

stex.sty: \TeX 2.0*

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

March 19, 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	21
3.5	Symbols/Notations/Verbalizations	32
3.6	Term References	44
3.7	sref	46
3.8	smultiling	49
3.9	smglom	49
3.10	mathhub	50
3.11	omdoc/omgroup	50
3.12	omtext	53
4	Things to deprecate	58

1 Introduction

TODO

2 User commands

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

3 Implementation

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

A conditional for LaTeXML:

```

19 \ifcsname if@latexml\endcsname\else
20 \ex\newif\csname if@latexml\endcsname\@latexmlfalse
21 \fi

```

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

```

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

```

3.1 sTeX base

The sTeX logo:

```

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

```

3.2 Paths and URIs

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

```

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

```

`\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`.

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

3.2.1 Path Canonicalization

We define some macros for later comparison.

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

```

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

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

```

```

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

```

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.

```

162 \newcommand\cpath@print[1]{%
163     \@cpath{#1}%
164     \@cpath@path%
165 }

```

\path@filename

```

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

```

```

168 \StrCount\filename@oldpath\@Slash[\filename@lastslash]%
169 \ifnum\filename@lastslash>0%
170 \StrBehind[\filename@lastslash]\filename@oldpath%
171 \@Slash[\filename@oldpath]%
172 \edef#2{\filename@oldpath}%
173 \else%
174 \edef#2{\filename@oldpath}%
175 \fi%
176 }

```

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

\path@filename@noext

```

177 \def\path@filename@noext#1#2{%
178 \path@filename{#1}{#2}%
179 \edef\filename@oldpath{#2}%
180 \StrCount\filename@oldpath\@Dot[\filename@lastdot]%
181 \ifnum\filename@lastdot>0%
182 \StrBefore[\filename@lastdot]\filename@oldpath%
183 \@Dot[\filename@oldpath]%
184 \edef#2{\filename@oldpath}%
185 \else%
186 \edef#2{\filename@oldpath}%
187 \fi%
188 }

```

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:

```

189 \newif\if@iswindows@\@iswindows@false
190 \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:

```

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

```



```

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

```

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:

```

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

```

```

242     \ifx#1/%
243         \edef\windows@temp{\windows@temp\@BackSlash}%
244     \else%
245         \edef\windows@temp{\windows@temp#1}%
246     \fi%
247 \else%
248     \ifx#1/%
249         \edef\windows@temp{\windows@temp:\@BackSlash}%
250         \@windowstopath@inpath@true%
251     \else%
252         \edef\windows@temp{\windows@temp#1}%
253     \fi%
254 \fi%
255 \windows@continue%
256 }

```

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:

```

257 \def\path@trimstring#1{%
258     \edef\pathsuris@trim@temp{#1}%
259     \IfBeginWith\pathsuris@trim@temp\@Space{%
260         \StrGobbleLeft\pathsuris@trim@temp1[#1]%
261         \path@trimstring{#1}%
262     }{%
263         \IfEndWith\pathsuris@trim@temp\@Space{%
264             \StrGobbleRight\pathsuris@trim@temp1[#1]%
265             \path@trimstring{#1}%
266         }{%
267             \edef#1{\pathsuris@trim@temp}%
268         }%
269     }%
270 }

```

Test 7: >foo bar<

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

```

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

```

```

280 \endgroup}
281 %\fi

```

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

3.2.4 sTeX input hooks

We determine the PWD of the current main document:

```

282 \edef\pwd@cmd{\if@iswindows@ -expand-var \@Percent%
283   CD\@Percent\else -var-value PWD\fi}
284 \@kpsewhich\stex@PWD\pwd@cmd
285 \edef\stex@mainfile{\stex@PWD\@Slash\jobname}
286 \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:

```

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

```

```

319     \global\let\stex@currfilename\stex@currfilename%
320     \fi%
321 }

```

\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:

```

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

```

339 \@kpsewhich\mathhub@path{--var-value MATHHUB}
340 \if@iswindows@\windows@to@path\mathhub@path\fi
341 \ifx\mathhub@path\@empty
342     \PackageWarning{stex}{MATHHUB system variable not %
343         found or wrongly set}{%
344         \defpath{MathHub}{%
345         \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.

```

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

```

```

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

```

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

```

375 \def\split@manifest@key{%
376     \IfSubStr{\manifest@line}{\@Colon}{%
377         \StrBefore{\manifest@line}{\@Colon}[\manifest@key]%
378         \StrBehind{\manifest@line}{\@Colon}[\manifest@line]%
379         \path@trimstring\manifest@line%
380         \path@trimstring\manifest@key%
381     }{%
382         \def\manifest@key{}%
383     }%
384 }

```

the next helper function iterates over lines in MANIFEST.MF

```

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

```

```

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

```

`\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.

```

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

```

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.
437 \def\mathhub@setcurrentreposinfo#1{%
438   \edef\mh@currentrepos{#1}%
439   \ifx\mh@currentrepos\empty%
440     \edef\currentrepos@dir{\@Dot}%
441     \def\currentrepos@narr{%
442       \def\currentrepos@ns{%
443         \def\currentrepos@id{%
444           \def\currentrepos@deps{%
445             \else%
446             \ifcsdef{mathhub@dir@\mh@currentrepos}{%
447               \@inmhrepostrue
448               \ex\let\ex\currentrepos@dir\csname mathhub@dir@#1\endcsname%
449               \ex\let\ex\currentrepos@narr\csname mathhub@narr@#1\endcsname%
450               \ex\let\ex\currentrepos@ns\csname mathhub@ns@#1\endcsname%
451               \ex\let\ex\currentrepos@deps\csname mathhub@deps@#1\endcsname%
452             }{%
453               \mathhub@parsemanifest{currentrepos@}\MathHub{#1}}%
454             \@setcurrentreposinfo%
455             \ifcsvoid{currentrepos@dir}\PackageError{stex}{No archive with %
456               name #1 found!}{make sure that #1 is directly in your MATHHUB folder %
457               and contains a MANIFEST.MF, either directly in #1 or in a meta-inf %
458               subfolder.}}{\@inmhrepostrue}%
459           }%
460         \fi%
461       }
462     }
463   \def\@setcurrentreposinfo{%
464     \edef\mh@currentrepos{\currentrepos@id}%
465     \ifcsvoid{currentrepos@dir}{%
466       \csxdef{mathhub@dir@\currentrepos@id}{\currentrepos@dir}%
467       \csxdef{mathhub@narr@\currentrepos@id}{\currentrepos@narr}%
468       \csxdef{mathhub@ns@\currentrepos@id}{\currentrepos@ns}%
469       \csxdef{mathhub@deps@\currentrepos@id}{\currentrepos@deps}%
470     }%
471   }
```

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

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

```

477 \message{Current sTeX repository: \mh@currentrepos}
478 }
479 }

```

3.3 Modules

```

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

Aux:
481 %\def\ignorespacesandpars{\begingroup\catcode13=10%
482 % \ifnextchar\relax{\endgroup}{\endgroup}}

and more adapted from http://tex.stackexchange.com/questions/179016/ignore-spaces-and-pars-after-an-environment
483 %\def\ignorespacesandparsafterend#1\ignorespaces\fi{#1%
484 % \fi\ignorespacesandpars}
485 %\def\ignorespacesandpars{\ifhmode\unskip\fi\ifnextchar\par%
486 % {\ex\ignorespacesandpars\@gobble}{}}

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

module@heading We make a convenience macro for the module heading. This can be customized.
494 \ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%
495 \newrobustcmd{module@heading}{%
496 \stepcounter{module}%
497 \ifmod@show%
498 \noindent{\textbf{Module}} \thesection.\thetitle [\module@name]}%
499 \sref@label{id{Module} \thesection.\thetitle [\module@name]}%
500 \ifx\module@title@empty :\quad\else\quad(\module@title)\hfill\\\fi%
501 \fi%
502 }%

```

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.

```

503 \newenvironment{module}[1][]{%
504 \begin{@module}[#1]%
505 \module@heading% make the headings
506 %\ignorespacesandpars
507 \parsemodule@maybesetcodes}{%
508 \end{@module}}%
509 \ignorespacesafterend%
510 }%

```



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

Some auxiliary methods:

```
512 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}
513 \def\addto@thismodule#1{%
514   \ifundefined{this@module}{}{%
515     \expandafter\g@addto@macro@safe\this@module{#1}%
516   }%
517 }
518 \def\addto@thismodulex#1{%
519   \ifundefined{this@module}{}{%
520     \edef\addto@thismodule@exp{#1}%
521     \expandafter\expandafter\expandafter\g@addto@macro@safe%
522     \expandafter\this@module\expandafter{\addto@thismodule@exp}%
523   }}
```

@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`.

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

```

546 \fi%
547 \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]}%
548 \setkeys{module}{ns=\@module@ns@tempuri}%
549 }

```

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.

```

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

```

Test 16: `stex`

`stex.1`

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

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

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

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

```

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

```

```

612 }{%
613   \end{latexml@annotateenv}%
614   \if@inmhrepos%
615   \@inmhreposfalse%
616   \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\expandafter\noexpand\csname mh@old@
617   \fi%
618 }%
619 \newenvironment{@structural@feature}[2]{%
620   \ifcvoid{currentmodule@uri}{%
621     \set@default@ns\let\currentmodule@ns\module@ns%
622     \set@next@moduleid\let\currentmodule@name\module@name%
623   }{%}%
624   \edef\currentmodule@name{\currentmodule@name\@Slash#2}%
625   \edef\currentmodule@uri{\currentmodule@ns\@QuestionMark\currentmodule@name}%
626   \parsemodule@maybesetcodes%
627   \begin{latexml@annotateenv}{feature:#1}{\currentmodule@uri}%
628 }{%
629   \end{latexml@annotateenv}%
630 }%

```

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 $\langle @invoke@module\langle uri \rangle \rangle @URI$, which expands to

the full uri of a module (i.e. via `\stex@module@<id>\@URI`). In the future, this macro can be extended with additional functionality, e.g. accessing symbols in a macro for overloaded (macro-)names.

