stex@currfile\stex@currfile%
314 \global\let\stex@currpath\stex@currpath%
315 \global\let\stex@currfilename\stex@currfilename%
316 }
317 \def\stex@currfile@pop{%
318   \ifx\stex@currfile@stack\@empty%
319     \global\let\stex@currfile\stex@mainfile%
320     \global\let\stex@currpath\stex@PWD%
321     \global\let\stex@currfilename\jobname%
322   \else%
323     \StrCut\stex@currfile@stack,\stex@currfile\stex@currfile@stack%
324     \path@filename\stex@currfile\stex@currfilename%
325     \StrLen\stex@currfilename[\stex@currfile@tmp]%
326     \StrGobbleRight\stex@currfile{\the\numexpr%
327       \stex@currfile@tmp+1 }[\stex@currpath]%
328     \global\let\stex@currfile\stex@currfile%
329     \global\let\stex@currpath\stex@currpath%
330     \global\let\stex@currfilename\stex@currfilename%
331   \fi%
332 }

```

\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:

```

333 \def\stexinput#1{%
334   \stex@iffileexists{#1}{%
335     \stex@currfile@push\stex@temp@path%
336     \input{\stex@currfile}%
337     \stex@currfile@pop%
338   }%
339   {%
340     \PackageError{stex}{File does not exist %
341       (#1): \stex@temp@path}{}%
342   }%
343 }
344 \def\stex@iffileexists#1#2#3{%
345   \edef\stex@temp@path{#1}%
346   \if@iswindows@\path@to@windows\stex@temp@path\fi%
347   \IfFileExists\stex@temp@path{#2}{#3}%
348 }
349 \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:

```

350 \@kpsewhich\mathhub@path{--var-value MATHHUB}
351 \if@iswindows@\windows@to@path\mathhub@path\fi
352 \ifx\mathhub@path\@empty
353   \PackageWarning{stex}{MATHHUB system variable not %
354     found or wrongly set}{%}
355   \defpath{MathHub}{%}
356 \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.

```

357 \def\mathhub@findmanifest#1{%
358   \@cpath{#1}%
359   \ifx\@cpath@path\@Slash%
360     \def\manifest@mf{%}
361   \else\ifx\@cpath@path\@empty%
362     \def\manifest@mf{%}
363   \else%
364     \edef\@findmanifest@path{\@cpath@path/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         \edef\@findmanifest@path{\@cpath@path/meta-inf/MANIFEST.MF}%
377         \if@iswindows@\path@to@windows\@findmanifest@path\fi%
378         \IfFileExists{\@findmanifest@path}{%
379           \edef\manifest@mf{\@findmanifest@path}%
380           \xdef\temp@archive@dir{\ex\detokenize\ex{\@cpath@path}}%
381         }{%
382           \mathhub@findmanifest{\@cpath@path/.}%
383         }%
384       \fi\fi%
385 }

```

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

```

386 \def\split@manifest@key{%
387   \IfSubStr{\manifest@line}{\@Colon}{%
388     \StrBefore{\manifest@line}{\@Colon}[\manifest@key]%
389     \StrBehind{\manifest@line}{\@Colon}[\manifest@line]%
390     \path@trimstring\manifest@line%
391     \path@trimstring\manifest@key%
392   }{%
393     \def\manifest@key{}%
394   }%
395 }

```

the next helper function iterates over lines in MANIFEST.MF

```

396 \def\parse@manifest@loop{%
397   \ifeof\@manifest%
398   \else%
399     \read\@manifest to \manifest@line\relax%
400     \split@manifest@key%
401     % id
402     \IfStrEq\manifest@key{id}{%
403       \xdef\manifest@mf{id}\manifest@line}%
404     }{%
405       % narration-base
406       \IfStrEq\manifest@key{narration-base}{%
407         \xdef\manifest@mf@narr{\manifest@line}%
408       }{%
409         % namespace
410         \IfStrEq\manifest@key{source-base}{%
411           \xdef\manifest@mf@ns{\manifest@line}%
412         }{%
413           \IfStrEq\manifest@key{ns}{%
414             \xdef\manifest@mf@ns{\manifest@line}%
415           }{%
416             % dependencies
417             \IfStrEq\manifest@key{dependencies}{%
418               \xdef\manifest@mf@deps{\manifest@line}%
419             }{%
420               }}}}}%
421     \parse@manifest@loop%
422   \fi%
423 }

```

`\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.

```

424 \newread\@manifest
425 \def\mathhub@parsemanifest#1#2{%
426   \gdef\temp@archive@dir{}%
427   \mathhub@findmanifest{#2}%
428   \begingroup%
429     \newlinechar=-1%

```

```

430 \endlinechar=-1%
431 \gdef\manifest@mf@id{}%
432 \gdef\manifest@mf@narr{}%
433 \gdef\manifest@mf@ns{}%
434 \gdef\manifest@mf@deps{}%
435 \immediate\openin\@manifest=\manifest@mf\relax%
436 \parse@manifest@loop%
437 \immediate\closein\@manifest%
438 \endgroup%
439 \if@iswindows@\windows@to@path\manifest@mf\fi%
440 \cslet{#1id}\manifest@mf@id%
441 \cslet{#1narr}\manifest@mf@narr%
442 \cslet{#1ns}\manifest@mf@ns%
443 \cslet{#1deps}\manifest@mf@deps%
444 \ifcsvoid{manifest@mf@id}{}%
445 \cslet{#1dir}\temp@archive@dir%
446 }%
447 }

```

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

```

```

471 \fi%
472 }
473
474 \def\@setcurrentreposinfo{%
475 \edef\mh@currentrepos{\currentrepos@id}%
476 \ifcsvoid{currentrepos@dir}{\}%
477 \csxdef{mathhub@dir@\currentrepos@id}{\currentrepos@dir}%
478 \csxdef{mathhub@narr@\currentrepos@id}{\currentrepos@narr}%
479 \csxdef{mathhub@ns@\currentrepos@id}{\currentrepos@ns}%
480 \csxdef{mathhub@deps@\currentrepos@id}{\currentrepos@deps}%
481 }%
482 }

```

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

```

483 \newif\if@inmhrepos\@inmhreposfalse
484 \ifcsvoid{stex@PWD}{\}%
485 \mathhub@parsemanifest{currentrepos@}\stex@PWD
486 \@setcurrentreposinfo
487 \ifcsvoid{currentrepos@dir}{\message{sTeX: Not currently in a MathHub repository}}{\%
488 \message{Current sTeX repository: \mh@currentrepos}
489 }
490 }

```

3.3 Modules

```

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

```

Aux:

```

492 %\def\ignorespacesandpars{\begingroup\catcode13=10%
493 % \@ifnextchar\relax{\endgroup}{\endgroup}}

```

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

```

494 %\def\ignorespacesandparsafterend#1\ignorespaces\fi{#1%
495 % \fi\ignorespacesandpars}
496 %\def\ignorespacesandpars{\ifhmode\unskip\fi\@ifnextchar\par%
497 % {\ex\ignorespacesandpars\@gobble}{}}

```

Options for the module-environment:

```

498 \addmetakey*{module}{title}
499 \addmetakey*{module}{name}
500 \addmetakey*{module}{creators}
501 \addmetakey*{module}{contributors}
502 \addmetakey*{module}{srccite}
503 \addmetakey*{module}{ns}
504 \addmetakey*{module}{narr}

```

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

```

505 \ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%
506 \newrobustcmd\module@heading{%
507 \stepcounter{module}%

```



```

508 \ifmod@show%
509 \noindent{\textbf{Module} \thesection.\thetitle [\module@name]]}%
510 \sref@label@id{Module \thesection.\thetitle [\module@name]]}%
511 \ifx\module@title\empty : \quad\else\quad(\module@title)\hfill\\\fi%
512 \fi%
513 }%

```

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.

```

514
515 % meta-theory
516 \def\stex@metatheory{fomid:/foundation?Meta}
517 \protected\def\metatheory#1{%
518 \if@inimport\else\latexml@annotate@invisible{metatheory}{#1}{}\fi%
519 }
520
521 \newenvironment{module}[1][{}]{%
522 \begin{@module}[#1]%
523 \module@heading% make the headings
524 %\ignorespacesandpars
525 \parsemodule@maybesetcodes}{%
526 \end{@module}%
527 \ignorespacesafterend%
528 }%
529 \ifmod@show\surroundwithmdframed{module@om@common}\fi%

```

Some auxiliary methods:

```

530 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}
531 \def\addto@thismodule#1{%
532 \@ifundefined{this@module}{\fi}%
533 \expandafter\g@addto@macro@safe\this@module{#1}%
534 }%
535 }
536 \def\addto@thismoduleex#1{%
537 \@ifundefined{this@module}{\fi}%
538 \edef\addto@thismodule@exp{#1}%
539 \expandafter\expandafter\expandafter\g@addto@macro@safe%
540 \expandafter\this@module\expandafter{\addto@thismodule@exp}%
541 }}

```

@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`.

```

542 \newif\ifarchive@ns@empty@archive@ns@empty@false
543 \def\set@default@ns{%
544   \edef\@module@ns@temp{\stex@currpath}%
545   \if@iswindows@windows@to@path\@module@ns@temp\fi%
546   \archive@ns@empty@false%
547   \stex@debug{Generate new namespace^^J Filepath: \@module@ns@temp}%
548   \ifcvoid{mh@currentrepos}{\archive@ns@empty@true}%
549   {\ex\ifx\csname mathhub@ns@mh@currentrepos\endcsname\@empty\archive@ns@empty@true\fi}%
550   }%
551   \stex@debug{ \ifarchive@ns@empty@ Namespace empty\else Namespace not empty\fi}%
552   \ifarchive@ns@empty@%
553     \edef\@module@ns@tempuri{file\@Colon\@Slash\@Slash\@module@ns@temp}%
554   \else%
555     \edef\@module@filepath@temppath{\@module@ns@temp}%
556     \edef\@module@ns@tempuri{\csname mathhub@ns@mh@currentrepos\endcsname}%
557     \edef\@module@archivedirpath{\csname mathhub@dir@mh@currentrepos\endcsname\@Slash source}%
558     \edef\@module@archivedirpath{\ex\detokenize\ex{\@module@archivedirpath}}%
559     \IfBeginWith\@module@filepath@temppath\@module@archivedirpath{%
560       \StrLen\@module@archivedirpath[\ns@temp@length]%
561       \StrGobbleLeft\@module@filepath@temppath\ns@temp@length[\@module@filepath@temprest]%
562       \edef\@module@ns@tempuri{\@module@ns@tempuri\@module@filepath@temprest}%
563     }{}%
564   \fi%
565   \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]}%
566   \setkeys{module}{ns=\@module@ns@tempuri}%
567 }

```

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.

```

568 \def\set@next@moduleid{%
569   \path@filename@noext\stex@currfile\stex@next@moduleid@filename%
570   \edef\set@next@moduleid@csname{namespace@\@module@ns@\@QuestionMark\stex@next@moduleid@filename}%
571   \unless\ifcsname\set@next@moduleid@csname\endcsname%
572     \csgdef{\set@next@moduleid@csname}{0}%
573   \fi%
574   \edef\namespace@currnum{\csname\set@next@moduleid@csname\endcsname}%
575   \edef\module@temp@setidname{\noexpand\setkeys{module}{name=%
576     \stex@next@moduleid@filename\ex\unless\ex\ifnum\csname\set@next@moduleid@csname\endcsname=0.
577   \module@temp@setidname%
578   \csxdef{\set@next@moduleid@csname}{\the\numexpr\namespace@currnum+1}%
579 }

```

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
- `\ $\langle uri \rangle$` that expands to `\invoke@module{ $\langle uri \rangle$ }` (see below).
- `\stex@module@ $\langle name \rangle$` that expands to $\langle uri \rangle$, if unambiguous, otherwise to ambiguous.