```

631 \def\@URI{uri} % TODO check this
632 \def\@invoke@module#1#2{%
633   \ifx\@URI#2%
634     #1%
635   \else%
636     % TODO something else
637     #2%
638   \fi%
639 }

```

3.4 Inheritance

3.4.1 Selective Inclusion

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

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

```

\parsemodule@allow* The first step is setting up a functionality for registering \TeX macros and envi-
                    ronments as part of a module signature.
640 \newif\if@smsmode\@smsmodefalse
641 \def\parsemodule@allow#1{%
642   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#1\endcsname}%
643 }
644 \def\parsemodule@allowenv#1{%
645   \ex\def\csname parsemodule@allowedenv@#1\endcsname{\csname#1}%
646 }
647 \def\parsemodule@replacemacro#1#2{%
648   \ex\def\csname parsemodule@allowedmacro@#1\ex\endcsname\ex{\csname#2\endcsname}%
649 }
650 \def\parsemodule@replaceenv#1#2{%
651   \ex\def\csname parsemodule@allowedenv@#1\endcsname{\csname#2}%
652 }
653 \def\parsemodule@escapechar@beginstring{begin}
654 \def\parsemodule@escapechar@endstring{end}

```

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

```

655 \parsemodule@allow{symdef}
656 \parsemodule@allow{abbrdef}
657 \parsemodule@allow{importmodule}

```

```

658 \parsemodule@allowenv{module}
659 \parsemodule@allowenv{@module}
660 \parsemodule@allow{importmhmodule}
661 \parsemodule@allow{gimport}
662 \parsemodule@allowenv{modsig}
663 \parsemodule@allowenv{mhmodsig}
664 \parsemodule@allowenv{mhmodnl}
665 \parsemodule@allowenv{modnl}
666 \parsemodule@allowenv{@structural@feature}
667 \parsemodule@allow{symvariant}
668 \parsemodule@allow{symi}
669 \parsemodule@allow{symii}
670 \parsemodule@allow{symiii}
671 \parsemodule@allow{symiv}
672 \parsemodule@allow{notation}
673 \parsemodule@allow{symdecl}
674
675 % to deprecate:
676
677 \parsemodule@allow{defi}
678 \parsemodule@allow{defii}
679 \parsemodule@allow{defiii}
680 \parsemodule@allow{defiv}
681 \parsemodule@allow{adefi}
682 \parsemodule@allow{adefii}
683 \parsemodule@allow{adefiii}
684 \parsemodule@allow{adefiv}
685 \parsemodule@allow{defis}
686 \parsemodule@allow{defiis}
687 \parsemodule@allow{defiiis}
688 \parsemodule@allow{defivs}
689 \parsemodule@allow{Defi}
690 \parsemodule@allow{Defii}
691 \parsemodule@allow{Defiii}
692 \parsemodule@allow{Defiv}
693 \parsemodule@allow{Defis}
694 \parsemodule@allow{Defiis}
695 \parsemodule@allow{Defiiis}
696 \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*).

```

697 \catcode'\.=0

```

```

698 .catcode'\=13
699 .def.@active@slash{\}
700 .catcode'.<=1
701 .catcode'.>=2
702 .catcode'\'=12
703 .catcode'\'=12
704 .def.@open@brace{<{>
705 .def.@close@brace{>}
706 .catcode'\=0
707 \catcode'\.=12
708 \catcode'\{=1
709 \catcode'\}=2
710 \catcode'\<=12
711 \catcode'\>=12

```

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

`\set@parsemodule@catcodes`

```

712 \def\set@parsemodule@catcodes{,inputenc,}
713 \let\parsemodule@old@PackageError\PackageError
714 \def\parsemodule@packageerror#1#2#3{%
715   \IfSubStr\parsemodule@ignorepackageerrors{,#1,}{-}{%
716     \parsemodule@old@PackageError{#1}{#2}{#3}%
717   }%
718 }
719 \def\set@parsemodule@catcodes{%
720   \ifcat'\=0%
721     \global\catcode'\=13%
722     \global\catcode'\#=12%
723     \global\catcode'\{=12%
724     \global\catcode'\}=12%
725     \global\catcode'\$=12%$
726     \global\catcode'\^=12%
727     \global\catcode'\_ =12%
728     \global\catcode'\&=12%
729     \ex\global\ex\let\@active@slash\parsemodule@escapechar%
730     \global\let\parsemodule@old@PackageError\PackageError%
731     \global\let\PackageError\parsemodule@packageerror%
732     \fi%
733 }

```

`\reset@parsemodule@catcodes`

```

734 \def\reset@parsemodule@catcodes{%
735   \ifcat'\=13%
736     \global\catcode'\=0%
737     \global\catcode'\#=6%
738     \global\catcode'\{=1%
739     \global\catcode'\}=2%
740     \global\catcode'\$=3%$
741     \global\catcode'\^=7%

```

```

742     \global\catcode'\_ =8%
743     \global\catcode'\&=4%
744     \global\let\PackageError\parsemodule@old@PackageError%
745     \fi%
746 }

```

`\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.

```

747 \def\parsemodule@maybesetcodes{%
748   \if@smsmode\set@parsemodule@catcodes\fi%
749 }

```

`\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:

```

750
751 \def\parsemodule@escapechar{%
752   \def\parsemodule@escape@currcls{}%
753   \parsemodule@escape@parse@nextchar@%
754 }%

```

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`.

```

755 \long\def\parsemodule@escape@parse@nextchar@#1{%
756   \ifcat a#1\relax%
757     \edef\parsemodule@escape@currcls{\parsemodule@escape@currcls#1}%
758     \let\parsemodule@do@next\parsemodule@escape@parse@nextchar@%
759   \else%
760     \def\parsemodule@last@char{#1}%
761     \ifx\parsemodule@escape@currcls\@empty%
762       \def\parsemodule@do@next{}%
763     \else%
764       \def\parsemodule@do@next{\parsemodule@escapechar@checkcls}%
765     \fi%
766   \fi%
767   \parsemodule@do@next%
768 }

```


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.

```

769 \def\parsemodule@escapechar@checkcs{%
770   \ifx\parsemodule@escape@currcc\parsemodule@escapechar@beginstring%
771     \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@last@char}%
772   \else%
773     \ifx\parsemodule@escape@currcc\parsemodule@escapechar@endstring%
774       \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@last@char}%
775     \else%
776       \ifcvoid{parsemodule@allowedmacro@\parsemodule@escape@currcc}{%
777         \def\parsemodule@do@next{\relax\parsemodule@last@char}%
778       }{%
779         \ifx\parsemodule@last@char\@open@brace%
780           \ex\let\ex\parsemodule@do@next@ii\csname parsemodule@allowedmacro@\parsemodule@escape@currcc\endcsname%
781         \edef\parsemodule@do@next{\noexpand\parsemodule@converttoproperbraces\@open@brace}%
782       \else%
783         \reset@parsemodule@catcodes%
784         \edef\parsemodule@do@next{\ex\noexpand\csname parsemodule@allowedmacro@\parsemodule@escape@currcc\endcsname}%
785       \fi%
786     }%
787   \fi%
788 \fi%
789 \parsemodule@do@next%
790 }

```

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.

```

791 \ex\ex\ex\def%
792 \ex\ex\ex\parsemodule@converttoproperbraces%
793 \ex\@open@brace\ex#\ex1\@close@brace{%
794   \reset@parsemodule@catcodes%
795   \parsemodule@do@next@ii{#1}%
796 }

```

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.

```

797 \ex\ex\ex\def%
798 \ex\ex\ex\parsemodule@escapechar@checkbeginenv%
799 \ex\@open@brace\ex#\ex1\@close@brace{%
800   \ifcsvoid{parsemodule@allowedenv@#1}{%
801     \def\parsemodule@do@next{#1}%
802   }{%
803     \reset@parsemodule@catcodes%
804     \edef\parsemodule@envname{\csname parsemodule@allowedenv@#1\endcsname}%
805     \ex\def\ex\parsemodule@do@next\ex{%
806       \ex\begin\ex{\parsemodule@envname}%
807     }%
808   }%
809   \parsemodule@do@next%
810 }
811 \ex\ex\ex\def%
812 \ex\ex\ex\parsemodule@escapechar@checkendenv%
813 \ex\@open@brace\ex#\ex1\@close@brace{%
814   \ifcsvoid{parsemodule@allowedenv@#1}{%
815     \def\parsemodule@do@next{#1}%
816   }{%
817     \edef\parsemodule@envname{\csname parsemodule@allowedenv@#1\endcsname}%
818     \ex\def\ex\parsemodule@do@next\ex{%
819       \ex\end\ex{\parsemodule@envname}%
820     }%
821   }%
822   \parsemodule@do@next%
823 }

```

`\@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.

```

824 \newrobustcmd\@requiremodules[1]{%
825   \if@tempswa\requiremodules{#1}\fi%
826 }%

```

`\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.

```

827 \newrobustcmd\requiremodules[1]{%
828   \mod@showfalse%
829   \edef\mod@path{#1}%
830   \edef\mod@path{\expandafter\detokenize\expandafter{\mod@path}}%
831   \requiremodules@smsmode{#1}%
832 }%