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

```
580 \newenvironment{@module}[1][]{%
581   \metasetkeys{module}{#1}%
582   \ifcvoid{module@name}{\let\module@name\module@id}{}% % TODO deprecate
583   \ifcvoid{module@name}{\set@next@moduleid}{}%
584   \let\module@id\module@name % TODO deprecate
585   \ifcvoid{currentmodule@uri}{%
586     \ifx\module@ns\@empty\set@default@ns\fi%
587     \ifx\module@narr\@empty%
588       \setkeys{module}{narr=\module@ns}%
589     \fi%
590   }{%
591     \if@smsmode%
592       \ifx\module@ns\@empty\set@default@ns\fi%
593       \ifx\module@narr\@empty%
594         \setkeys{module}{narr=\module@ns}%
595       \fi%
596     \else%
597       % Nested Module:
598       \stex@debug{Nested module! Parent: \currentmodule@uri}%
599       \setkeys{module}{name=\currentmodule@name\@Slash\module@name}%
600       \let\module@id\module@name % TODO deprecate
601       \setkeys{module}{ns=\currentmodule@ns}%
602     \fi%
```

```

603 }%
604 \edef\module@uri{\module@ns\@QuestionMark\module@name}%
605 \csgdef\module@names@\module@uri}{}%
606 \csgdef\module@imports@\module@uri}{}%
607 \csxdef\module@uri{\noexpand\@invoke@module{\module@uri}}%
608 \ifcsvoid\stex@module@\module@name}{%
609   \ex\global\ex\let\csname stex@module@\module@name\ex\endcsname\csname\module@uri\endcsname%
610 }{%
611   \ex\edef\csname stex@module@\module@name\endcsname{\detokenize{ambiguous}}%
612 }%
613 \edef\this@module{%
614   \ex\noexpand\csname module@defs@\module@uri\endcsname%
615 }%
616 \ex\xdef\csname stex@lastmodule@\module@name\endcsname{\module@uri}%
617 \csdef\module@defs@\module@uri}{}%
618 \ifcsvoid\mh@currentrepos}{}%
619   \@inmhrepostrue%
620   \addto@thismodule{\ex\edef\ex\noexpand\csname mh@old@repos@\module@uri\endcsname%
621     {\noexpand\mh@currentrepos}}%
622   \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\mh@currentrepos}}%
623 }%
624 \let\currentmodule@name\module@name%
625 \let\currentmodule@ns\module@ns%
626 \let\currentmodule@uri\module@uri%
627 \stex@debug{^^JNew module: \module@uri^^J}%
628 \parsemodule@maybesetcodes%
629 \begin{latexml@annotateenv}{theory}{\module@uri}%
630 }{%
631   \end{latexml@annotateenv}%
632   \if@inmhrepos%
633     \@inmhreposfalse%
634     \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\expandafter\noexpand\csname mh@old@
635   \fi%
636 }%
637 \newenvironment{@structural@feature}[2]{%
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     \begin{latexml@annotateenv}{feature:#1}{\currentmodule@uri\QuestionMark#2}%
645     \edef\currentmodule@uri{\currentmodule@ns\@QuestionMark\currentmodule@name}%
646     \parsemodule@maybesetcodes%
647   }{%
648     \end{latexml@annotateenv}%
649   }%
650 \newcommand\structural@feature[3]{\begingroup%
651   \ifcsvoid{currentmodule@uri}{%
652     \set@default@ns\let\currentmodule@ns\module@ns%

```

```

653 \set@next@moduleid\let\currentmodule@name\module@name%
654 }{}%
655 \edef\currentmodule@name{\currentmodule@name\@Slash#2\_feature}%
656 \parsemodule@maybesetcodes%
657 \latexml@annotate{feature:#1}{\currentmodule@uri\QuestionMark#2}{%
658 \edef\currentmodule@uri{\currentmodule@ns\@QuestionMark\currentmodule@name}%
659 #3}%
660 \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 $\langle uri \rangle$ and $\langle stex@module@id \rangle$, that ultimately expand to $\langle @invoke@module\langle uri \rangle \rangle$. Currently, the only functionality is $\langle @invoke@module\langle uri \rangle \rangle \langle @URI \rangle$, which expands to the full uri of a module (i.e. via $\langle stex@module@id \rangle \langle @URI \rangle$). In the future, this macro can be extended with additional functionality, e.g. accessing symbols in a macro for overloaded (macro-)names.

```

661 \def\@URI{uri} % TODO check this
662 \def\@invoke@module#1#2{%
663   \ifx\@URI#2%
664     #1%
665   \else%
666     % TODO something else
667     #2%
668   \fi%

```

669 }

3.4 Inheritance

3.4.1 Selective Inclusion

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

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

```
\parsemodule@allow* The first step is setting up a functionality for registering \sTeX macros and envi-
                     ronments as part of a module signature.
670 \newif\if@smsmode\@smsmodefalse
671 \def\parsemodule@allow#1{%
672   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#1\endcsname}%
673 }
674 \def\parsemodule@allowenv#1{%
675   \ex\def\csname parsemodule@allowedenv@#1\endcsname{#1}%
676 }
677 \def\parsemodule@replacemacro#1#2{%
678   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#2\endcsname}%
679 }
680 \def\parsemodule@replaceenv#1#2{%
681   \ex\def\csname parsemodule@allowedenv@#1\endcsname{#2}%
682 }
683 \def\parsemodule@escapechar@beginstring{begin}
684 \def\parsemodule@escapechar@endstring{end}

                     and now we use that to actually register all the STeX functionality as relevant
                     for sms mode.
685 \parsemodule@allow{symdef}
686 \parsemodule@allow{abbrdef}
687 \parsemodule@allow{metatheory}
688 \parsemodule@allow{importmodule}
689 \parsemodule@allowenv{module}
690 \parsemodule@allowenv{@module}
691 \parsemodule@allow{importmhmodule}
692 \parsemodule@allow{gimport}
693 \parsemodule@allowenv{modsig}
694 \parsemodule@allowenv{mhmodsig}
695 \parsemodule@allowenv{mhmodnl}
696 \parsemodule@allowenv{modnl}
697 \parsemodule@allowenv{@structural@feature}
698 \parsemodule@allow{symvariant}
699 \parsemodule@allow{structural@feature}
```

```

700 \parsemodule@allow{syml}
701 \parsemodule@allow{syml}
702 \parsemodule@allow{syml}
703 \parsemodule@allow{syml}
704 \parsemodule@allow{notation}
705 \parsemodule@allow{symdecl}
706
707 % to deprecate:
708
709 \parsemodule@allow{defi}
710 \parsemodule@allow{defii}
711 \parsemodule@allow{defiii}
712 \parsemodule@allow{defiv}
713 \parsemodule@allow{adefi}
714 \parsemodule@allow{adefii}
715 \parsemodule@allow{adefiii}
716 \parsemodule@allow{adefiv}
717 \parsemodule@allow{defis}
718 \parsemodule@allow{defiis}
719 \parsemodule@allow{defiiis}
720 \parsemodule@allow{defivs}
721 \parsemodule@allow{Defi}
722 \parsemodule@allow{Defii}
723 \parsemodule@allow{Defiii}
724 \parsemodule@allow{Defiv}
725 \parsemodule@allow{Defis}
726 \parsemodule@allow{Defiis}
727 \parsemodule@allow{Defiiis}
728 \parsemodule@allow{Defivs}

```

To read external modules without producing output, `\requiremodules` redefines the `\`-character to be an *active* character that, instead of executing a macro, checks whether a macro name has been registered using `\parsemodule@allow` before selectively executing the corresponding macro or ignoring it. To produce the relevant code, we therefore define a macro `\@active@slash` that produces a `\`-character with category code 13 (*active*), as well as `\@open@brace` and `\@close@brace`, which produce open and closing braces with category code 12 (*other*).

```

729 \catcode'\.=0
730 .catcode'\.=13
731 .def.\@active@slash{\}
732 .catcode'\.<=1
733 .catcode'\.>=2
734 .catcode'\.{=12
735 .catcode'\.}=12
736 .def.\@open@brace<{>
737 .def.\@close@brace<>
738 .catcode'\.=0
739 \catcode'\.=12

```

```

740 \catcode'\{=1
741 \catcode'\}=2
742 \catcode'\<=12
743 \catcode'\>=12

```

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

`\set@parsemodule@catcodes`

```

744 \def\parsemodule@ignorepackageerrors{,inputenc,}
745 \let\parsemodule@old@PackageError\PackageError
746 \def\parsemodule@packageerror#1#2#3{%
747   \IfSubStr\parsemodule@ignorepackageerrors{,#1,}{}{}%
748   \parsemodule@old@PackageError{#1}{#2}{#3}%
749   }%
750 }
751 \def\set@parsemodule@catcodes{%
752   \ifcat'\=0%
753     \global\catcode'\=13%
754     \global\catcode'\#=12%
755     \global\catcode'\{=12%
756     \global\catcode'\}=12%
757     \global\catcode'\$=12%$
758     \global\catcode'\^=12%
759     \global\catcode'\_ =12%
760     \global\catcode'\&=12%
761     \ex\global\ex\let\@active@slash\parsemodule@escapechar%
762     \global\let\parsemodule@old@PackageError\PackageError%
763     \global\let\PackageError\parsemodule@packageerror%
764     \fi%
765 }

```

`\reset@parsemodule@catcodes`

```

766 \def\reset@parsemodule@catcodes{%
767   \ifcat'\=13%
768     \global\catcode'\=0%
769     \global\catcode'\#=6%
770     \global\catcode'\{=1%
771     \global\catcode'\}=2%
772     \global\catcode'\$=3%$
773     \global\catcode'\^=7%
774     \global\catcode'\_ =8%
775     \global\catcode'\&=4%
776     \global\let\PackageError\parsemodule@old@PackageError%
777     \fi%
778 }

```

`\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.

```
779 \def\parsemodule@maybesetcodes{%
780   \if@smsmode\set@parsemodule@catcodes\fi%
781 }
```

`\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:

```
782
783 \def\parsemodule@escapechar{%
784   \def\parsemodule@escape@currcls{}%
785   \parsemodule@escape@parse@nextchar%
786 }%
```

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`.

```
787 \long\def\parsemodule@escape@parse@nextchar#1{%
788   \ifcat a#1\relax%
789     \edef\parsemodule@escape@currcls{\parsemodule@escape@currcls#1}%
790     \let\parsemodule@do@next\parsemodule@escape@parse@nextchar%
791   \else%
792     \def\parsemodule@last@char{#1}%
793     \ifx\parsemodule@escape@currcls\@empty%
794       \def\parsemodule@do@next{}%
795     \else%
796       \def\parsemodule@do@next{\parsemodule@escapechar@checkcls}%
797     \fi%
798   \fi%
799   \parsemodule@do@next%
800 }
```

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.

```

801 \def\parsemodule@escapechar@checkcs{%
802   \ifx\parsemodule@escape@currccs\parsemodule@escapechar@beginstring%
803     \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@last@char}%
804   \else%
805     \ifx\parsemodule@escape@currccs\parsemodule@escapechar@endstring%
806       \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@last@char}%
807     \else%
808       \ifcsvoid{parsemodule@allowedmacro@\parsemodule@escape@currccs}{%
809         \def\parsemodule@do@next{\relax\parsemodule@last@char}%
810       }{%
811         \ifx\parsemodule@last@char\@open@brace%
812           \ex\let\ex\parsemodule@do@next@ii\csname parsemodule@allowedmacro@\parsemodule@escape@currccs\endcsname%
813         \edef\parsemodule@do@next{\noexpand\parsemodule@converttoproperbraces\@open@brace}%
814         \else%
815           \reset@parsemodule@catcodes%
816           \edef\parsemodule@do@next{\ex\noexpand\csname parsemodule@allowedmacro@\parsemodule@escape@currccs\endcsname}%
817         \fi%
818       }%
819     \fi%
820   \fi%
821   \parsemodule@do@next%
822 }

```

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.

```

823 \ex\ex\ex\def%
824 \ex\ex\ex\parsemodule@converttoproperbraces%
825 \ex\@open@brace\ex#\ex1\@close@brace{%
826   \reset@parsemodule@catcodes%
827   \parsemodule@do@next@ii{#1}%
828 }

```

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.

```

829 \ex\ex\ex\def%
830 \ex\ex\ex\parsemodule@escapechar@checkbeginenv%
831 \ex\@open@brace\ex#\ex1\@close@brace{%
832   \ifcsvoid{parsemodule@allowedenv@#1}{%
833     \def\parsemodule@do@next{#1}%
834   }{%
835     \reset@parsemodule@catcodes%

```

```

836     \edef\parsemodule@envname{\csname parsemodule@allowedenv@#1\endcsname}%
837     \ex\def\ex\parsemodule@do@next\ex{%
838         \ex\begin\ex{\parsemodule@envname}%
839     }%
840 }%
841 \parsemodule@do@next%
842 }
843 \ex\ex\ex\def%
844 \ex\ex\ex\parsemodule@escapechar@checkendenv%
845 \ex\@open@brace\ex#\ex1\@close@brace{%
846     \ifcvoid{parsemodule@allowedenv@#1}{%
847         \def\parsemodule@do@next{#1}%
848     }{%
849         \edef\parsemodule@envname{\csname parsemodule@allowedenv@#1\endcsname}%
850         \ex\def\ex\parsemodule@do@next\ex{%
851             \ex\end\ex{\parsemodule@envname}%
852         }%
853     }%
854     \parsemodule@do@next%
855 }

```

`\@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.

```

856 \newrobustcmd\@requiremodules[1]{%
857     \if@tempswa\requiremodules{#1}\fi%
858 }%

```

`\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.

```

859 \newrobustcmd\requiremodules[1]{%
860     \mod@showfalse%
861     \edef\mod@path{#1}%
862     \edef\mod@path{\expandafter\detokenize\expandafter{\mod@path}}%
863     \requiremodules@smsmode{#1}%
864 }%

```

`\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.

```

865 \newbox\modules@import@tempbox
866 \def\requiremodules@smsmode#1{%
867     \setbox\modules@import@tempbox\vbox{%
868         \@smsmodetrue%
869         \set@parsemodule@catcodes%
870         \hbadness=100000\relax%
871         \hfuzz=10000pt\relax%

```

```

872      \vbadness=100000\relax%
873      \vfuzz=10000pt\relax%
874      \stexinput{#1.tex}%
875      \reset@parsemodule@catcodes%
876    }%
877    \parsemodule@maybesetcodes%
878  }

```

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

```

879 \newif\if@importmodule@switchrepos\@importmodule@switchreposfalse
880 \def\importmodule@bookkeeping#1#2#3{%
881   \@importmodule@switchreposfalse%
882   \stex@debug{Importmodule: #1^^J #2^^J\detokenize{#3}}%
883   \metasetkeys{importmodule}{#1}%
884   \ifcvoid{importmodule@mhrepos}{%
885     \ifcvoid{currentrepos@dir}{%
886       \stex@debug{Importmodule: Set importmodule@dir to \stex@PWD}%
887       \let\importmodule@dir\stex@PWD%
888     }{%
889       \stex@debug{Importmodule: Set importmodule@dir to \currentrepos@dir\@Slash source}%
890       \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
891     }%
892   }{%
893     \@importmodule@switchrepositotrue%
894     \stex@debug{Importmodule: Repository switch to \importmodule@mhrepos}%
895     \stex@debug{Importmodule: Current repos: \mh@currentrepos}%
896     \ex\let\csname importmodule@oldrepos@#2\endcsname\mh@currentrepos%
897     \mathhub@setcurrentreposinfo\importmodule@mhrepos%
898     \stex@debug{Importmodule: New repos: \mh@currentrepos^^J Namespace: \currentrepos@ns}%
899     \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
900   }%
901   \StrCut{#2}\@QuestionMark\importmodule@subdir\importmodule@modulename%
902   \ifx\importmodule@modulename\@empty%
903     \let\importmodule@modulename\importmodule@subdir%
904     \let\importmodule@subdir\@empty%
905   \else%
906     \ifx\importmodule@subdir\@empty\else%
907       \edef\importmodule@dir{\importmodule@dir\@Slash\importmodule@subdir}%
908     \fi%
909   \fi%
910   #3%
911   \if@importmodule@switchrepos%
912     \ex\mathhub@setcurrentreposinfo\csname importmodule@oldrepos@#2\endcsname%
913     \stex@debug{Importmodule: switched back to: \mh@currentrepos}%

```

```

914 \fi%
915 %\ignorespacesandpars%
916 }

\importmodule

917 %\srefaddidkey{importmodule}
918 \addmetakey{importmodule}{mhrepos}
919 \newcommand\importmodule[2][\@@importmodule[#1]{#2}{export}]
920 \newcommand\@@importmodule[3][\%
921 \importmodule@bookkeeping{#1}{#2}{\%
922 \importmodule[\importmodule@dir]\importmodule@modulename{#3}%
923 }%
924 }

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

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

925 \newcommand\@importmodule[3][\%
926 {\%
927 \edef\@load{#1}%
928 \edef\@importmodule@name{#2}%
929 \stex@debug{Loading #1}%
930 \if@smsmode\else\ifcsvoid\stex@module@\@importmodule@name{\% TODO check this
931 \stex@iffileexists\@load{
932 \stex@debug{Exists: #1}%
933 \requiremodules\@load}{\%
934 \stex@debug{Does not exist: #1~JTrying \@load\@Slash\@importmodule@name}%
935 \requiremodules{\@load\@Slash\@importmodule@name}%
936 }%
937 }{\fi%
938 \ifx\@load\@empty\else%
939 {\% TODO
940 \% \edef\@path{\csname module@#2@path\endcsname}%
941 \% \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do no
942 \% {\PackageError{stex}% else signal an error
943 \% {Module Name Clash\MessageBreak%
944 \% A module with name #2 was already loaded under the path "\@path"\MessageBreak%
945 \% The imported path "\@load" is probably a different module with the\MessageBreak%
946 \% same name; this is dangerous -- not importing}%
947 \% {Check whether the Module name is correct}%
948 \% }%
949 }%
950 \fi%
951 \global\let\@importmodule@load\@load%

```

```

952 }%
953 \edef\@export{#3}\def\@export{export}%prepare comparison
954 %\ifx\@export\@export\export@defs{#2}\fi% export the module
955 \ifx\@export\@export\addto@thismodulex{%
956   \noexpand\importmodule[\@importmodule@load]{#2}{noexport}%
957 }%
958 \if@smsmode\else
959 \ifcsvoid{this@module}{}{%
960   \ifcsvoid{module@imports@\module@uri}{
961     \csxdef{module@imports@\module@uri}{%
962       \csname stex@module@#2\endcsname\@URI% TODO check this
963     }%
964   }{%
965     \csxdef{module@imports@\module@uri}{%
966       \csname stex@module@#2\endcsname\@URI,% TODO check this
967       \csname module@imports@\module@uri\endcsname%
968     }%
969   }%
970 }%
971 \fi\fi%
972 \if@smsmode\else%
973   \edef\activate@module@name{#2}%
974   \StrCount\activate@module@name\@Slash[\activate@module@lastslash]%
975   \ifnum\activate@module@lastslash>0%
976     \StrCut[\activate@module@lastslash]\activate@module@name\@Slash\activate@module@temp\activa
977   \fi%
978   \ifcsvoid{stex@lastmodule@\activate@module@name}{%
979     \PackageError{stex}{No module with name \activate@module@name found}{}%
980   }{%
981     \ex\ex\ex\activate@defs\ex\ex\ex{\csname stex@lastmodule@\activate@module@name\endcsname}%
982   }%
983   \fi% activate the module
984 }%

```

Test 22: \importmodule {testmoduleimporta}:

```

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

```

Test 23: \importmodule {testmoduleimportb?importb}:

```

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

```

Test 24: >\relax<

```

>\macro:->\@invoke@symbol {fomid:/core/foundations/types?type.en?type}<

```

Default document module:

```

985 \AtBeginDocument{%
986   \set@default@ns%

```

```

987 \ifx\module@narr\@empty\setkeys{module}{narr=\module@ns}\fi%
988 \let\module@name\jobname%
989 \let\module@id\module@name % TODO deprecate
990 \edef\module@uri{\module@ns\@QuestionMark\module@name}%
991 \csgdef{module@names@\module@uri}{}%
992 \csgdef{module@imports@\module@uri}{}%
993 \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
994 \expandafter\global\expandafter\let\csname stex@module@\module@name\expandafter\endcsname\csname
995 \edef\this@module{%
996   \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
997 }%
998 \latexml@annotate@invisible{namespace}{\module@ns\@Slash\module@name}{}%
999 \csdef{module@defs@\module@uri}{}%
1000 \ifcsvoid{mh@currentrepos}{-}{%
1001   \@inmhrepostrue%
1002   \addto@thismodule{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\endcsname
1003     {\noexpand\mh@currentrepos}}%
1004   \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\mh@currentrepos}}%
1005 }%
1006 }

```

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.

```

1007 \newif\if@inimport\@inimportfalse
1008 \def\latexml@import#1{\latexml@annotate@invisible{import}{#1}{}}%
1009 \def\activate@defs#1{%
1010   \stex@debug{Activating import #1}%
1011   \if@inimport\else%
1012     \latexml@import{#1}%
1013     \def\inimport@module{#1}%
1014     \stex@debug{Entering import #1}%
1015     \@inimporttrue%
1016   \fi%
1017   \edef\activate@defs@uri{#1}%
1018   \ifcsundef{module@defs@\activate@defs@uri}{%
1019     \PackageError{stex}{No module with URI \activate@defs@uri loaded}{Probably missing an
1020       \detokenize{\importmodule} (or variant) somewhere?
1021   }
1022 }{%
1023   \ifcsundef{module@\activate@defs@uri @activated}%
1024     {\csname module@defs@\activate@defs@uri\endcsname}{}%
1025     \namedef{module@\activate@defs@uri @activated}{true}%
1026 }%
1027 \def\inimport@thismodule{#1}%
1028 \stex@debug{End of import #1}%

```

```

1029 \ifx\inimport@thismodule\inimport@module\@inimportfalse%
1030 \stex@debug{Leaving import #1}%
1031 \fi%
1032 }%

```

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

```

1033 \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.

```

1034 \def\inputref@preskip{}
1035 \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).

```

1036 \newrobustcmd\inputref[2] [] {%
1037 \importmodule@bookkeeping{#1}{#2}{%
1038 %\inputreftrue
1039 \inputref@preskip%
1040 \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
1041 \inputref@postskip%
1042 }%
1043 }%

```

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.

```

1044 \newif\if@symdeflocal\@symdeflocalfalse

```

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

```

1045 \def\define@in@module#1#2{
1046 \expandafter\edef\csname #1\endcsname{#2}%
1047 \edef\define@in@module@temp{%
1048 \def\expandafter\noexpand\csname#1\endcsname%
1049 {#2}%
1050 }%
1051 \if@symdeflocal\else%

```



```

1052 \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1053 \expandafter\endcsname\expandafter{\define@in@module@temp}%
1054 \fi%
1055 }

```

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

```

1056 \define@key{symdecl}{name}{\def\symdecl@name{#1}}%
1057 \define@key{symdecl}{type}{\def\symdecl@type{#1}}%
1058 \define@key{symdecl}{args}{\def\symdecl@args{#1}}%
1059 \define@key{symdecl}{local}[true]{\def\symdecl@local{#1}}%
1060
1061 \addmetakey[false]{symdecl}{local}[true]%
1062
1063 \newcommand\symdecl[2][]{%
1064 \def\symdecl@local{false}%
1065 \def\symdecl@name{}%
1066 \def\symdecl@type{}%
1067 \def\symdecl@args{}%
1068 \ifcsdef{this@module}{%
1069 \setkeys{symdecl}{#1}%
1070 \ifcvoid{symdecl@name}{
1071 \edef\symdecl@name{#2}%
1072 }{}%
1073 \edef\symdecl@uri{\module@uri\@QuestionMark\symdecl@name}%
1074 \stex@debug{Symdecl \symdecl@uri^^Jtype: \meaning\symdecl@type}%
1075 \ifcvoid{stex@symbol@\symdecl@name}{%
1076 \expandafter\edef\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}%
1077 }{}%
1078 \expandafter\def\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}%
1079 }%
1080 \edef\symdecl@symbolmacro{%
1081 \noexpand\ifcvoid{stex@symbol@\symdecl@name}{%
1082 \expandafter\edef\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\symd
1083 }{}%
1084 \expandafter\def\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\detok
1085 }%
1086 }%
1087 \if@inimport\else\if@smsmode\else\ifcvoid{symdecl@type}{}%
1088 \setbox\modules@import@tempbox\hbox{$\symdecl@type$}% only to have latex check this
1089 }\fi\fi%
1090 \ifcvoid{symdecl@args}{\csgdef{\symdecl@uri\@QuestionMark args}{}}{%
1091 \IfInteger\symdecl@args{\notation@num@to@ia@\symdecl@args\csxdef{\symdecl@uri\@QuestionMar
1092 \ex\global\ex\let\csname\symdecl@uri\@QuestionMark args\endcsname\symdecl@args%
1093 }%
1094 }%
1095 \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1096 \expandafter\endcsname\expandafter{\symdecl@symbolmacro}%

```