```

`\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.

```

833 \newbox\modules@import@tempbox
834 \def\requiremodules@smsmode#1{%
835   \setbox\modules@import@tempbox\vbox{%
836     \@smsmodetrue%
837     \set@parsemodule@catcodes%
838     \hbadness=100000\relax%
839     \hfuzz=10000pt\relax%
840     \vbadness=100000\relax%
841     \vfuzz=10000pt\relax%
842     \stexinput{#1.tex}%
843     \reset@parsemodule@catcodes%
844   }%
845   \parsemodule@maybesetcodes%
846 }

```

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`

```

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

```

```

873 \else%
874 \ifx\importmodule@subdir\@empty\else%
875 \edef\importmodule@dir{\importmodule@dir\@Slash\importmodule@subdir}%
876 \fi%
877 \fi%
878 #3%
879 \if\importmodule@switchrepos%
880 \ex\mathhub@setcurrentreposinfo\csname importmodule@oldrepos@#2\endcsname%
881 \stex@debug{Importmodule: switched back to: \mh@currentrepos}%
882 \fi%
883 %\ignorespacesandpars%
884 }

```

\importmodule

```

885 %\srefaddidkey{importmodule}
886 \addmetakey{importmodule}{mhrepos}
887 \newcommand\importmodule[2][]{\@importmodule[#1]{#2}{export}}
888 \newcommand\@importmodule[3][]{%
889 \importmodule@bookkeeping{#1}{#2}{%
890 \importmodule[\importmodule@dir]\importmodule@module@name{#3}%
891 }%
892 }

```

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

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

```

893 \newcommand\@importmodule[3][]{%
894 {%
895 \edef\@load{#1}%
896 \edef\@importmodule@name{#2}%
897 \stex@debug{Loading #1}%
898 \if@smsmode\else\ifcsvoid{stex@module@\@importmodule@name}{% TODO check this
899 \stex@iffileexists\@load{
900 \stex@debug{Exists: #1}%
901 \requiremodules\@load}{%
902 \stex@debug{Does not exist: #1^^JTrying \@load\@Slash\@importmodule@name}%
903 \requiremodules{\@load\@Slash\@importmodule@name}%
904 }%
905 }{} \fi%
906 \ifx\@load\@empty\else%
907 {% TODO
908 % \edef\@path{\csname module@#2@path\endcsname}%
909 % \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do not
910 % {\PackageError{stex}% else signal an error

```

```

911 %      {Module Name Clash\MessageBreak%
912 %      A module with name #2 was already loaded under the path "@path"\MessageBreak%
913 %      The imported path "@load" is probably a different module with the\MessageBreak%
914 %      same name; this is dangerous -- not importing}%
915 %      {Check whether the Module name is correct}%
916 %      }%
917   }%
918 \fi%
919 \global\let\@importmodule@load\@load%
920 }%
921 \edef\@export{#3}\def\@export{export}%prepare comparison
922 %\ifx\@export\@export\export@defs{#2}\fi% export the module
923 \ifx\@export\@export\addto@thismodulex{%
924   \noexpand\@importmodule[\@importmodule@load]{#2}{noexport}%
925 }%
926 \if@smsmode\else
927 \ifcsvoid{this@module}{}{%
928   \ifcsvoid{module@imports@\module@uri}{
929     \csxdef{module@imports@\module@uri}{%
930       \csname stex@module@#2\endcsname\@URI% TODO check this
931     }%
932   }{%
933     \csxdef{module@imports@\module@uri}{%
934       \csname stex@module@#2\endcsname\@URI,% TODO check this
935       \csname module@imports@\module@uri\endcsname%
936     }%
937   }%
938 }%
939 \fi\fi%
940 \if@smsmode\else%
941   \edef\activate@module@name{#2}%
942   \StrCount\activate@module@name\@Slash[\activate@module@lastslash]%
943   \ifnum\activate@module@lastslash>0%
944     \StrCut[\activate@module@lastslash]\activate@module@name\@Slash\activate@module@temp\activa
945   \fi%
946   \ifcsvoid{stex@lastmodule@\activate@module@name}{%
947     \PackageError{stex}{No module with name \activate@module@name found}{}%
948   }{%
949     \ex\ex\ex\activate@defs\ex\ex\ex{\csname stex@lastmodule@\activate@module@name\endcsname}%
950   }%
951 \fi% activate the module
952 }%

```

Test 22: `\importmodule {testmoduleimporta}:`

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

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

Test 23: `\importmodule {testmoduleimportb?importb}:`

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

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

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

Default document module:

```
953 \AtBeginDocument{%
954   \set@default@ns%
955   \ifx\module@narr\@empty\setkeys{module}{narr=\module@ns}\fi%
956   \let\module@name\jobname%
957   \let\module@id\module@name % TODO deprecate
958   \edef\module@uri{\module@ns\@QuestionMark\module@name}%
959   \csgdef{module@names@\module@uri}{}%
960   \csgdef{module@imports@\module@uri}{}%
961   \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
962   \expandafter\global\expandafter\let\csname stex@module@\module@name\expandafter\endcsname\csname
963   \edef\this@module{%
964     \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
965   }%
966   \csdef{module@defs@\module@uri}{}%
967   \ifcvoid{\mh@currentrepos}{}%
968     \@inmhrepostrue%
969     \addto@thismodule{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\endcsname
970       {\noexpand\mh@currentrepos}}%
971     \addto@thismodule{\noexpand\mathhub@setcurrentreposinfo{\mh@currentrepos}}%
972   }%
973 }
```

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.

```
974 \newif\if@inimport\@inimportfalse
975 \def\latexml@import#1{\latexml@annotate@invisible{import}{#1}{\ }}%
```

```

976 \def\activate@defs#1{%
977   \stex@debug{Activating import #1}%
978   \if@inimport\else%
979     \latexml@import{#1}%
980     \def\inimport@module{#1}%
981     \stex@debug{Entering import #1}%
982     \@inimporttrue%
983   \fi%
984   \edef\activate@defs@uri{#1}%
985   \ifcsundef{module@defs@\activate@defs@uri}{%
986     \PackageError{stex}{No module with URI \activate@defs@uri loaded}{Probably missing an
987       \detokenize{\importmodule} (or variant) somewhere?
988     }
989   }{%
990     \ifcsundef{module@\activate@defs@uri @activated}{%
991       {\csname module@defs@\activate@defs@uri\endcsname}{}%
992       \@namedef{module@\activate@defs@uri @activated}{true}%
993     }%
994     \def\inimport@thismodule{#1}%
995     \stex@debug{End of import #1}%
996     \ifx\inimport@thismodule\inimport@module\@inimportfalse%
997       \stex@debug{Leaving import #1}%
998     \fi%
999 }%

```

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

```
1000 \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.

```

1001 \def\inputref@preskip{}
1002 \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).

```

1003 \newrobustcmd\inputref[2] [] {%
1004   \importmodule@bookkeeping{#1}{#2}{%
1005     %\inputreftrue
1006     \inputref@preskip%
1007     \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
1008     \inputref@postskip%
1009   }%
1010 }%

```

Test 27: [Module 3.13](#)[type.en]:

3.5 Symbols/Notations/Verbalizations

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

```
1011 \newif\if@symdeflocal\@symdeflocalfalse
```

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

```
1012 \def\define@in@module#1#2{
1013   \expandafter\edef\csname #1\endcsname{#2}%
1014   \edef\define@in@module@temp{%
1015     \def\expandafter\noexpand\csname#1\endcsname%
1016     {#2}%
1017   }%
1018   \if@symdeflocal\else%
1019     \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1020     \expandafter\endcsname\expandafter{\define@in@module@temp}%
1021   \fi%
1022 }
```

`\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.

```
1023 \addmetakey{symdecl}{name}%
1024 \addmetakey{symdecl}{type}%
1025 \addmetakey{symdecl}{args}%
1026 \addmetakey[false]{symdecl}{local}[true]%
1027
1028 \newcommand\symdecl[2][{}]{%
1029   \ifcsdef{this@module}{%
1030     \metasetkeys{symdecl}{#1}%
1031     \ifcsvoid{symdecl@name}{
1032       \edef\symdecl@name{#2}%
1033     }{}%
1034     \edef\symdecl@uri{\module@uri\@QuestionMark\symdecl@name}%
1035     \ifcsvoid{stex@symbol@\symdecl@name}{%
1036       \expandafter\edef\csname stex@symbol@\symdecl@name\endcsname{\symdecl@uri}%
1037     }{%
1038       \expandafter\def\csname stex@symbol@\symdecl@name\endcsname{\detokenize{ambiguous}}%
1039     }%
1040     \edef\symdecl@symbolmacro{%
1041       \noexpand\ifcsvoid{stex@symbol@\symdecl@name}{%
1042         \expandafter\edef\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\symd
1043       }{%
1044         \expandafter\def\expandafter\noexpand\csname stex@symbol@\symdecl@name\endcsname{\detok
1045       }%
1046     }%
1047     \ifcsvoid{symdecl@type}{}%
1048     \setbox\modules@import@tempbox\hbox{$\symdecl@type$} % only to have latex check this
```



```

1049 }%
1050 \ifcsvoid{symdecl@args}{\csgdef{symdecl@uri\@QuestionMark args}{}}{%
1051   \IfInteger{symdecl@args}{\notation@numto@ia@{symdecl@args}{\csxdef{symdecl@uri\@QuestionMa
1052     \ex\globale\ex\let\csname\symdecl@uri\@QuestionMark args\endcsname\symdecl@args%
1053   }%
1054 }%
1055 \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
1056 \expandafter\endcsname\expandafter{\symdecl@symbolmacro}%
1057 \ifcsvoid{symdecl@uri}{%
1058   \ifcsvoid{module@names@\module@uri}{%
1059     \csxdef{module@names@\module@uri}{\symdecl@name}%
1060   }{%
1061     \csxdef{module@names@\module@uri}{\symdecl@name,%
1062       \csname module@names@\module@uri\endcsname}%
1063   }%
1064 }{%
1065 % not compatible with circular dependencies, e.g. test/omdoc/07-modules/smsstest.tex
1066 \PackageWarning{stex}{symbol already defined: \symdecl@uri}{%
1067   You need to pick a fresh name for your symbol%
1068 }%
1069 }%
1070 \define@in@module\symdecl@uri{\noexpand\@invoke@symbol{\symdecl@uri}}%
1071 \IfStrEq{symdecl@local}{false}{%
1072   \define@in@module{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
1073 }{%
1074   \csdef{#2}{\noexpand\@invoke@symbol{\symdecl@uri}}%
1075 }%
1076 }{%
1077 \PackageError{stex}{\detokenize{symdecl} not in a module}{You need to be in a module%
1078 in order to declare a new symbol}
1079 }%
1080 \if@inimport\else\latexml@symdecl\symdecl@uri{\symdecl@type$}\fi%
1081 \if@insymdef\else\parsemodule@maybesetcodes\fi%
1082 }
1083 \def\latexml@symdecl#1{\latexml@annotate@invisible{symdecl}{#1}{\ }}