```

1097 \ifcsvoid{\symdecl@uri}{%
1098 \ifcsvoid{module@names@\module@uri}{%
1099 \csxdef{module@names@\module@uri}{\symdecl@name}%
1100 }{%
1101 \csxdef{module@names@\module@uri}{\symdecl@name,%
1102 \csname module@names@\module@uri\endcsname}%
1103 }%
1104 }{%
1105 % not compatible with circular dependencies, e.g. test/omdoc/07-modules/smsstesta.tex
1106 \PackageWarning{stex}{symbol already defined: \symdecl@uri}{%
1107 You need to pick a fresh name for your symbol%
1108 }%
1109 }%
1110 \define@in@module\symdecl@uri{\noexpand\@invoke@symbol{\symdecl@uri}}%
1111 \IfStrEq\symdecl@local{false}{%
1112 \define@in@module{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
1113 }{%
1114 \csdef{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
1115 }%
1116 }{%
1117 \PackageError{stex}{\detokenize{\symdecl} not in a module}{You need to be in a module%
1118 in order to declare a new symbol}
1119 }%
1120 \if@inimport\else\if@inabbrdef\else\if@smsmode\else%
1121 \latexml@symdecl\symdecl@uri{\$ \symdecl@type$}{\csname \symdecl@uri \@QuestionMark args\endcsname}%
1122 \fi\fi\fi%
1123 \if@insymdef\else\parsemodule@maybesetcodes\fi%
1124 }
1125 \def\latexml@symdecl#1#2#3#4#5{\latexml@annotate@invisible{\symdecl}{#1}{%
1126 \latexml@annotate{type}{#2}%
1127 \latexml@annotate{args}{#3}{%
1128 \latexml@annotate{definiens}{#4}%
1129 \latexml@annotate{macroname}{#2}{%
1130 }}

```

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:

```

1131 % TODO make this work with actual macros (it doesn't)
1132 \edef\stex@ambiguous{\detokenize{ambiguous}}
1133 \edef\stex@macrostring{\detokenize{\macro->\@invoke@symbol}}
1134 \def\modules@getURIfromName#1{%
1135 \def\notation@uri{}%
1136 \edef\modules@getURI@name{#1}%

```

```

1137 \ifcsvoid{\modules@getURI@name}{%
1138   \edef\modules@temp@meaning{%
1139 }{%
1140   \edef\modules@temp@meaning{\ex\meaning\csname\modules@getURI@name\endcsname}%
1141 }%
1142 \IfBeginWith\modules@temp@meaning\stex@macrostring{%
1143   % is a \@invoke@symbol macro
1144   \StrPosition\modules@temp@meaning\@close@brace[\stex@tempnum]%
1145   \StrMid\modules@temp@meaning{26}{\the\numexpr\stex@tempnum-1\@Space}[\notation@uri]%
1146 }{%
1147   % Check whether full URI or module?symbol or just name
1148   \StrCount\modules@getURI@name\@Colon[\isuri@number]%
1149   \ifnum\isuri@number>0%
1150     \edef\notation@uri{\modules@getURI@name}%
1151   \else
1152     \StrCount\modules@getURI@name\@QuestionMark[\isuri@number]%
1153     \ifnum\isuri@number=2%
1154       \edef\notation@uri{\modules@getURI@name}%
1155     \else%
1156       \ifnum\isuri@number=1%
1157         % module?name
1158         \StrCut\modules@getURI@name\@QuestionMark\isuri@mod\isuri@name%
1159         \ifcsvoid{stex@module@\isuri@mod}{%
1160           \PackageError{stex}{No module with name \isuri@mod\@Space loaded}{}%
1161         }{%
1162           \expandafter\ifx\csname stex@module@\isuri@mod\endcsname\stex@ambiguous%
1163           \PackageError{stex}{Module name \isuri@mod\@Space is ambiguous}{}%
1164         \else%
1165           \edef\notation@uri{\csname stex@module@\isuri@mod\endcsname\@URI\@QuestionMark\is
1166         \fi%
1167       }%
1168     \else%
1169       %name
1170       \ifcsvoid{stex@symbol@\modules@getURI@name}{%
1171         \PackageError{stex}{No symbol with name \modules@getURI@name\@Space known}{}%
1172       }{%
1173         \ifcsvoid{\module@uri\@QuestionMark\modules@getURI@name}{%
1174           \expandafter\ifx\csname stex@symbol@\modules@getURI@name\endcsname\stex@ambiguous%
1175           % Symbol name ambiguous and not in current module
1176           \PackageError{stex}{Symbol name, URI or macroname \detokenize{#1} found!}{}%
1177         \else%
1178           % Symbol not in current module, but unambiguous
1179           \edef\notation@uri{\csname stex@symbol@\modules@getURI@name\endcsname}%
1180         \fi%
1181       }{% Symbol in current module
1182         \edef\notation@uri{\module@uri\@QuestionMark\modules@getURI@name}%
1183       }%
1184     }%
1185   \fi%
1186 \fi%

```

```

1187     \fi%
1188   }%
1189 }

```

\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.

```

1190 \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`:

```

1191 \def\notation@parse@params#1#2{%
1192   \def\notation@curr@prec{%
1193     \def\notation@curr@args{%
1194       \def\notation@curr@variant{%
1195         \def\notation@curr@arityvar{%
1196           \def\notation@curr@provided@arity{#2}
1197           \def\notation@curr@lang{%
1198             \def\notation@options@temp{#1}
1199             \notation@parse@params%
1200             \ifx\notation@curr@args\@empty%
1201               \ifx\notation@curr@provided@arity\@empty%
1202                 \notation@num@to@ia\notation@curr@arityvar%
1203               \else%
1204                 \notation@num@to@ia\notation@curr@provided@arity%
1205               \fi%
1206             \fi%
1207             \StrLen\notation@curr@args[\notation@curr@arity]%
1208           }
1209           \def\notation@parse@params@{%
1210             \IfSubStr\notation@options@temp,{%
1211               \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
1212               \notation@parse@param%
1213               \notation@parse@params%
1214             }\ifx\notation@options@temp\@empty\else%
1215               \let\notation@option@temp\notation@options@temp%
1216               \notation@parse@param%
1217             \fi}%
1218         }
1219       }
1220     \def\notation@parse@param{%
1221       \path@trimstring\notation@option@temp%
1222       \ifx\notation@option@temp\@empty\else%
1223         \IfSubStr\notation@option@temp={%
1224           \StrCut\notation@option@temp=\notation@key\notation@value%
1225           \path@trimstring\notation@key%
1226           \path@trimstring\notation@value%

```

```

1227 \IfStrEq\notation@key{prec}{%
1228 \edef\notation@curr@prec{\notation@value}%
1229 }{%
1230 \IfStrEq\notation@key{args}{%
1231 \edef\notation@curr@args{\notation@value}%
1232 }{%
1233 \IfStrEq\notation@key{lang}{%
1234 \edef\notation@curr@lang{\notation@value}%
1235 }{%
1236 \IfStrEq\notation@key{variant}{%
1237 \edef\notation@curr@variant{\notation@value}%
1238 }{%
1239 \IfStrEq\notation@key{arity}{%
1240 \edef\notation@curr@arityvar{\notation@value}%
1241 }{%
1242 }}}}%
1243 }{%
1244 \edef\notation@curr@variant{\notation@option@temp}%
1245 }%
1246 \fi%
1247 }
1248
1249 % converts an integer to a string of 'i's, e.g. 3 => iii,
1250 % and stores the result in \notation@curr@args
1251 \def\notation@num@to@ia#1{%
1252 \IfInteger{#1}{
1253 \notation@num@to@ia@#1%
1254 }{%
1255 %
1256 }%
1257 }
1258 \def\notation@num@to@ia@#1{%
1259 \ifnum#1>0%
1260 \edef\notation@curr@args{\notation@curr@args i}%
1261 \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
1262 \fi%
1263 }
1264
1265 \newcount\notation@argument@counter
1266
1267 % parses the notation arguments and wraps them in
1268 % \notation@assoc and \notation@argprec for flexary arguments and precedences
1269 \providerobustcmd\notation[3][]{%
1270 \modules@getURIfromName{#2}%
1271 \notation@parse@params{#1}{}%
1272 \def\notation@temp@notation{%
1273 \ex\let\ex\notation@curr@args\csname\notation@uri\@QuestionMark args\endcsname%
1274 \let\notation@curr@todo@args\notation@curr@args%
1275 \StrLen\notation@curr@todo@args[\notation@curr@arity]%
1276 \ex\renewcommand\ex\notation@temp@notation\ex[\notation@curr@arity]{#3}%

```

```

1277 % precedence
1278 \let\notation@curr@precstring\notation@curr@prec%
1279 \IfSubStr\notation@curr@prec%;{%
1280   \StrCut\notation@curr@prec%;\notation@curr@prec\notation@curr@prec%
1281   \ifx\notation@curr@prec\@empty\def\notation@curr@prec{0}\fi%
1282 }{%
1283   \ifx\notation@curr@prec\@empty%
1284     \ifnum\notation@curr@arity=0\relax%
1285       \edef\notation@curr@prec{\infprec}%
1286     \else%
1287       \def\notation@curr@prec{0}%
1288     \fi%
1289   \else%
1290     \edef\notation@curr@prec{\notation@curr@prec}%
1291     \def\notation@curr@prec{%
1292       \fi%
1293   }%
1294 % arguments
1295 \notation@argument@counter=0%
1296 \def\notation@curr@extargs{%
1297   \notation@do@args%
1298 }
1299
1300 \edef\notation@ichar{\detokenize{i}}%
1301 \edef\notation@achar{\detokenize{a}}%
1302 \edef\notation@bchar{\detokenize{b}}%
1303
1304 % parses additional notation components for (associative) arguments
1305 \def\notation@do@args{%
1306   \advance\notation@argument@counter by 1%
1307   \def\notation@nextarg@temp{%
1308     \ifx\notation@curr@todo@args\@empty%
1309       \ex\notation@after%
1310     \else%
1311       % argument precedence
1312       \IfSubStr\notation@curr@prec{x}{%
1313         \StrCut\notation@curr@prec{x}\notation@curr@argprec\notation@curr@prec%
1314       }{%
1315         \edef\notation@curr@argprec{\notation@curr@prec}%
1316         \def\notation@curr@prec{%
1317           }%
1318         \ifx\notation@curr@argprec\@empty%
1319           \let\notation@curr@argprec\notation@curr@prec%
1320         \fi%
1321         \StrChar\notation@curr@todo@args1[\notation@argchar]%
1322         \edef\notation@argchar{\ex\detokenize\ex{\notation@argchar}}%
1323         \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1324         \ifx\notation@argchar\notation@ichar%
1325           % normal argument
1326           \edef\notation@nextarg@temp%

```

```

1327     {\stex@arg{\the\notation@argument@counter}{\notation@curr@argprec}{#####\the\notation
1328 }%
1329     \ex\g@addto@macro@safe\ex\notation@curr@extargs%
1330     \ex{\notation@nextarg@temp}%
1331     \ex\ex\ex\notation@do@args%
1332 \else\ifx\notation@argchar\notation@bchar%
1333     % bound argument
1334     \edef\notation@nextarg@temp{%
1335         {\stex@arg{\the\notation@argument@counter}{\notation@curr@argprec}{#####\the\notation
1336 }%
1337     \ex\g@addto@macro@safe\ex\notation@curr@extargs%
1338     \ex{\notation@nextarg@temp}%
1339     \ex\ex\ex\ex\ex\ex\notation@do@args%
1340 \else%
1341     % associative argument
1342     \ex\ex\ex\ex\ex\ex\ex\notation@parse@assocarg%
1343 \fi%
1344 \fi%
1345 \fi%
1346 }
1347
1348 \def\notation@parse@assocarg#1{%
1349 \def\notation@parse@assocop{#1}%
1350 \edef\notation@nextarg@temp{%
1351     {\stex@arg{\the\notation@argument@counter}{\notation@curr@argprec}{\notation@assoc{\ex\unexp
1352     {#####\the\notation@argument@counter}}}%
1353 }%
1354 \ex\g@addto@macro@safe\ex\notation@curr@extargs\ex{\notation@nextarg@temp}%
1355 \notation@do@args%
1356 }
1357
1358 \protected\def\safe@newcommand#1{%
1359 \ifdefined#1\ex\renewcommand\else\ex\newcommand\fi#1%
1360 }
1361
1362 % finally creates the actual macros
1363 \def\notation@after{
1364     % \notation@curr@prec
1365     % \notation@curr@args
1366     % \notation@curr@variant
1367     % \notation@curr@arity
1368     % \notation@curr@provided@arity
1369     % \notation@curr@lang
1370     % \notation@uri
1371     \def\notation@temp@fragment{}%
1372     \ifx\notation@curr@arityvar\@empty\else%
1373         \edef\notation@temp@fragment{arity=\notation@curr@arityvar}%
1374     \fi%
1375     \ifx\notation@curr@lang\@empty\else%
1376         \ifx\notation@temp@fragment\@empty%