```

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

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

3.5.1 Notations

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

```

1084 \edef\stex@ambiguous{\detokenize{ambiguous}}
1085 \edef\stex@macrostring{\detokenize{macro:->\@invoke@symbol}}
1086 \def\modules@getURIfromName#1{%
1087   \def\notation@uri{}%
1088   \edef\modules@getURI@name{#1}%

```

```

1089 \ifcsvoid{\modules@getURI@name}{
1090   \edef\modules@temp@meaning{
1091 }{
1092   \edef\modules@temp@meaning{\expandafter\meaning\csname\modules@getURI@name\endcsname}
1093 }
1094 \IfBeginWith\modules@temp@meaning\stex@macrostring{
1095   % is a \@invoke@symbol macro
1096   \StrPosition\modules@temp@meaning\@close@brace[\stex@tempnum]
1097   \StrMid\modules@temp@meaning{26}{\the\numexpr\stex@tempnum-1\@Space}{\notation@uri]
1098 }{
1099   % Check whether full URI or module?symbol or just name
1100   \StrCount\modules@getURI@name\@QuestionMark[\isuri@number]
1101   \ifnum\isuri@number=2
1102     \edef\notation@uri{\modules@getURI@name}
1103   \else
1104     \ifnum\isuri@number=1
1105       % module?name
1106       \StrCut\modules@getURI@name\@QuestionMark\isuri@mod\isuri@name
1107       \ifcsvoid{stex@module@\isuri@mod}{
1108         \PackageError{stex}{No module with name \isuri@mod\@Space loaded}{}
1109       }{
1110         \expandafter\ifx\csname stex@module@\isuri@mod\endcsname\stex@ambiguous
1111         \PackageError{stex}{Module name \isuri@mod\@Space is ambiguous}{}
1112       }
1113       \else
1114         \edef\notation@uri{\csname stex@module@\isuri@mod\endcsname\@URI\@QuestionMark\isuri@mod}
1115       \fi
1116     }
1117   \else
1118     %name
1119     \ifcsvoid{stex@symbol@\modules@getURI@name}{
1120       \PackageError{stex}{No symbol with name \modules@getURI@name\@Space known}{}
1121     }{
1122       \ifcsvoid{\module@uri\@QuestionMark\modules@getURI@name}{
1123         \expandafter\ifx\csname stex@symbol@\modules@getURI@name\endcsname\stex@ambiguous
1124         % Symbol name ambiguous and not in current module
1125         \PackageError{stex}{Symbol name, URI or macroname \detokenize{#1} found!}{}%
1126       }
1127       \else
1128         % Symbol not in current module, but unambiguous
1129         \edef\notation@uri{\csname stex@symbol@\modules@getURI@name\endcsname}
1130       \fi
1131     }{ % Symbol in current module
1132       \edef\notation@uri{\module@uri\@QuestionMark\modules@getURI@name}
1133     }
1134   }
1135 }
1136 }

```

\notation Adds a new notation to a symbol foo, as in: `\notation[lang=en,arity=0,variant=op]{foo}{...}`

`\notation[variant=bar]{foo}[2]{...}\notation[args=aia,prec=500;50x49x51]{foo}{#1 bla #2}`
the actual notation is ultimately stored in `\<uri>#<variant>`, where `<variant>` contains `arity`, `lang` and `variant` in that order.

1137 `\newif\if@innotation\@innotationfalse`

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

1138 `\providerobustcmd\notation[2][]{%}`
1139 `\edef\notation@first{#1}%`
1140 `\edef\notation@second{#2}%`
1141 `\notation@%`
1142 `}`
1143
1144 `\newcommand\notation@[2][0]{%`
1145 `\edef\notation@donext{\noexpand\notation@@[\notation@first]%}`
1146 `{\notation@second}[#1]}%`
1147 `\notation@donext{#2}%`
1148 `}`
1149

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

1150 `\def\notation@parse@params#1#2{%`
1151 `\def\notation@curr@prec{%}`
1152 `\def\notation@curr@args{%}`
1153 `\def\notation@curr@variant{%}`
1154 `\def\notation@curr@arityvar{%}`
1155 `\def\notation@curr@provided@arity{#2}`
1156 `\def\notation@curr@lang{%}`
1157 `\def\notation@options@temp{#1}`
1158 `\notation@parse@params@%`
1159 `\ifx\notation@curr@args\@empty%`
1160 `\ifx\notation@curr@provided@arity\@empty%`
1161 `\notation@num@to@ia\notation@curr@arityvar%`
1162 `\else%`
1163 `\notation@num@to@ia\notation@curr@provided@arity%`
1164 `\fi%`
1165 `\fi%`
1166 `\StrLen\notation@curr@args[\notation@curr@arity]%`
1167 `}`
1168 `\def\notation@parse@params@{%`
1169 `\IfSubStr\notation@options@temp,{%`
1170 `\StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%`
1171 `\notation@parse@param%`
1172 `\notation@parse@params@%`
1173 `}{\ifx\notation@options@temp\@empty\else%`
1174 `\let\notation@option@temp\notation@options@temp%`
1175 `\notation@parse@param%`
1176 `\fi}%`
1177 `}`

```

1178
1179 \def\notation@parse@param{%
1180   \path@trimstring\notation@option@temp%
1181   \ifx\notation@option@temp\@empty\else%
1182     \IfSubStr\notation@option@temp={%
1183       \StrCut\notation@option@temp=\notation@key\notation@value%
1184       \path@trimstring\notation@key%
1185       \path@trimstring\notation@value%
1186       \IfStrEq\notation@key{prec}{%
1187         \edef\notation@curr@prec{\notation@value}%
1188       }{%
1189         \IfStrEq\notation@key{args}{%
1190           \edef\notation@curr@args{\notation@value}%
1191         }{%
1192           \IfStrEq\notation@key{lang}{%
1193             \edef\notation@curr@lang{\notation@value}%
1194           }{%
1195             \IfStrEq\notation@key{variant}{%
1196               \edef\notation@curr@variant{\notation@value}%
1197             }{%
1198               \IfStrEq\notation@key{arity}{%
1199                 \edef\notation@curr@arityvar{\notation@value}%
1200               }{%
1201                 }}}}%
1202       }{%
1203         \edef\notation@curr@variant{\notation@option@temp}%
1204       }%
1205   \fi%
1206 }
1207
1208 % converts an integer to a string of 'i's, e.g. 3 => iii,
1209 % and stores the result in \notation@curr@args
1210 \def\notation@num@to@ia#1{%
1211   \IfInteger{#1}{
1212     \notation@num@to@ia@#1%
1213   }{%
1214     %
1215   }%
1216 }
1217 \def\notation@num@to@ia@#1{%
1218   \ifnum#1>0%
1219     \edef\notation@curr@args{\notation@curr@args i}%
1220     \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
1221   \fi%
1222 }
1223
1224
1225 \newcount\notation@argument@counter
1226

```

```

1227 % parses the notation arguments and wraps them in
1228 % \notation@assoc and \notation@argprec for flexary arguments and precedences
1229 \def\notation@@[#1]#2[#3]#4{%
1230   \modules@getURIfromName{#2}%
1231   \notation@parse@params{#1}{#3}%
1232   \let\notation@curr@todo@args\notation@curr@args%
1233   \def\notation@temp\notation{}%
1234   \ex\renewcommand\ex\notation@temp\notation\ex[\notation@curr@arity]{#4}%
1235   % precedence
1236   \let\notation@curr@precstring\notation@curr@prec%
1237   \IfSubStr\notation@curr@prec;{%
1238     \StrCut\notation@curr@prec;\notation@curr@prec\notation@curr@prec%
1239     \ifx\notation@curr@prec\empty\def\notation@curr@prec{0}\fi%
1240   }{%
1241     \ifx\notation@curr@prec\empty%
1242       \ifnum\notation@curr@arity=0\relax%
1243         \edef\notation@curr@prec{\infprec}%
1244       \else%
1245         \def\notation@curr@prec{0}%
1246       \fi%
1247     \else%
1248       \edef\notation@curr@prec{\notation@curr@prec}%
1249       \def\notation@curr@prec{}%
1250     \fi%
1251   }%
1252   % arguments
1253   \notation@argument@counter=0%
1254   \def\notation@curr@extargs{}%
1255   \notation@do@args%
1256 }
1257
1258 \edef\notation@ichar{\detokenize{i}}%
1259
1260 % parses additional notation components for (associative) arguments
1261 \def\notation@do@args{%
1262   \advance\notation@argument@counter by 1%
1263   \def\notation@nextarg@temp{}%
1264   \ifx\notation@curr@todo@args\empty%
1265     \ex\notation@after%
1266   \else%
1267     % argument precedence
1268     \IfSubStr\notation@curr@prec{x}{%
1269       \StrCut\notation@curr@prec{x}\notation@curr@argprec\notation@curr@prec%
1270     }{%
1271       \edef\notation@curr@argprec{\notation@curr@prec}%
1272       \def\notation@curr@prec{}%
1273     }%
1274     \ifx\notation@curr@argprec\empty%
1275       \let\notation@curr@argprec\notation@curr@prec%
1276     \fi%