```

```

1377     \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1378     \else%
1379     \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1380     \fi%
1381 \fi%
1382 \ifx\notation@curr@variant\@empty\else%
1383     \ifx\notation@temp@fragment\@empty%
1384     \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1385     \else%
1386     \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1387     \fi%
1388 \fi%
1389 \ex\ex\ex\def\ex\ex\ex\notation@temp@notation\ex\ex\ex%
1390     {\ex\notation@temp@notation\notation@curr@extargs}%
1391 \ifnum\notation@curr@arity=0%
1392     \edef\notation@temp@notation{\stex@dooms{\notation@uri}{\notation@temp@fragment}{\notation@
1393 \else%
1394     \IfSubStr\notation@curr@args\notation@bchar{%
1395     \edef\notation@temp@notation{\stex@doomb{\notation@uri}{\notation@temp@fragment}{\notation
1396     }{%
1397     \edef\notation@temp@notation{\stex@dooma{\notation@uri}{\notation@temp@fragment}{\notation
1398     }%
1399 \fi%
1400 \stex@debug{Notation \notation@uri: \meaning\notation@temp@notation}%
1401 \notation@final%
1402 \parsemodule@maybesetcodes%
1403 }
1404
1405 \def\notation@final{%
1406     \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1407     \stex@debug{Defining \notation@csname of arity \notation@curr@arity}%
1408     \ifcvoid{\notation@csname}{%
1409     \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1410     \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@curr@arity\ex]%
1411     \ex{\notation@temp@notation}%
1412     \edef\symdecl@temps{%
1413     \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@curr@ari
1414     }%
1415     \ex@g@addto@macro@safe\csname module@defs\module@uri\ex\endcsname\ex{\symdecl@temps}%
1416     \ex@g@addto@macro@safe\csname module@defs\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1417     }{%
1418     \PackageWarning{stex}{notation already defined: \notation@csname}{%
1419     Choose a different set of notation options (variant,lang,arity)%
1420     }%
1421     }%
1422     \@innotationfalse%
1423     \if@inimport\else\if@latexml%
1424     \let\notation@simarg@args\notation@curr@args%
1425     \notation@argument@counter=0%
1426     \def\notation@simargs{%

```



```

1427 \notation@simulate@arguments%
1428 \latexml@notation\notation@uri\notation@temp@fragment\notation@curr@args\notation@curr@prec
1429 {\$ \csname\notation@csname\ex\endcsname\notation@simargs$}%
1430 \fi\fi%
1431 }
1432 \def\notation@simulate@arguments{%
1433 \ifx\notation@simarg@args\@empty\else%
1434 \advance\notation@argument@counter by 1%
1435 \IfBeginWith\notation@simarg@args{i}{%
1436 \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\the\notation@argument
1437 }}{%
1438 \IfBeginWith\notation@simarg@args{b}{%
1439 \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\the\notation@argument
1440 }}{%
1441 \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\@Fragment\the\nota
1442 }}%
1443 }}%
1444 \StrGobbleLeft\notation@simarg@args1[\notation@simarg@args]%
1445 \notation@simulate@arguments%
1446 \fi%
1447 }
1448 % URI, fragment, arity, notation
1449 \def\latexml@notation#1#2#3#4#5{\latexml@annotate@invisible{notation}{#1}{%
1450 \latexml@annotate{notationfragment}{#2}{}%
1451 \latexml@annotate{args}{#3}{}%
1452 \latexml@annotate{precedence}{#4}{}%
1453 \latexml@annotate{notationcomp}{#5}%
1454 }}

```

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

```

1455 \protected\def\notation@assoc#1#2{% function, argv
1456 \let\@tmpop=\relax% do not print the function the first time round
1457 \@for\@I:=#2\do{\@tmpop% print the function
1458 % write the i-th argument with locally updated precedence
1459 \@I%
1460 \def\@tmpop{#1}%
1461 }%
1462 }%
1463
1464 \def\notation@lparen{()
1465 \def\notation@rparen{)}
1466 \def\infpref{1000000}
1467 \def\neginfpref{-\infpref}
1468
1469 \newcount\notation@downprec
1470 \notation@downprec=\neginfpref
1471
1472 % patching displaymode
1473 \newif\if@displaymode\@displaymodefalse

```

```

1474 \ex\everydisplay\ex{\the\everydisplay\@displaymodetrue}
1475 \let\old@displaystyle\displaystyle
1476 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1477
1478 \protected\def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1479   \def\notation@innertmp{#1}%
1480   \if@displaymode%
1481     \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1482     \ex\notation@resetbrackets\ex\notation@innertmp%
1483     \ex\right\notation@rparen%
1484   \else%
1485     \ex\ex\ex\notation@lparen%
1486     \ex\notation@resetbrackets\ex\notation@innertmp%
1487     \notation@rparen%
1488   \fi%
1489 }
1490
1491 \protected\def\withbrackets#1#2#3{%
1492   \edef\notation@lparen{#1}%
1493   \edef\notation@rparen{#2}%
1494   #3%
1495   \notation@resetbrackets%
1496 }
1497
1498 \protected\def\notation@resetbrackets{%
1499   \def\notation@lparen{()%
1500   \def\notation@rparen{)%}%
1501 }
1502
1503 \protected\def\stex@dooms#1#2#3#4{%
1504   \if@innotation%
1505     \notation@symprec{#3}{#4}%
1506   \else%
1507     \@innotationtrue%
1508     \latexml@oms{#1}{#2}{\notation@symprec{#3}{#4}}%
1509     \@innotationfalse%
1510   \fi%
1511 }
1512
1513 \protected\def\stex@doomb#1#2#3#4{%
1514   \if@innotation%
1515     \notation@symprec{#3}{#4}%
1516   \else%
1517     \@innotationtrue%
1518     \latexml@ombind{#1}{#2}{\notation@symprec{#3}{#4}}%
1519     \@innotationfalse%
1520   \fi%
1521 }
1522
1523 \protected\def\stex@dooma#1#2#3#4{%

```

```

1524 \if@innotation%
1525     \notation@symprec{#3}{#4}%
1526 \else%
1527     \@innotationtrue%
1528     \latexml@oma{#1}{#2}{\notation@symprec{#3}{#4}}%
1529     \@innotationfalse%
1530 \fi%
1531 }
1532
1533 % for LaTeXML Bindings
1534 \protected\def\latexml@oms#1#2#3{%
1535     \latexml@annotate{OMID}{#1\@Fragment#2}{#3}%
1536 }
1537
1538 \protected\def\latexml@oma#1#2#3{%
1539     \edef\latexml@oma@uri{%
1540         \ifcsname#1\@QuestionMark args\endcsname%
1541         #1\@Fragment\csname#1\@QuestionMark args\endcsname\@Fragment#2%
1542         \else#1\@Fragment\@Fragment#2\fi%
1543     }%
1544     \latexml@annotate{OMA}{\latexml@oma@uri}{#3}%
1545 }
1546
1547 \protected\def\latexml@ombind#1#2#3{%
1548     \edef\latexml@oma@uri{%
1549         \ifcsname#1\@QuestionMark args\endcsname%
1550         #1\@Fragment\csname#1\@QuestionMark args\endcsname\@Fragment#2%
1551         \else#1\@Fragment\@Fragment#2\fi%
1552     }%
1553     \latexml@annotate{OMBIND}{\latexml@oma@uri}{#3}%
1554 }
1555
1556 \def\notation@symprec#1#2{%
1557     \ifnum#1>\notation@downprec\relax%
1558         \notation@resetbrackets#2%
1559     \else%
1560         \ifnum\notation@downprec=\infprec\relax%
1561             \notation@resetbrackets#2%
1562         \else
1563             \if@inarray@
1564                 \notation@resetbrackets#2
1565             \else\dobrackets{#2}\fi%
1566         \fi\fi%
1567 }
1568
1569 \newif\if@inarray@\@inarray@false
1570
1571
1572 \protected\def\stex@arg#1#2#3{%
1573     \@innotationfalse%

```

```

1574 \latexml@arg{#1}{\notation@argprec{#2}{#3}}%
1575 \@innotationtrue%
1576 }
1577
1578 % for LaTeXML Bindings
1579 \def\latexml@arg#1#2{%
1580 \latexml@annotate{arg}{#1}{#2}%
1581 }
1582
1583 \def\notation@argprec#1#2{%
1584 \def\notation@innertmp{#2}
1585 \edef\notation@downprec@temp{\number#1}%
1586 \notation@downprec=\ex\notation@downprec@temp%
1587 \ex\relax\ex\notation@innertmp%
1588 \ex\notation@downprec\ex=\number\notation@downprec\relax%
1589 }

```

Macros for introducing MMT/OMDoc primitives manually

```

1590 \protected\def\stex@oms#1#2{\modules@getURIfromName{#1}\latexml@oms{\notation@uri}{#2}}
1591 \protected\def\stex@oma#1#2{\modules@getURIfromName{#1}\latexml@oma{\notation@uri}{#2}}
1592 \protected\def\stex@ombind#1#2{\modules@getURIfromName{#1}\latexml@ombind{\notation@uri}{#2}}
1593 \protected\def\stex@rule#1#2{%
1594 \latexml@annotate@invisible{mmtrule}{#1}{%
1595 \notation@argument@counter=0%
1596 \@for\@I:=#2\do{%
1597 \advance\notation@argument@counter by 1%
1598 \latexml@annotate{arg}{\the\notation@argument@counter}{\@I}%
1599 }%
1600 }%
1601 }

```

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

```

1602 \protected\def\@invoke@symbol#1{%
1603 \ifmmode%
1604 \def\@invoke@symbol@first{#1}%
1605 \let\invoke@symbol@next\invoke@symbol@math%
1606 \else%
1607 \def\invoke@symbol@next{\invoke@symbol@text{#1}}%
1608 \fi%
1609 \invoke@symbol@next%
1610 }

```

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

```

1611 \newcommand\invoke@symbol@math[1] [] {%
1612 \notation@parse@params{#1}{}%
1613 \def\notation@temp@fragment{}%
1614 \ifx\notation@curr@arityvar\empty\else%
1615 \edef\notation@temp@fragment{arity=\notation@curr@arity}%

```

```

1616 \fi%
1617 \ifx\notation@curr@lang\@empty\else%
1618 \ifx\notation@temp@fragment\@empty%
1619 \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1620 \else%
1621 \edef\notation@temp@fragment{\notation@temp@fragment\& lang=\notation@curr@lang}%
1622 \fi%
1623 \fi%
1624 \ifx\notation@curr@variant\@empty\else%
1625 \ifx\notation@temp@fragment\@empty%
1626 \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1627 \else%
1628 \edef\notation@temp@fragment{\notation@temp@fragment\& variant=\notation@curr@va
1629 \fi%
1630 \fi%
1631 \csname\@invoke@symbol@first\@Fragment\notation@temp@fragment\endcsname%
1632 }

```

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

```

1633 \def\setstexlang#1{%
1634 \def\stex@lang{#1}%
1635 }%
1636 \setstexlang{en}
1637 \def\setstexvariant#1#2{%
1638 % TODO
1639 }
1640 \def\setstexvariants#1{%
1641 \def\stex@variants{#1}%
1642 }

```

Test 29: Module 3.15[FooBar]: \symdecl [args=a]{plus}

\symdecl [args=a]{times}

\symdecl {vara}

\symdecl {varb}

\symdecl {varc}

\symdecl {vard}

\symdecl {vare}

\notation {vara}{a}

\notation {varb}{b}

\notation {varc}{c}

\notation {vard}{d}

\notation {vare}{e}

\notation [prec=500;500]{plus}{\withbrackets \langle \rangle {####1}}{+}

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

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

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

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

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

`\abbrdef` The `\abbrdef` macro is a variant of `\symdecl` that does the same on the L^AT_EX level, and adds a definiens on the OMDoc level.

```

1643 \newif\if@inabbrdef\@inabbrdeffalse
1644 \def\abbrdef@definiens{}
1645 \newcommand\abbrdef[3][\%
1646   \@inabbrdeftrue\symdecl[#1]{#2}%
1647   \@inabbrdeffalse%
1648   \ex\let\ex\abbrdef@args\csname\symdecl@uri\@QuestionMark args\endcsname%
1649   \StrLen\abbrdef@args[\abbrdef@arity]
1650   \ex\renewcommand\ex\abbrdef@definiens\ex[\abbrdef@arity]{\unexpanded{#3}}%
1651   \if@inimport\else\if@latexml%
1652     \let\notation@simarg@args\abbrdef@args%
1653     \notation@argument@counter=0%
1654     \def\notation@simargs{}\%
1655     \notation@simulate@arguments%
1656     \latexml\symdecl\symdecl@uri{\$ \symdecl@type$}{\csname\symdecl@uri\@QuestionMark args\endcsname
1657       {\$ \ex\abbrdef@definiens\notation@simargs$}{#2}%
1658   \fi\fi%
1659 }
```

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

```

1660 \newif\if@inoms
1661 \def\invoke@symbol@text#1{%
1662   \edef\invoke@symbol@uri{#1}%
1663   \def\invoke@symbol@return{}%
1664   \notation@argument@counter=0%
1665   \edef\invoke@symbol@arity{\csname #1\@QuestionMark args\endcsname}%
1666   \ifx\invoke@symbol@arity\empty\@inomsfalse\fi%
1667   \invoke@symbol@text@args%
```

```

1668 }
1669
1670 \edef\notation@Xchar{\detokenize{X}}%
1671
1672 \protected\def\opref#1{%
1673   \modules@getURIfromName{#1}%
1674   \let\invoke@symbol@uri\notation@uri%
1675   \def\invoke@symbol@return{}%
1676   \notation@argument@counter=0%
1677   \def\invoke@symbol@arity{}%
1678   \@inomstrue%
1679   \invoke@symbol@text@args%
1680 }
1681
1682 \def\invoke@symbol@text@args{%
1683   \advance\notation@argument@counter by 1%
1684   \edef\notation@charnum{\the\notation@argument@counter}%
1685   \StrChar\invoke@symbol@arity{\the\notation@argument@counter}[\invoke@symbol@nextchar]%
1686   \ifx\invoke@symbol@nextchar\notation@Xchar%
1687     \ex\invoke@symbol@text@args%
1688   \else%
1689     \ifx\invoke@symbol@nextchar\empty%
1690       \let\invoke@symbol@nextstep\invoke@symbol@text@finally%
1691       \ex\ex\ex\invoke@symbol@maybesqbracket%
1692     \else%
1693       \let\invoke@symbol@nextstep\invoke@symbol@normalarg%
1694       \ex\ex\ex\invoke@symbol@maybestarI%
1695     \fi%
1696   \fi%
1697 }
1698
1699 \def\invoke@symbol@maybestarI{%
1700   \@ifnextchar*{%
1701     \@ifnextchar[{%
1702       \invoke@symbol@switchnum%
1703     }{%
1704       \invoke@symbol@invisible%
1705     }%
1706   }{%
1707     \invoke@symbol@maybesqbracket%
1708   }%
1709 }
1710
1711 \def\invoke@symbol@maybesqbracket{%
1712   \@ifnextchar[{\invoke@symbol@verbcomp}{\invoke@symbol@nextstep}%
1713 }
1714
1715 \def\invoke@symbol@verbcomp[#1]{%
1716   \ex\def\ex\invoke@symbol@return\ex{\invoke@symbol@return #1}%
1717   \invoke@symbol@nextstep%

```

```

1718 }
1719
1720 \def\invoke@symbol@invisible*#1{% TODO a-args
1721   \edef\invoke@symbol@frame{\noexpand\latexml@annotate@invisible{arg}{\notation@charnum}}%
1722   \ex\ex\ex\def\ex\ex\ex\invoke@symbol@return\ex\ex\ex{\ex\invoke@symbol@return\invoke@symbol@f
1723   \invoke@symbol@text@args%
1724 }
1725
1726 \def\invoke@symbol@normalarg#1{% TODO a-args
1727   \edef\invoke@symbol@frame{\noexpand\latexml@annotate{arg}{\notation@charnum}}%
1728   \ex\ex\ex\def\ex\ex\ex\invoke@symbol@return\ex\ex\ex{\ex\invoke@symbol@return\invoke@symbol@f
1729   \invoke@symbol@text@args%
1730 }
1731
1732 \def\invoke@symbol@switchnum* [#1]{%
1733   \advance\notation@argument@counter by -1%
1734   \edef\notation@charnum{#1}%
1735   \StrChar\invoke@symbol@arity\notation@charnum[\invoke@symbol@nextchar]%
1736   \ifx\invoke@symbol@nextchar\notation@ichar%
1737     \StrLeft\invoke@symbol@arity{\numexpr\notation@charnum-1}[\invoke@symbol@newarityLeft]%
1738     \StrGobbleLeft\invoke@symbol@arity\notation@charnum[\invoke@symbol@newarity]%
1739     \edef\invoke@symbol@newarity{\invoke@symbol@newarityLeft\notation@Xchar\invoke@symbol@newar
1740   \else% TODO
1741     \fi%
1742   \invoke@symbol@maybestarII%
1743 }
1744
1745 \def\invoke@symbol@maybestarII{%
1746   \@ifnextchar*{%
1747     \invoke@symbol@invisible%
1748   }{%
1749     \invoke@symbol@normalarg%
1750   }%
1751 }
1752
1753 \def\invoke@symbol@text@finally{%
1754   \stex@debug{HERE! \meaning\invoke@symbol@return}%
1755   \if@inoms\latexml@oms{\invoke@symbol@uri}{\}{\invoke@symbol@return}%
1756   \else\latexml@oma{\invoke@symbol@uri}{\}{\invoke@symbol@return}%
1757   \fi%
1758 }

```

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`

```
1759 \newif\ifhref\hreffalse%
1760 \AtBeginDocument{%
1761   \ifpackageloaded{hyperref}{%
1762     \hreftrue%
1763   }{%
1764     \hreffalse%
1765   }%
1766 }
```

`\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.

```
1767 \newbox\stex@targetbox
1768 \def\termref@maketarget#1#2{%
1769   % #1: symbol URI
1770   % #2: text
1771   \stex@debug{Here: #1 <> #2}%
1772   \ifhref\if@smsmode\else%
1773     \hypertarget{sref@#1@target}{#2}%
1774   \fi\fi%
1775   \stex@debug{Here!}%
1776   \expandafter\edef\csname sref@#1\endcsname##1{%
1777     \ifhref\if@smsmode\else\noexpand\hyperlink{sref@#1@target}{##1}\fi\fi%
1778   }%
1779 }
```

`\@termref`

```
1780 \def\@termref#1#2{%
1781   % #1: symbol URI
1782   % #2: text
1783   \ifcvoid{#1}{%
1784     \StrCut[2]{#1}\@QuestionMark\termref@mod\termref@name%
1785     \ifcvoid{\termref@mod}{%
1786       \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{%
1787         }{%
1788           \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
1789             contains no symbol with name \termref@name.%
1790           }{%
1791             }%
1792         }%
1793       \ifcvoid{sref@#1}{%
1794         #2% TODO: No reference point exists!
1795       }{%
1796         \csname sref@#1\endcsname{#2}%
1797       }%
1798     }%
1799 }
```

```

\tref
1800
1801 \def\@capitalize#1{\uppercase{#1}}%
1802 \newrobustcmd\capitalize[1]{\expandafter\@capitalize #1}%
1803
1804 \newcommand\tref[2][]{%
1805   \edef\tref@name{#1}%
1806   \expandafter\modules@getURIfromName\expandafter{\tref@name}%
1807   \expandafter\@termref\expandafter{\notation@uri}{#2}%
1808 }
1809 \def\trefs#1{%
1810   \modules@getURIfromName{#1}%
1811   % TODO
1812 }
1813 \def\Tref#1{%
1814   \modules@getURIfromName{#1}%
1815   % TODO
1816 }
1817 \def\Trefs#1{%
1818   \modules@getURIfromName{#1}%
1819   % TODO
1820 }

\defi
1821 \addmetakey{defi}{name}
1822 \def\@definiendum#1#2{%
1823   \parsemodule@maybesetcodes%
1824   \stex@debug{Here: #1 | #2}%
1825   \termref@maketarget{#1}{#2}%\termref@maketarget{#1}{\defemph{#2}}%
1826 }
1827
1828 \newcommand\defi[2][]{%
1829   \metasetkeys{defi}{#1}%
1830   \ifx\defi@name\@empty%
1831     \symdecl@constructname{#2}%
1832     \let\defi@name\symdecl@name%
1833     \let\defi@verbalization\symdecl@verbalization%
1834   \else%
1835     \edef\defi@verbalization{#2}%
1836   \fi%
1837   \ifcvoid{\module@uri\@QuestionMark\defi@name}{%
1838     \symdecl\defi@name%
1839   }{\edef\symdecl@uri{\module@uri\@QuestionMark\defi@name}}%
1840   \@definiendum\symdecl@uri\defi@verbalization%
1841 }
1842 \def\Defi#1{%
1843   \symdecl{#1}%
1844   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization}%
1845 }
1846 \def\defis#1{%

```

```

1847 \symdecl{#1}%
1848 \@definiendum\symdecl@uri{\symdecl@verbalization s}%
1849 }
1850 \def\Defis#1{%
1851 \symdecl{#1}%
1852 \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization s}%
1853 }

```

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
1854 \newif\ifhref\hreffalse%
1855 \AtBeginDocument{%
1856 \ifpackageloaded{hyperref}{%
1857 \hreftrue%
1858 }{%
1859 \hreffalse%
1860 }%
1861 }%
1862 \newcommand\sref@href@ifh[2]{%
1863 \ifhref%
1864 \href{#1}{#2}%
1865 \else%
1866 #2%
1867 \fi%
1868 }%
1869 \newcommand\sref@hlink@ifh[2]{%
1870 \ifhref%
1871 \hyperlink{#1}{#2}%
1872 \else%
1873 #2%
1874 \fi%
1875 }%
1876 \newcommand\sref@target@ifh[2]{%
1877 \ifhref%
1878 \hypertarget{#1}{#2}%
1879 \else%
1880 #2%
1881 \fi%
1882 }%

```

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.

```

1883 \def\sref@target{%
1884   \ifx\sref@id\empty%
1885     \relax%
1886   \else%
1887     \edef\@target{sref@ifcsundef{sref@part}}{\sref@part @}\sref@id @target}%
1888     \sref@target@ifh\@target}%
1889   \fi%
1890 }%

\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.

1891 \addmetakey{srefaddidkey}{prefix}
1892 \newcommand\srefaddidkey[2][]{%
1893   \metasetkeys{srefaddidkey}{#1}%
1894   \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
1895   \metakeys@ext@clear@keys{#2}{id}{}%
1896   \metakeys@ext@showkeys{#2}{id}%
1897   \define@key{#2}{id}{%
1898     \edef\sref@id{\srefaddidkey@prefix ##1}%
1899     %\expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
1900     \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1901   }%
1902 }%

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

1903 \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.

1904 \ifextrefs%
1905   \newwrite\refs@file%
1906 \else%
1907   \def\refs@file{\@auxout}%
1908 \fi%

\sref@def This macro writes an \@sref@def command to the current aux file and also exe-
cutes it.

1909 \newcommand\sref@def[3]{%
1910   \protected@write\refs@file{}{\string\@sref@def{#1}{#2}{#3}}%
1911 }%

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

1912 \newcommand\sref@label[2]{%
1913   \sref@def{\ifcsundef{sref@part}}{\sref@part @}{#2}{page}{\thepage}%
1914   \sref@def{\ifcsundef{sref@part}}{\sref@part @}{#2}{label}{#1}%
1915 }%

```

`\sreflabel` The `\sreflabel` macro is a semantic version of `\label`, it combines the categorization given in the first argument with L^AT_EX's `\@currentlabel`.

```
1916 \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.

```
1917 \def\sref@id{} % make sure that defined
1918 \newcommand\sref@label@id[1]{%
1919   \ifx\sref@id\@empty%
1920     \relax%
1921   \else%
1922     \sref@label{#1}{\sref@id}%
1923   \fi%
1924 }%
```

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

```
1925 \newcommand\sref@label@id@arg[2]{%
1926   \def\@@id{#2}
1927   \ifx\@@id\@empty%
1928     \relax%
1929   \else%
1930     \sref@label{#1}{\@@id}%
1931   \fi%
1932 }%
```

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.

```
1933 \newenvironment{modsig}[2][]{\def\@test{#1}%
1934 \ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1935 \expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
1936 %\ignorespacesandpars
1937 }
1938 {\end{module}}\ignorespacesandpars
1939 }
```

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`.

```

1940 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1941 \newrobustcmd\@gimport@star[2] [] {\def\@test{#1}%
1942 \edef\mh@repos{\mh@currentrepos}%
1943 \ifx\@test\@empty%
1944 \importmhmodule[conservative,mhrepos=\mh@repos,path=#2]{#2}%
1945 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1946 \mathhub@setcurrentreposinfo{\mh@repos}%
1947 %\ignorespacesandpars
1948 \parsemodule@maybesetcodes}
1949 \newrobustcmd\@gimport@nostar[2] [] {\def\@test{#1}%
1950 \edef\mh@repos{\mh@currentrepos}%
1951 \ifx\@test\@empty%
1952 \importmhmodule[mhrepos=\mh@repos,path=#2]{#2}%
1953 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1954 \mathhub@setcurrentreposinfo{\mh@repos}%
1955 %\ignorespacesandpars
1956 \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.