```

```

1277 \StrChar\notation@curr@todo@args1[\notation@argchar]%
1278 \edef\notation@argchar{\ex\detokenize\ex{\notation@argchar}}%
1279 \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1280 \ifx\notation@argchar\notation@ichar%
1281 % normal argument
1282 \edef\notation@nextarg@temp{%
1283   {\stex@arg{\the\notation@argument@counter}}{\notation@curr@argprec}{#####\the
1284 }%
1285 \ex\g@addto@macro@safe\ex\notation@curr@extargs%
1286 \ex{\notation@nextarg@temp}%
1287 \ex\ex\ex\notation@do@args%
1288 \else%
1289 % associative argument
1290 \ex\ex\ex\notation@parse@assocarg%
1291 \fi%
1292 \fi%
1293 }
1294
1295 \def\notation@parse@assocarg#1{%
1296   \edef\notation@nextarg@temp{%
1297     {\stex@arg{\the\notation@argument@counter}}{\notation@curr@argprec}{\notation@assoc{#1}{###
1298 }%
1299   \ex\g@addto@macro@safe\ex\notation@curr@extargs\ex{\notation@nextarg@temp}%
1300   \notation@do@args%
1301 }
1302
1303 \protected\def\safe@newcommand#1{%
1304   \ifdefined#1\ex\renewcommand\else\ex\newcommand\fi#1%
1305 }
1306
1307 % finally creates the actual macros
1308 \def\notation@after{
1309   % \notation@curr@prec
1310   % \notation@curr@args
1311   % \notation@curr@variant
1312   % \notation@curr@arity
1313   % \notation@curr@provided@arity
1314   % \notation@curr@lang
1315   % \notation@uri
1316   \def\notation@temp@fragment{}%
1317   \ifx\notation@curr@arityvar\@empty\else%
1318     \edef\notation@temp@fragment{arity=\notation@curr@arityvar}%
1319   \fi%
1320   \ifx\notation@curr@lang\@empty\else%
1321     \ifx\notation@temp@fragment\@empty%
1322       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1323     \else%
1324       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1325     \fi%
1326   \fi%

```

```

1327 \ifx\notation@curr@variant\@empty\else%
1328 \ifx\notation@temp@fragment\@empty%
1329 \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1330 \else%
1331 \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1332 \fi%
1333 \fi%
1334 \ex\ex\ex\def\ex\ex\ex\notation@temp@notation\ex\ex\ex%
1335 {\ex\notation@temp@notation\notation@curr@extargs}%
1336 \ifnum\notation@curr@arity=0
1337 \edef\notation@temp@notation{\stex@oms{\notation@uri\@Fragment\notation@temp@fragment}}{\not
1338 \else
1339 \edef\notation@temp@notation{\stex@oma{\notation@uri\@Fragment\notation@temp@fragment}}{\not
1340 \fi
1341 \stex@debug{Notation \notation@uri: \meaning\notation@temp@notation}%
1342 \notation@final%
1343 \parsemodule@maybesetcodes%
1344 }
1345
1346 \def\notation@final{%
1347 \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1348 \stex@debug{Defining \notation@csname of arity \notation@curr@arity}%
1349 \ifcvoid{\notation@csname}{%
1350 \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1351 \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@curr@arity\ex]%
1352 \ex{\notation@temp@notation}%
1353 \edef\symdecl@temps{%
1354 \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@curr@ari
1355 }%
1356 \ex@g@addto@macro@safe\csname module@defs@module@uri\ex\endcsname\ex{\symdecl@temps}%
1357 \ex@g@addto@macro@safe\csname module@defs@module@uri\ex\endcsname\ex{\ex{\notation@temp@no
1358 }-%
1359 \PackageWarning{stex}{notation already defined: \notation@csname}{%
1360 Choose a different set of notation options (variant,lang,arity)%
1361 }%
1362 }%
1363 \@innotationfalse%
1364 \if@inimport\else\if@latexml%
1365 \let\notation@simarg@args\notation@curr@args%
1366 \notation@argument@counter=0%
1367 \def\notation@simargs{%
1368 \notation@simulate@arguments%
1369 \latexml\notation\notation@uri\notation@temp@fragment\notation@curr@args\notation@curr@prec
1370 {\csname\notation@csname\ex\endcsname\notation@simargs$}%
1371 \fi\fi%
1372 }
1373 \def\notation@simulate@arguments{%
1374 \ifx\notation@simarg@args\@empty\else%
1375 \advance\notation@argument@counter by 1%
1376 \IfBeginWith\notation@simarg@args{i}{%

```

```

1377 \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\the\notation@argument
1378 }{%
1379 \edef\notation@simargs{\notation@simargs{\noexpand\textrm{\@Fragment\@Fragment\the\notation
1380 }%
1381 \StrGobbleLeft\notation@simarg@args1[\notation@simarg@args]%
1382 \notation@simulate@arguments%
1383 \fi%
1384 }
1385 % URI, fragment, arity, notation
1386 \def\latexml@notation#1#2#3#4#5{\latexml@annotate@invisible{notation}{#1}{%
1387 \latexml@annotate{fragment}{#2}{\ }%
1388 \latexml@annotate{arity}{#3}{\ }%
1389 \latexml@annotate{precedence}{#4}{\ }%
1390 \latexml@annotate{notation}{#5}{\ }%
1391 }}

```

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

```

1392 \protected\def\notation@assoc#1#2{% function, argv
1393 \let\@tmpop=\relax% do not print the function the first time round
1394 \@for\@I:=#2\do{\@tmpop% print the function
1395 % write the i-th argument with locally updated precedence
1396 \@I%
1397 \def\@tmpop{#1}%
1398 }%
1399 }%
1400
1401 \def\notation@lparen{()
1402 \def\notation@rparen{)}
1403 \def\infpref{1000000}
1404 \def\neginfpref{-\infpref}
1405
1406 \newcount\notation@downprec
1407 \notation@downprec=\neginfpref
1408
1409 % patching displaymode
1410 \newif\if@displaymode\@displaymodefalse
1411 \ex\everydisplay\ex{\the\everydisplay\@displaymodetrue}
1412 \let\old@displaystyle\displaystyle
1413 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1414
1415 \protected\def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1416 \def\notation@innertmp{#1}%
1417 \if@displaymode%
1418 \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1419 \ex\notation@resetbrackets\ex\notation@innertmp%
1420 \ex\right\notation@rparen%
1421 \else%
1422 \ex\ex\ex\notation@lparen%
1423 \ex\notation@resetbrackets\ex\notation@innertmp%

```



```

1424     \notation@rparen%
1425     \fi%
1426 }
1427
1428 \protected\def\withbrackets#1#2#3{%
1429     \edef\notation@lparen{#1}%
1430     \edef\notation@rparen{#2}%
1431     #3%
1432     \notation@resetbrackets%
1433 }
1434
1435 \protected\def\notation@resetbrackets{%
1436     \def\notation@lparen{()%
1437     \def\notation@rparen{)%}
1438 }
1439
1440 \protected\def\stex@oms#1#2#3{%
1441     \if@innotation%
1442         \notation@symprec{#2}{#3}%
1443     \else%
1444         \@innotationtrue%
1445         \latexml@oms{#1}{\notation@symprec{#2}{#3}}%
1446         \@innotationfalse%
1447     \fi%
1448 }
1449
1450 % for LaTeXML Bindings
1451 \def\latexml@oms#1#2{%
1452     \if@latexml\latexml@annotate{OMS}{#1}{#2}\else#2\fi%
1453 }
1454
1455 \protected\def\stex@oma#1#2#3{%
1456     \if@innotation%
1457         \notation@symprec{#2}{#3}%
1458     \else%
1459         \@innotationtrue%
1460         \latexml@oma{#1}{\notation@symprec{#2}{#3}}%
1461         \@innotationfalse%
1462     \fi%
1463 }
1464
1465 % for LaTeXML Bindings
1466 \def\latexml@oma#1#2{%
1467     \if@latexml\latexml@annotate{OMA}{#1}{#2}\else#2\fi%
1468 }
1469
1470 \def\notation@symprec#1#2{%
1471     \ifnum#1>\notation@downprec\relax%
1472         \notation@resetbrackets#2%
1473     \else%