```

1957 \def\modules@@first#1/#2;{#1}
1958 \newcommand\libinput[1]{%
1959 \stex@debug{Libinput current repo: \meaning\mh@currentrepos}%
1960 \ifcsvoid{mh@currentrepos}{%
1961   \PackageError{stex}{current MathHub repository not found}{}}%
1962 {}
1963 \edef\@mh@group{\expandafter\modules@@first\mh@currentrepos;}
1964 \let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
1965 \def\mh@inffile{\MathHub{\@mh@group/meta-inf/lib/#1}}
1966 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1967 \IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
1968 \IfFileExists\mh@inffile{}\IfFileExists\mh@libfile{}{%
1969   {\PackageError{stex}
1970     {Library file missing; cannot input #1.tex\MessageBreak%
1971       Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
1972       do not exist}%
1973   {Check whether the file name is correct}}}}

```

```

1974 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1975 \let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}

```

3.12 omdoc/omgroup

```

1976 \newcount\section@level
1977
1978 \section@level=2
1979 \ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{}
1980 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{}
1981 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{}
1982 \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>*.

```

1983 \newcommand\omgroup@nonum[2]{%
1984 \ifx\hyper@anchor\undefined\else\phantomsection\fi%
1985 \addcontentsline{toc}{#1}{#2}\@nameuse{#1}*{#2}}

```

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

```

1986 \newcommand\omgroup@num[2]{%
1987 \edef\@@ID{\sref@id}
1988 \ifx\omgroup@short\empty% no short title
1989 \@nameuse{#1}{#2}%
1990 \else% we have a short title
1991 \@ifundefined{rdfmeta@sectioning}%
1992   {\@nameuse{#1}[\omgroup@short]{#2}}%
1993   {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
1994 \fi%
1995 \sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@@ID}

```

`omgroup`

```

1996 \def\@true{true}
1997 \def\@false{false}
1998 \srefaddidkey{omgroup}
1999 \addmetakey{omgroup}{date}
2000 \addmetakey{omgroup}{creators}
2001 \addmetakey{omgroup}{contributors}
2002 \addmetakey{omgroup}{srccite}
2003 \addmetakey{omgroup}{type}
2004 \addmetakey*{omgroup}{short}
2005 \addmetakey*{omgroup}{display}
2006 \addmetakey[false]{omgroup}{loadmodules}[true]

```

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

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

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

```
2007 \newif\if@mainmatter\@mainmattertrue
2008 \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.

```
2009 \addmetakey{omdoc@sect}{name}
2010 \addmetakey[false]{omdoc@sect}{clear}[true]
2011 \addmetakey{omdoc@sect}{ref}
2012 \addmetakey[false]{omdoc@sect}{num}[true]
2013 \newcommand\omdoc@sectioning[3] [] {\metasetkeys{omdoc@sect}{#1}%
2014 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
2015 \if@mainmatter% numbering not overridden by frontmatter, etc.
2016 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
2017 \def\current@section@level{\omdoc@sect@name}%
2018 \else\omgroup@nonum{#2}{#3}%
2019 \fi}% if@mainmatter
```

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

```
2020 \newcommand\omgroup@redefine@addtocontents[1]{%
2021 %\edef\@import{#1}%
2022 %\@for\@I:=\@import\do{%
2023 %\edef\@path{\csname module@\@I @path\endcsname}%
2024 %\@ifundefined{tf@toc}\relax%
2025 %    {\protected@write\tf@toc}{\string\@requiremodules{\@path}}}%
2026 %\ifx\hyper@anchor\undefined% hyperref.sty loaded?
2027 %\def\addcontentsline##1##2##3{%
2028 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{#1}{##3}}{\thepage}}%
2029 %\else% hyperref.sty not loaded
2030 %\def\addcontentsline##1##2##3{%
2031 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{#1}{##3}}{\thepage}}{\@c
2032 %\fi
2033 }% hyperref.sty loaded?
```

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

```
2034 \newcount\omgroup@level
2035 \newenvironment{omgroup}[2] [] % keys, title
2036 {\metasetkeys{omgroup}{#1}\sref@target%
2037 \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.

```
2038 \ifx\omgroup@loadmodules\@true%
2039 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
2040 {\@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`.

```

2041 \advance\section@level by 1\relax%
2042 \ifcase\section@level%
2043 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
2044 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
2045 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
2046 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
2047 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%
2048 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{#2}%
2049 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{#2}%
2050 \fi% \ifcase
2051 \at@begin@omgroup[#1]\section@level{#2}}% for customization
2052 {\advance\section@level by -1\advance\omgroup@level by -1}

```

and finally, we localize the sections

```

2053 \newcommand\omdoc@part@kw{Part}
2054 \newcommand\omdoc@chapter@kw{Chapter}
2055 \newcommand\omdoc@section@kw{Section}
2056 \newcommand\omdoc@subsection@kw{Subsection}
2057 \newcommand\omdoc@subsubsection@kw{Subsubsection}
2058 \newcommand\omdoc@paragraph@kw{paragraph}
2059 \newcommand\omdoc@subparagraph@kw{subparagraph}

```

`\setSGvar` set a global variable

```

2060 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}

```

`\useSGvar` use a global variable

```

2061 \newrobustcmd\useSGvar[1]{%
2062   \ifundefined{sTeX@Gvar@#1}
2063   {\PackageError{omdoc}
2064     {The sTeX Global variable #1 is undefined}
2065     {set it with \protect\setSGvar}}
2066 \@nameuse{sTeX@Gvar@#1}}

```

`blindomgroup`

```

2067 \newcommand\at@begin@blindomgroup[1]{%
2068 \newenvironment{blindomgroup}
2069 {\advance\section@level by 1\at@begin@blindomgroup\setion@level}
2070 {\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).

```

2071 \srefaddidkey{omtext}
2072 \addmetakey[] {omtext}{functions}
2073 \addmetakey*{omtext}{display}
2074 \addmetakey{omtext}{for}
2075 \addmetakey{omtext}{from}
2076 \addmetakey{omtext}{type}
2077 \addmetakey*{omtext}{title}
2078 \addmetakey*{omtext}{start}
2079 \addmetakey{omtext}{theory}
2080 \addmetakey{omtext}{continues}
2081 \addmetakey{omtext}{verbalizes}
2082 \addmetakey{omtext}{subject}

\st@flow We define this macro, so that we can test whether the display key has the value
flow
2083 \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.

2084 \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.

2085 \def\omtext@pre@skip{\smallskip}
2086 \def\omtext@post@skip{}
2087 \newenvironment{omtext}[1] [] {\@in@omtexttrue%
2088 \bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
2089 \def\lec##1{\@lec{##1}}%
2090 \omtext@pre@skip\par\noindent%
2091 \ifx\omtext@title\@empty%
2092 \ifx\omtext@start\@empty\else%
2093 \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
2094 \fi% end omtext@start empty
2095 \else\stDMemph{\omtext@title}:\enspace%
2096 \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
2097 \fi% end omtext@title empty
2098 %\ignorespacesandpars
2099 }
2100 {\egroup\omtext@post@skip\@in@omtextfalse%\ignorespacesandpars
2101 }

```

3.13.2 Phrase-level Markup

```

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

2102 \srefaddidkey{phrase}
2103 \addmetakey{phrase}{style}
2104 \addmetakey{phrase}{class}
2105 \addmetakey{phrase}{index}

```

```

2106 \addmetakey{phrase}{verbalizes}
2107 \addmetakey{phrase}{type}
2108 \addmetakey{phrase}{only}
2109 \newcommand\phrase[2] [] {\metasetkeys{phrase}{#1}%
2110 \ifx\prhaseonly\empty\only<\phraseonly>{#2}\else #2\fi}

\coref*
2111 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}
2112 \newcommand\corefs[2]{#1\textsubscript{#2}}
2113 \newcommand\coreft[2]{#1\textsuperscript{#2}}

\n*lex
2114 \newcommand\nlex[1]{\green{\sl{#1}}}
2115 \newcommand\nlcex[1]{*\green{\sl{#1}}}

sinlinequote
2116 \def\@sinlinequote#1{‘‘\sl{#1}}’}
2117 \def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
2118 \newcommand\sinlinequote[2] []
2119 {\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.

```

2120 \newcommand\vdec[2] [] {#2}
2121 \newcommand\vest[2] [] {#2}
2122 \newcommand\vcond[2] [] {#2}

```

EdN:1	<pre> \strucdec 1 2123 \newcommand\strucdec[2] [] {#2} </pre>
EdN:2	<pre> \impdec 2 2124 \newcommand\impdec[2] [] {#2} </pre>

3.13.4 Block-Level Markup

```

sblockquote
2125 \def\begin@sblockquote{\begin{quote}\sl}
2126 \def\end@sblockquote{\end{quote}}
2127 \def\begin@@sblockquote#1{\begin@sblockquote}
2128 \def\end@@sblockquote#1{\def\@lec##1{\textrm{##1}}\@lec{#1}\end@sblockquote}
2129 \newenvironment{sblockquote}[1] []
2130 {\def\@opt{#1}\ifx\@opt\empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
2131 {\ifx\@opt\empty\end@sblockquote\else\end@@sblockquote\@opt\fi}

```

¹EdNOTE: document above

²EdNOTE: document above

sboxquote

```
2132 \newenvironment{sboxquote}[1] []
2133 {\def\@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
2134 {\@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.

```
2135 \providecommand{\@lec}[1]{( #1 )}
2136 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@lec{#1}}
2137 \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.

```
2138 \addmetakey{omdoc@index}{at}
2139 \addmetakey[false]{omdoc@index}{loadmodules}[true]
2140 \newcommand\omdoc@indexi[2] [] {\ifindex%
2141 \metasetkeys{omdoc@index}{#1}%
2142 \@bsphack\begingroup\@sanitize%
2143 \protected@write\@indexfile{\string\indexentry%
2144 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
2145 \ifx\omdoc@index@loadmodules\@true%
2146 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
2147 \else #2\fi% loadmodules
2148 }\thepage}}%
2149 \endgroup\@esphack\fi}%ifindex
2150 \newcommand\omdoc@indexii[3] [] {\ifindex%
2151 \metasetkeys{omdoc@index}{#1}%
2152 \@bsphack\begingroup\@sanitize%
2153 \protected@write\@indexfile{\string\indexentry%
2154 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
2155 \ifx\omdoc@index@loadmodules\@true%
2156 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
2157 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
2158 \else #2!#3\fi% loadmodules
2159 }\thepage}}%
2160 \endgroup\@esphack\fi}%ifindex
2161 \newcommand\omdoc@indexiii[4] [] {\ifindex%
2162 \metasetkeys{omdoc@index}{#1}%
2163 \@bsphack\begingroup\@sanitize%
2164 \protected@write\@indexfile{\string\indexentry%
2165 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
2166 \ifx\omdoc@index@loadmodules\@true%
2167 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
2168 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
2169 \else #2!#3\fi% loadmodules
2170 }\thepage}}%
2171 \endgroup\@esphack\fi}%ifindex
```

```

2163 \@bsphack\beginngroup\@sanitize%
2164 \protected@write\@indexfile{}\string\indexentry%
2165 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
2166 \ifx\omdoc@index@loadmodules\@true%
2167 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
2168 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
2169 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
2170 \else #2!#3!#4\fi% loadmodules
2171 }\the page}}%
2172 \endgroup\@esphack\fi}%ifindex
2173 \newcommand\omdoc@indexiv[5][\ifindex%
2174 \metasetkeys{omdoc@index}{#1}%
2175 \@bsphack\beginngroup\@sanitize%
2176 \protected@write\@indexfile{}\string\indexentry%
2177 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
2178 \ifx\omdoc@index@loadmodules\@true%
2179 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
2180 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
2181 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
2182 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
2183 \else #2!#3!#4!#5\fi% loadmodules
2184 }\the page}}%
2185 \endgroup\@esphack\fi}%ifindex

```

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

indi