```

```

1474 \ifnum\notation@downprec=\infpref\relax%
1475 \notation@resetbrackets#2%
1476 \else
1477 \if@inarray@
1478 \notation@resetbrackets#2
1479 \else\dobrackets{#2}\fi%
1480 \fi\fi%
1481 }
1482
1483 \newif\if@inarray@\@inarray@false
1484
1485
1486 \protected\def\stex@arg#1#2#3{%
1487 \innotationfalse%
1488 \latexml@arg{#1}{\notation@argprec{#2}{#3}}%
1489 \innotationtrue%
1490 }
1491
1492 % for LaTeXML Bindings
1493 \def\latexml@arg#1#2{%
1494 \if\latexml\latexml@annotate{arg}{#1}{#2}\else#2\fi%
1495 }
1496
1497 \def\notation@argprec#1#2{%
1498 \def\notation@innertmp{#2}
1499 \edef\notation@downprec@temp{\number#1}%
1500 \notation@downprec=\expandafter\notation@downprec@temp%
1501 \expandafter\relax\expandafter\notation@innertmp%
1502 \expandafter\notation@downprec\expandafter=\number\notation@downprec\relax%
1503 }

```

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

```

1504 \protected\def\@invoke@symbol#1{%
1505 \def\@invoke@symbol@first{#1}%
1506 \symbol@args%
1507 }

```

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

```

1508 \newcommand\symbol@args[1][{}]{%
1509 \notation@parse@params{#1}{}%
1510 \def\notation@temp@fragment{}%
1511 \ifx\notation@curr@arityvar\@empty\else%
1512 \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1513 \fi%
1514 \ifx\notation@curr@lang\@empty\else%
1515 \ifx\notation@temp@fragment\@empty%
1516 \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1517 \else%

```

```

1518     \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1519     \fi%
1520 \fi%
1521 \ifx\notation@curr@variant\@empty\else%
1522     \ifx\notation@temp@fragment\@empty%
1523         \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1524     \else%
1525         \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
1526     \fi%
1527 \fi%
1528 %
1529 \ifmmode\def\invoke@symbol@next{\@invoke@symbol@math\@invoke@symbol@first\notation@temp@fragm
1530 \else\def\invoke@symbol@next{\@invoke@symbol@text\@invoke@symbol@first\notation@temp@fragment.
1531 \invoke@symbol@next%
1532 }

```

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

```

1533 \def\@invoke@symbol@math#1#2{%
1534     \csname #1\@Ffragment#2\endcsname%
1535 }

```

TODO:

```

1536 \def\@invoke@symbol@text#1#2{%
1537 }

```

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

```

1538 \def\setstexlang#1{%
1539     \def\stex@lang{#1}%
1540 }%
1541 \setstexlang{en}
1542 \def\setstexvariant#1#2{%
1543     % TODO
1544 }
1545 \def\setstexvariants#1{%
1546     \def\stex@variants{#1}%
1547 }

```

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

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

```

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

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

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

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



$$\left[ \times \left\{ \frac{vara}{varb} , \plus \left\{ \frac{vara}{\frac{vara}{varb}} , \times \right. \right. \right.$$


$$\left. \left. \left. \left\{ \varc , \plus \left\{ \vard , \vare , 2 \right\} \right\} \right\} \right\} \right]$$


```

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

3.6 Term References

```

\ifhref
1548 \newif\ifhref\hreffalse%
1549 \AtBeginDocument{%
1550   \ifpackageloaded{hyperref}{%
1551     \hreftrue%
1552   }{%
1553     \hreffalse%
1554   }%
1555 }

```

`\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.

```

1556 \newbox\stex@targetbox
1557 \def\termref@maketarget#1#2{%
1558   % #1: symbol URI
1559   % #2: text
1560   \stex@debug{Here: #1 <> #2}%
1561   \ifhref\if@smsmode\else%

```

```

1562     \hypertarget{sref@#1@target}{#2}%
1563 \fi\fi%
1564 \stex@debug{Here!}%
1565 \expandafter\edef\csname sref@#1\endcsname##1{%
1566     \ifhref\if@smsmode\else\noexpand\hyperlink{sref@#1@target}{##1}\fi\fi%
1567 }%
1568 }

\@termref
1569 \def\@termref#1#2{%
1570 % #1: symbol URI
1571 % #2: text
1572 \ifcsvoid{#1}{%
1573     \StrCut[2]{#1}\@QuestionMark\termref@mod\termref@name%
1574     \ifcsvoid{\termref@mod}{%
1575         \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
1576     }{%
1577         \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
1578             contains no symbol with name \termref@name.%
1579         }{%
1580         }%
1581     }{%
1582         \ifcsvoid{sref@#1}{%
1583             #2% TODO: No reference point exists!
1584         }{%
1585             \csname sref@#1\endcsname{#2}%
1586         }%
1587     }%
1588 }

\tref
1589
1590 \def\@capitalize#1{\uppercase{#1}}%
1591 \newrobustcmd\capitalize[1]{\expandafter\@capitalize #1}%
1592
1593 \newcommand\tref[2][]{%
1594     \edef\tref@name{#1}%
1595     \expandafter\modules@getURIfromName\expandafter{\tref@name}%
1596     \expandafter\@termref\expandafter{\notation@uri}{#2}%
1597 }
1598 \def\trefs#1{%
1599     \modules@getURIfromName{#1}%
1600 % TODO
1601 }
1602 \def\Tref#1{%
1603     \modules@getURIfromName{#1}%
1604 % TODO
1605 }
1606 \def\Trefs#1{%
1607     \modules@getURIfromName{#1}%

```

```

1608 % TODO
1609 }

\defi
1610 \addmetakey{defi}{name}
1611 \def\@definiendum#1#2{%
1612   \parsemodule@maybesetcodes%
1613   \stex@debug{Here: #1 | #2}%
1614   \termref@maketarget{#1}{#2}%\termref@maketarget{#1}{\defemph{#2}}%
1615 }
1616
1617 \newcommand\defi[2][{}]{%
1618   \metasetkeys{defi}{#1}%
1619   \ifx\defi@name\@empty%
1620     \symdecl@constructname{#2}%
1621     \let\defi@name\symdecl@name%
1622     \let\defi@verbalization\symdecl@verbalization%
1623   \else%
1624     \edef\defi@verbalization{#2}%
1625   \fi%
1626   \ifcvoid{\module@uri\@QuestionMark\defi@name}{%
1627     \symdecl\defi@name%
1628   }\edef\symdecl@uri{\module@uri\@QuestionMark\defi@name}}%
1629   \@definiendum\symdecl@uri\defi@verbalization%
1630 }
1631 \def\Defi#1{%
1632   \symdecl{#1}%
1633   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization}%
1634 }
1635 \def\defis#1{%
1636   \symdecl{#1}%
1637   \@definiendum\symdecl@uri{\symdecl@verbalization s}%
1638 }
1639 \def\Defis#1{%
1640   \symdecl{#1}%
1641   \@definiendum\symdecl@uri{\capitalize\symdecl@verbalization s}%
1642 }

```

3.7 sref

We find out whether the `hyperref` package is loaded, since we may want to use it for cross-references, for which we set up some internal macros that gracefully degrade if `hyperref` is not loaded.

```

\sref@*@ifh
1643 \newif\ifhref\hreffalse%
1644 \AtBeginDocument{%
1645   \@ifpackageloaded{hyperref}{%
1646     \hreftrue%

```

```

1647 }{%
1648   \hreffalse%
1649 }%
1650 }%
1651 \newcommand\sref@href@ifh[2]{%
1652   \ifhref%
1653     \href{#1}{#2}%
1654   \else%
1655     #2%
1656   \fi%
1657 }%
1658 \newcommand\sref@hlink@ifh[2]{%
1659   \ifhref%
1660     \hyperlink{#1}{#2}%
1661   \else%
1662     #2%
1663   \fi%
1664 }%
1665 \newcommand\sref@target@ifh[2]{%
1666   \ifhref%
1667     \hypertarget{#1}{#2}%
1668   \else%
1669     #2%
1670   \fi%
1671 }%

```

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.

```

1672 \def\sref@target{%
1673   \ifx\sref@id@empty%
1674     \relax%
1675   \else%
1676     \edef\@target{\sref@ifcsundef{sref@part}}{\sref@part @}\sref@id @target}%
1677     \sref@target@ifh\@target{}%
1678   \fi%
1679 }%

```

\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.

```

1680 \addmetakey{srefaddidkey}{prefix}
1681 \newcommand\srefaddidkey[2][]{%
1682   \metasetkeys{srefaddidkey}{#1}%
1683   \@metakeys@ext@clear@keys{#2}{\sref@id}{}}% id cannot have a default

```

```

1684 \metakeys@ext@clear@keys{#2}{id}{}%
1685 \metakeys@ext@showkeys{#2}{id}%
1686 \define@key{#2}{id}{%
1687   \edef\sref@id{\srefaddidkey@prefix ##1}%
1688   %\expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
1689   \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1690 }%
1691 }%

```

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

```

1692 \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.

```

1693 \ifextrefs%
1694   \newwrite\refs@file%
1695 \else%
1696   \def\refs@file{\@auxout}%
1697 \fi%

```

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

```

1698 \newcommand\sref@def[3]{%
1699   \protected@write\refs@file{}\string\sref@def{#1}{#2}{#3}}%
1700 }%

```

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

```

1701 \newcommand\sref@label[2]{%
1702   \sref@def{\ifcsundef\sref@part}{\sref@part @}{#2}{page}{\thepage}%
1703   \sref@def{\ifcsundef\sref@part}{\sref@part @}{#2}{label}{#1}%
1704 }%

```

`\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`.