```

2186 \newcommand\aindi[3][\{#2\}\omdoc@indexi[#1]{#3}}
2187 \newcommand\indi[2][\{#2\}\omdoc@indexi[#1]{#2}}
2188 \newcommand\indis[2][\{#2\}\omdoc@indexi[#1]{#2s}}
2189 \newcommand\Indi[2][\{\captitalize{#2}\}\omdoc@indexi[#1]{#2}}
2190 \newcommand\Indis[2][\{\capitalize{#2}\}\omdoc@indexi[#1]{#2s}}
2191
2192 \newcommand\@indii[3][\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
2193 \newcommand\aindii[4][\{#2\}\@indii[#1]{#3}{#4}}
2194 \newcommand\indii[3][\{#2 #3\}\@indii[#1]{#2}{#3}}
2195 \newcommand\indiis[3][\{#2 #3s\}\@indii[#1]{#2}{#3}}
2196 \newcommand\Indii[3][\{\captitalize{#2 #3}\}\@indii[#1]{#2}{#3}}
2197 \newcommand\Indiis[3][\{\capitalize{#2 #3}\}\@indii[#1]{#2}{#3}}
2198
2199 \newcommand\@indiii[4][\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexiii[#1]{#3}{#2 (#4)}}
2200 \newcommand\aindiii[5][\{#2\}\@indiii[#1]{#3}{#4}{#5}}
2201 \newcommand\indiii[4][\{#2 #3 #4\}\@indiii[#1]{#2}{#3}{#4}}
2202 \newcommand\indiis[4][\{#2 #3 #4s\}\@indiii[#1]{#2}{#3}{#4}}
2203 \newcommand\Indiii[4][\{\captitalize{#2 #3 #4}\}\@indiii[#1]{#2}{#3}{#4}}
2204 \newcommand\Indiis[4][\{\capitalize{#2 #3 #4s}\}\@indiii[#1]{#2}{#3}{#4}}
2205
2206 \newcommand\@indiv[5][\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
2207 \newcommand\aindiv[6][\{#2\}\@indiv[#1]{#3}{#4}{#5}{#6}}
2208 \newcommand\indiv[5][\{#2 #3 #4 #5\}\@indiv[#1]{#2}{#3}{#4}{#5}}

```

```

2209 \newcommand\indivs[5] []{{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
2210 \newcommand\Indiv[5] []{\capitalizе{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
2211 \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^AT_EX_ML.

```

2212 \newcommand\hateq{\ensuremath{\widehat{=}}\xspace}
2213 \newcommand\hatequiv{\ensuremath{\widehat{equiv}}\xspace}
2214 \@ifundefined{ergo}%
2215 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
2216 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
2217 \newcommand{\reflect@squig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%
2218 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
2219 \newcommand\notergo{\ensuremath{\not\leadsto}}
2220 \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*`

```

2221 \newcommand\indextoo[2] []{\indi[#1]{#2}}%
2222 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}%
2223 \newcommand\indexalt[2] []{\aindi[#1]{#2}}%
2224 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead}%
2225 \newcommand\twintoo[3] []{\indii[#1]{#2}{#3}}%
2226 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}%
2227 \newcommand\twinalt[3] []{\aindii[#1]{#2}{#3}}%
2228 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead}%
2229 \newcommand\atwintoo[4] []{\indiii[#1]{#2}{#3}{#4}}%
2230 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead}%
2231 \newcommand\atwinalt[4] []{\aindii[#1]{#2}{#3}{#4}}%
2232 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instead}%

```

`\my*graphics`

```

2233 \newcommand\mygraphics[2] []{\includegraphics[#1]{#2}}%
2234 \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics}%
2235 \newcommand\mycgraphics[2] []{\begin{center}\mygraphics[#1]{#2}\end{center}}%
2236 \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics}%
2237 \newcommand\mybgraphics[2] []{\fbox{\mygraphics[#1]{#2}}}%
2238 \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics}%
2239 \newcommand\mycbgraphics[2] []{\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}%
2240 \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphics}%

```

4 Things to deprecate

Module options:

```

2241 \addmetakey*{module}{id} % TODO: deprecate properly
2242 \addmetakey*{module}{load}
2243 \addmetakey*{module}{path}
2244 \addmetakey*{module}{dir}
2245 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
2246 \addmetakey*{module}{noalign}[true]
2247
2248 \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^AT_EX part.

```

2249 %\srefaddidkey{symdef}% what does this do?
2250 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
2251 \define@key{symdef}{noverb}[all]{}%
2252 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
2253 \define@key{symdef}{specializes}{}%
2254 \addmetakey*{symdef}{noalign}[true]
2255 \define@key{symdef}{primary}[true]{}%
2256 \define@key{symdef}{assocarg}{}%
2257 \define@key{symdef}{bvars}{}%
2258 \define@key{symdef}{bargs}{}%
2259 \addmetakey{symdef}{lang}%
2260 \addmetakey{symdef}{prec}%
2261 \addmetakey{symdef}{arity}%
2262 \addmetakey{symdef}{variant}%
2263 \addmetakey{symdef}{ns}%
2264 \addmetakey{symdef}{args}%
2265 \addmetakey{symdef}{name}%
2266 \addmetakey*{symdef}{title}%
2267 \addmetakey*{symdef}{description}%
2268 \addmetakey{symdef}{subject}%
2269 \addmetakey*{symdef}{display}%
2270 \addmetakey*{symdef}{gfc}%

```

EdN:3

3

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

```

2271 \def\symdef{\@ifnextchar[{\@symdef}{\@symdef []}}%
2272 \def\@symdef[#1]#2{\@ifnextchar[{\@@symdef[#1]{#2}}{\@@symdef[#1]{#2}[0]}}%

```

³EDNOTE: 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.

```

2273 \def\@@symdef[#1]#2[#3]{%
2274   \@insymdef@true%
2275   \metasetkeys{symdef}{#1}%
2276   \edef\symdef@tmp@optpars{\ifcvoid{symdef@name}{[]}{[name=\symdef@name]}}%
2277   \expandafter\symdecl\symdef@tmp@optpars{#2}%
2278   \@insymdef@false%
2279   \notation[#1]{#2}[#3]%
2280 }% mod@show
2281 \def\symdef@type{Symbol}%
2282 \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`.

```

2283 \def\symvariant#1{%
2284   \ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}%
2285   }%
2286 \def\@symvariant#1[#2]#3#4{%
2287   \notation[#3]{#1}[#2]{#4}%
2288 %\ignorespacesandpars
2289 }%

```

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

```

2290 \newif\if@importing\@importingfalse
2291 \define@key{symi}{noverb}[all]{}%
2292 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
2293 \define@key{symi}{specializes}{}%
2294 \define@key{symi}{gfc}{}%
2295 \define@key{symi}{noalign}[true]{}%
2296 \newcommand\symi{\@ifstar\@symi@star\symi}
2297 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
2298   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces
2299   }
2300 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
2301   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignorespaces
2302   }
2303 \newcommand\symii{\@ifstar\@symii@star\symii}
2304 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
2305   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespaces
2306   }
2307 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
2308   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\ignorespaces
2309   }
2310 \newcommand\symiii{\@ifstar\@symiii@star\symiii}
2311 \newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%

```



```

2312 \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi%\ignorespacesandpars%
2313 }
2314 \newcommand\@symiii@star[4][\metasetkeys{symi}{#1}%
2315 \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi%\ignorespacesandpars%
2316 }
2317 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
2318 \newcommand\@symiv[5][\metasetkeys{symi}{#1}%
2319 \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi%\ignorespacesandpars%
2320 }
2321 \newcommand\@symiv@star[5][\metasetkeys{symi}{#1}%
2322 \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4-#5}\fi%\ignorespacesandpars%
2323 }

```

`\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`.

```

2324 %\srefaddidkey{importmhmodule}%
2325 \addmetakey{importmhmodule}{mhrepos}%
2326 \addmetakey{importmhmodule}{path}%
2327 \addmetakey{importmhmodule}{ext}% why does this exist?
2328 \addmetakey{importmhmodule}{dir}%
2329 \addmetakey[false]{importmhmodule}{conservative}[true]%
2330 \newcommand\importmhmodule[2][\%
2331 \parsemodule@maybesetcodes
2332 \metasetkeys{importmhmodule}{#1}%
2333 \ifx\importmhmodule@dir\@empty%
2334 \edef\@path{\importmhmodule@path}%
2335 \else\edef\@path{\importmhmodule@dir/#2}\fi%
2336 \ifx\@path\@empty% if module name is not set
2337 \@importmodule[] {#2}{export}%
2338 \else%
2339 \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
2340 \ifx\importmhmodule@mhrepos\@empty% if in the same repos
2341 \relax% no need to change mh@currentrepos, i.e., current directory.
2342 \else%
2343 \mathhub@setcurrentreposinfo\importmhmodule@mhrepos% change it.
2344 \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}}%
2345 \fi%
2346 \@importmodule[\MathHub{\mh@currentrepos/source/\@path}] {#2}{export}%
2347 \mathhub@setcurrentreposinfo\mh@@repos% after importing, reset to old value
2348 \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\mh@@repos}}%
2349 \fi%
2350 %\ignorespacesandpars%
2351 }

```

`\usemhmodule`

```

2352 \addmetakey{importmhmodule}{load}
2353 \addmetakey{importmhmodule}{id}
2354 \addmetakey{importmhmodule}{dir}
2355 \addmetakey{importmhmodule}{mhrepos}
2356
2357 \addmetakey{importmodule}{load}
2358 \addmetakey{importmodule}{id}
2359
2360 \newcommand\usemhmodule[2][]{%
2361 \metasetkeys{importmhmodule}{#1}%
2362 \ifx\importmhmodule@dir\empty%
2363 \edef\@path{\importmhmodule@path}%
2364 \else\edef\@path{\importmhmodule@dir/#2}\fi%
2365 \ifx\@path\empty%
2366 \usemodule[id=\importmhmodule@id]{#2}%
2367 \else%
2368 \edef\mh@@repos{\mh@currentrepos}%
2369 \ifx\importmhmodule@mhrepos\empty%
2370 \else\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
2371 \usemodule{\@path\@QuestionMark#2}%
2372 %\usemodule[load=\MathHub{\mh@currentrepos/source/\@path},
2373 % id=\importmhmodule@id]{#2}%
2374 \mathhub@setcurrentreposinfo\mh@@repos%
2375 \fi%
2376 %\ignorespacesandpars
2377 }

\mhinputref
2378 \newcommand\mhinputref[2][]{%
2379 \edef\mhinputref@first{#1}%
2380 \ifx\mhinputref@first\empty%
2381 \inputref{#2}%
2382 \else%
2383 \inputref[mhrepos=\mhinputref@first]{#2}%
2384 \fi%
2385 }

\trefi*
2386 \newcommand\trefi[2][]{%
2387 \edef\trefi@mod{#1}%
2388 \ifx\trefi@mod\empty\tref{#2}\else\tref{#1\@QuestionMark#2}\fi%
2389 }
2390 \newcommand\trefii[3][]{%
2391 \edef\trefi@mod{#1}%
2392 \ifx\trefi@mod\empty\tref{#2-#3}\else\tref{#1\@QuestionMark#2-#3}\fi%
2393 }

\defi*
2394 \def\defii#1#2{\defi{#1!#2}}

```

```

2395 \def\Defii#1#2{\Defi{#1!#2}}
2396 \def\defiis#1#2{\defis{#1!#2}}
2397 \def\Defiis#1#2{\Defis{#1!#2}}
2398 \def\defiii#1#2#3{\defi{#1!#2!#3}}
2399 \def\Defiii#1#2#3{\Defi{#1!#2!#3}}
2400 \def\defiis#1#2#3{\defis{#1!#2!#3}}
2401 \def\Defiis#1#2#3{\Defis{#1!#2!#3}}
2402 \def\defiv#1#2#3#4{\defi{#1!#2!#3!#4}}
2403 \def\Defiv#1#2#3#4{\Defi{#1!#2!#3!#4}}
2404 \def\defivs#1#2#3#4{\defis{#1!#2!#3!#4}}
2405 \def\Defivs#1#2#3#4{\Defis{#1!#2!#3!#4}}
2406 \def\adefi#1#2{\defi[name=#2]{#1}}
2407 \def\adefii#1#2#3{\defi[name=#2-#3]{#1}}
2408 \def\adefiii#1#2#3#4{\defi[name=#2-#3-#4]{#1}}
2409 \def\adefiv#1#2#3#4#5{\defi[name=#2-#3-#4-#5]{#1}}

```