```

1705 \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.

```

1706 \def\sref@id{} % make sure that defined
1707 \newcommand\sref@label@id[1]{%
1708   \ifx\sref@id\empty%
1709     \relax%
1710   \else%
1711     \sref@label{#1}{\sref@id}%
1712   \fi%
1713 }%

```


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

```

1714 \newcommand\sref@label@id@arg[2]{%
1715   \def\@@id{#2}
1716   \ifx\@@id\empty%
1717     \relax%
1718   \else%
1719     \sref@label{#1}{\@@id}%
1720   \fi%
1721 }%
```

3.8 smultiling

`modsig` The `modsig` environment is just a layer over the `module` environment. We also redefine macros that may occur in module signatures so that they do not create markup. Finally, we set the flag `\mod@<mod>@multiling` to `true`.

```

1722 \newenvironment{modsig}[2][\def\@test{#1}%
1723 \ifx\@test\empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1724 \expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
1725 %\ignorespacesandpars
1726 }
1727 {\end{module}}\ignorespacesandpars
1728 }
```

3.9 smglom

`\gimport` Just a shortcut, we have a starred and unstarred version, the first one is conservative. For example, if we execute:

```
\gimport[smglom/numberfields]{naturalnumbers}
```

First we are redirected to `\@gimport@nostar`, we store the `smglom/numberfields` (*the repo's path*) in `\@test`, then store `\mh@currentrepos` (*current directory*) in `\mh@repos`. If no repo's path is offered, that means the module to import is under the same directory, so we let `mhrepos=\mh@repos` and pass bunch of parameters to `\importmhmodule`, which is defined in `module.sty`. If there's a repo's path, then we let `mhrepos=<the repo's path>`. Finally we use `\mhcurrentrepos` (defined in `module.sty`) to change the `\mh@currentrepos`.

```

1729 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1730 \newrobustcmd\@gimport@star[2][\def\@test{#1}%
1731 \edef\mh@repos{\mh@currentrepos}%
1732 \ifx\@test\empty%
1733 \importmhmodule[conservative,mhrepos=\mh@repos,path=#2]{#2}%
1734 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1735 \mathhub@setcurrentreposinfo{\mh@repos}%
1736 %\ignorespacesandpars
```

```

1737 \parsemodule@maybesetcodes}
1738 \newrobustcmd\@gimport@nostar[2][\def\@test{#1}%
1739 \edef\mh@@repos{\mh@currentrepos}%
1740 \ifx\@test\@empty%
1741 \importmhmodule[mhrepos=\mh@@repos,path=#2]{#2}%
1742 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1743 \mathhub@setcurrentreposinfo{\mh@@repos}%
1744 %\ignorespacesandpars
1745 \parsemodule@maybesetcodes}

```

3.10 mathhub

`\libinput` the `\libinput` macro inputs from the `lib` directory of the MathHub repository and then the `meta-inf/lib` repository of the group, if they exist. Since in practice nested libinputs may occur, we make sure that we stash the old values of `\mh@inffile` and `\mh@libfile` and restore them at the end.

```

1746 \def\modules@@first#1/#2;{#1}
1747 \newcommand\libinput[1]{%
1748 \stex@debug{Libinput current repo: \meaning\mh@currentrepos}%
1749 \ifcsvoid{mh@currentrepos}{%
1750 \PackageError{stex}{current MathHub repository not found}{}}%
1751 {}
1752 \edef\@mh@group{\expandafter\modules@@first\mh@currentrepos;}
1753 \let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
1754 \def\mh@inffile{\MathHub{\@mh@group/meta-inf/lib/#1}}
1755 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1756 \IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
1757 \IfFileExists\mh@inffile{}{\IfFileExists\mh@libfile}{}%
1758 {\PackageError{stex}
1759 {Library file missing; cannot input #1.tex\MessageBreak%
1760 Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
1761 do not exist}%
1762 {Check whether the file name is correct}}}%
1763 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1764 \let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}

```

3.11 omdoc/omgroup

```

1765 \newcount\section@level
1766
1767 \section@level=2
1768 \ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{%
1769 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{%
1770 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{%
1771 \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>`.

```

1772 \newcommand\omgroup@nonum[2]{%

```

```

1773 \ifx\hyper@anchor\@undefined\else\phantomsection\fi%
1774 \addcontentsline{toc}{#1}{#2}\@nameuse{#1}*{#2}}

\omgroup@num convenience macro: \omgroup@nonum{<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.

1775 \newcommand\omgroup@num[2]{%
1776 \edef\@ID{\sref@id}
1777 \ifx\omgroup@short\@empty% no short title
1778 \@nameuse{#1}{#2}%
1779 \else% we have a short title
1780 \@ifundefined{rdfmeta@sectioning}%
1781 {\@nameuse{#1}[\omgroup@short]{#2}}%
1782 {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
1783 \fi%
1784 \sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@ID}

omgroup
1785 \def\@true{true}
1786 \def\@false{false}
1787 \srefaddidkey{omgroup}
1788 \addmetakey{omgroup}{date}
1789 \addmetakey{omgroup}{creators}
1790 \addmetakey{omgroup}{contributors}
1791 \addmetakey{omgroup}{srccite}
1792 \addmetakey{omgroup}{type}
1793 \addmetakey*{omgroup}{short}
1794 \addmetakey*{omgroup}{display}
1795 \addmetakey[false]{omgroup}{loadmodules}[true]

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

1796 \newif\if@mainmatter\@mainmattertrue
1797 \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.

1798 \addmetakey{omdoc@sect}{name}
1799 \addmetakey[false]{omdoc@sect}{clear}[true]
1800 \addmetakey{omdoc@sect}{ref}
1801 \addmetakey[false]{omdoc@sect}{num}[true]
1802 \newcommand\omdoc@sectioning[3][]{\metasetkeys{omdoc@sect}{#1}%
1803 \ifx\omdoc@sect@clear\@true\cleardoublepage\fi%
1804 \if@mainmatter% numbering not overridden by frontmatter, etc.
1805 \ifx\omdoc@sect@num\@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
1806 \def\current@section@level{\omdoc@sect@name}%
1807 \else\omgroup@nonum{#2}{#3}%

```

```

1808 \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.
1809 \newcommand\omgroup@redefine@addtocontents[1]{%
1810 %\edef\@import{#1}%
1811 %\@for\@I:=\@import\do{%
1812 %\edef\@path{\csname module@\@I @path\endcsname}%
1813 %\@ifundefined{tf@toc}\relax%
1814 % {\protected@write\tf@toc}{\string\requiremodules{\@path}}}%
1815 %\ifx\hyper@anchor\@undefined% hyperref.sty loaded?
1816 %\def\addcontentsline##1##2##3{%
1817 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}}%
1818 %\else% hyperref.sty not loaded
1819 %\def\addcontentsline##1##2##3{%
1820 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}{\@c
1821 %\fi
1822 }% hypreref.sty loaded?

now the omgroup environment itself. This takes care of the table of contents
via the helper macro above and then selects the appropriate sectioning com-
mand from article.cls. It also registers the current level of omgroups in the
\omgroup@level counter.
1823 \newcount\omgroup@level
1824 \newenvironment{omgroup}[2][]% keys, title
1825 {\metasetkeys{omgroup}{#1}\sref@target%
1826 \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.
1827 \ifx\omgroup@loadmodules\@true%
1828 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
1829 {\@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.
1830 \advance\section@level by 1\relax%
1831 \ifcase\section@level%
1832 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
1833 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
1834 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
1835 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
1836 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%
1837 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{#2}%
1838 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{#2}%
1839 \fi% \ifcase
1840 \at@begin@omgroup[#1]\section@level{#2}}% for customization
1841 {\advance\section@level by -1\advance\omgroup@level by -1}

```

and finally, we localize the sections

```

1842 \newcommand\omdoc@part@kw{Part}
1843 \newcommand\omdoc@chapter@kw{Chapter}
1844 \newcommand\omdoc@section@kw{Section}
1845 \newcommand\omdoc@subsection@kw{Subsection}
1846 \newcommand\omdoc@subsubsection@kw{Subsubsection}
1847 \newcommand\omdoc@paragraph@kw{paragraph}
1848 \newcommand\omdoc@subparagraph@kw{subparagraph}

\setSGvar set a global variable
1849 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}

\useSGvar use a global variable
1850 \newrobustcmd\useSGvar[1]{%
1851   \@ifundefined{sTeX@Gvar@#1}
1852   {\PackageError{omdoc}
1853    {The sTeX Global variable #1 is undefined}
1854    {set it with \protect\setSGvar}}
1855   \@nameuse{sTeX@Gvar@#1}}

blindomgroup
1856 \newcommand\at@begin@blindomgroup[1]{%
1857   \newenvironment{blindomgroup}
1858   {\advance\section@level by 1\at@begin@blindomgroup\setion@level}
1859   {\advance\section@level by -1}}

```

3.12 omtex

3.12.1 Mathematical Text

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

```

1860 \srefaddidkey{omtext}
1861 \addmetakey[] {omtext}{functions}
1862 \addmetakey* {omtext}{display}
1863 \addmetakey {omtext}{for}
1864 \addmetakey {omtext}{from}
1865 \addmetakey {omtext}{type}
1866 \addmetakey* {omtext}{title}
1867 \addmetakey* {omtext}{start}
1868 \addmetakey {omtext}{theory}
1869 \addmetakey {omtext}{continues}
1870 \addmetakey {omtext}{verbalizes}
1871 \addmetakey {omtext}{subject}

\st@flow We define this macro, so that we can test whether the display key has the value
flow
1872 \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.

```
1873 \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.

```
1874 \def\omtext@pre@skip{\smallskip}
1875 \def\omtext@post@skip{}
1876 \newenvironment{omtext}[1][\@in@omtexttrue%
1877   \bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
1878   \def\lec##1{\@lec{##1}}}%
1879   \omtext@pre@skip\par\noindent%
1880   \ifx\omtext@title\@empty%
1881     \ifx\omtext@start\@empty\else%
1882       \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
1883     \fi% end omtext@start empty
1884   \else\stDMemph{\omtext@title}:\enspace%
1885     \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
1886   \fi% end omtext@title empty
1887   %\ignorespacesandpars
1888 }
1889 {\egroup\omtext@post@skip\@in@omtextfalse%\ignorespacesandpars
1890 }
```

3.12.2 Phrase-level Markup

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

```
1891 \srefaddidkey{phrase}
1892 \addmetakey{phrase}{style}
1893 \addmetakey{phrase}{class}
1894 \addmetakey{phrase}{index}
1895 \addmetakey{phrase}{verbalizes}
1896 \addmetakey{phrase}{type}
1897 \addmetakey{phrase}{only}
1898 \newcommand\phrase[2][\metasetkeys{phrase}{#1}%
1899 \ifx\phrase@only\@empty\only<\phrase@only>{#2}\else #2\fi}
```

`\coref*`

```
1900 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}
1901 \newcommand\corefs[2]{#1\textsubscript{#2}}
1902 \newcommand\coreft[2]{#1\textsuperscript{#2}}
```

`\n*lex`

```
1903 \newcommand\nlex[1]{\green{\sl{#1}}}
1904 \newcommand\nlcex[1]{*\green{\sl{#1}}}
```

`sinlinequote`

```

1905 \def\@sinlinequote#1{‘‘{\sl{#1}}’’}
1906 \def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
1907 \newcommand\sinlinequote[2] []
1908 {\def\@opt{#1}\ifx\@opt\@empty\@sinlinequote{#2}\else\@@sinlinequote\@opt{#2}\fi}

```

3.12.3 Declarations (under development)

The declaration macros are still under development (i.e. the macros) are still under development and may change at any time. Currently they are completely empty.

```

1909 \newcommand\vdec[2] []{#2}
1910 \newcommand\vest[2] []{#2}
1911 \newcommand\vcond[2] []{#2}

```

EdN:1 \strucdec 1
 1912 \newcommand\strucdec[2] []{#2}

EdN:2 \impdec 2
 1913 \newcommand\impdec[2] []{#2}

3.12.4 Block-Level Markup

sblockquote

```

1914 \def\begin@sblockquote{\begin{quote}\sl}
1915 \def\end@sblockquote{\end{quote}}
1916 \def\begin@@sblockquote#1{\begin@sblockquote}
1917 \def\end@@sblockquote#1{\def\@lec##1{\textrm{##1}}\@lec{#1}\end@sblockquote}
1918 \newenvironment{sblockquote}[1] []
1919 { \def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
1920 { \ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}

```

sboxquote

```

1921 \newenvironment{sboxquote}[1] []
1922 {\def\@@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
1923 {\@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.

```

1924 \providecommand{\@@lec}[1]{(#{1})}
1925 \def\@lec#1{\strut\hfil\strut\hfill\@lec{#1}}
1926 \def\lec#1{\@lec{#1}\par}

```

¹EdNOTE: document above

²EdNOTE: document above

3.12.5 Index Markup

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

```

1927 \addmetakey{omdoc@index}{at}
1928 \addmetakey[false]{omdoc@index}{loadmodules}[true]
1929 \newcommand\omdoc@indexi[2][\ifindex%
1930 \metasetkeys{omdoc@index}{#1}%
1931 \@bsphack\begingroup\@sanitize%
1932 \protected@write\@indexfile{}\string\indexentry%
1933 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1934 \ifx\omdoc@index@loadmodules\@true%
1935 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
1936 \else #2\fi% loadmodules
1937 }\the page}}%
1938 \endgroup\@esphack\fi}%ifindex
1939 \newcommand\omdoc@indexii[3][\ifindex%
1940 \metasetkeys{omdoc@index}{#1}%
1941 \@bsphack\begingroup\@sanitize%
1942 \protected@write\@indexfile{}\string\indexentry%
1943 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1944 \ifx\omdoc@index@loadmodules\@true%
1945 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1946 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
1947 \else #2!#3\fi% loadmodules
1948 }\the page}}%
1949 \endgroup\@esphack\fi}%ifindex
1950 \newcommand\omdoc@indexiii[4][\ifindex%
1951 \metasetkeys{omdoc@index}{#1}%
1952 \@bsphack\begingroup\@sanitize%
1953 \protected@write\@indexfile{}\string\indexentry%
1954 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1955 \ifx\omdoc@index@loadmodules\@true%
1956 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1957 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1958 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1959 \else #2!#3!#4\fi% loadmodules
1960 }\the page}}%
1961 \endgroup\@esphack\fi}%ifindex
1962 \newcommand\omdoc@indexiv[5][\ifindex%
1963 \metasetkeys{omdoc@index}{#1}%
1964 \@bsphack\begingroup\@sanitize%
1965 \protected@write\@indexfile{}\string\indexentry%
1966 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%

```



```

1967 \ifx\omdoc@index@loadmodules \@true%
1968 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1969 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1970 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1971 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
1972 \else #2!#3!#4!#5\fi% loadmodules
1973 }\the page}}%
1974 \endgroup\@esphack\fi}%ifindex

```

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

`*indi*`

```

1975 \newcommand\aindi[3] []{{#2}\omdoc@indexi[#1]{#3}}
1976 \newcommand\indi[2] []{{#2}\omdoc@indexi[#1]{#2}}
1977 \newcommand\indis[2] []{{#2}\omdoc@indexi[#1]{#2s}}
1978 \newcommand\Indi[2] []{{\captitalize{#2}}\omdoc@indexi[#1]{#2}}
1979 \newcommand\Indis[2] []{{\capitalize{#2}}\omdoc@indexi[#1]{#2s}}
1980
1981 \newcommand\@indii[3] []{\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
1982 \newcommand\aindii[4] []{{#2}\@indii[#1]{#3}{#4}}
1983 \newcommand\indii[3] []{{#2 #3}\@indii[#1]{#2}{#3}}
1984 \newcommand\indiis[3] []{{#2 #3s}\@indii[#1]{#2}{#3}}
1985 \newcommand\Indii[3] []{{\captitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1986 \newcommand\Indiis[3] []{{\capitalize{#2 #3}}\@indii[#1]{#2}{#3}}
1987
1988 \newcommand\@indiii[4] []{\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexiii[#1]{#3}{#2 (#4)}}
1989 \newcommand\aindiii[5] []{{#2}\@indiii[#1]{#3}{#4}{#5}}
1990 \newcommand\indiii[4] []{{#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
1991 \newcommand\indiis[4] []{{#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}
1992 \newcommand\Indiii[4] []{{\captitalize{#2 #3 #4}}\@indiii[#1]{#2}{#3}{#4}}
1993 \newcommand\Indiis[4] []{{\capitalize{#2 #3 #4s}}\@indiii[#1]{#2}{#3}{#4}}
1994
1995 \newcommand\@indiv[5] []{\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
1996 \newcommand\aindiv[6] []{{#2}\@indiv[#1]{#3}{#4}{#5}{#6}}
1997 \newcommand\indiv[5] []{{#2 #3 #4 #5}\@indiv[#1]{#2}{#3}{#4}{#5}}
1998 \newcommand\indivs[5] []{{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
1999 \newcommand\Indiv[5] []{{\capitalize{#2 #3 #4 #5s}}\@indiv[#1]{#2}{#3}{#4}{#5}}
2000 \newcommand\Indivs[5] []{{\capitalize{#2 #3 #4 #5s}}\@indiv[#1]{#2}{#3}{#4}{#5}}

```

3.12.6 Miscellaneous

Some shortcuts that use math symbols but are not mathematical at all; in particular, they should not be translated by L^AT_EX_ML.

```

2001 \newcommand\hateq{\ensuremath{\widehat{=}}\xspace}
2002 \newcommand\hatequiv{\ensuremath{\widehat{\equiv}}\xspace}
2003 \@ifundefined{ergo}%
2004 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
2005 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
2006 \newcommand{\reflectsquig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%

```

```

2007 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
2008 \newcommand\notergo{\ensuremath{\not\leadsto}}
2009 \newcommand\notogre{\ensuremath{\not\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%

```

3.12.7 Deprecated Functionality

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

`*def*`

```

2010 \newcommand\indextoo[2] [] {\indi[#1]{#2}}%
2011 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}%
2012 \newcommand\indexalt[2] [] {\aindi[#1]{#2}}%
2013 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead}%
2014 \newcommand\twintoo[3] [] {\indii[#1]{#2}{#3}}%
2015 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}%
2016 \newcommand\twinalt[3] [] {\aindii[#1]{#2}{#3}}%
2017 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead}%
2018 \newcommand\atwintoo[4] [] {\indiii[#1]{#2}{#3}{#4}}%
2019 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead}%
2020 \newcommand\atwinalt[4] [] {\aindii[#1]{#2}{#3}{#4}}%
2021 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instead}%

```

`\my*graphics`

```

2022 \newcommand\mygraphics[2] [] {\includegraphics[#1]{#2}}%
2023 \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics}%
2024 \newcommand\mycgraphics[2] [] {\begin{center}\mygraphics[#1]{#2}\end{center}}%
2025 \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics}%
2026 \newcommand\mybgraphics[2] [] {\fbox{\mygraphics[#1]{#2}}}%
2027 \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics}%
2028 \newcommand\mycbgraphics[2] [] {\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}%
2029 \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphics}%

```

4 Things to deprecate

Module options:

```

2030 \addmetakey*{module}{id} % TODO: deprecate properly
2031 \addmetakey*{module}{load}
2032 \addmetakey*{module}{path}
2033 \addmetakey*{module}{dir}
2034 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
2035 \addmetakey*{module}{noalign}[true]
2036
2037 \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.

```

2038 %\srefaddidkey{symdef}% what does this do?
2039 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
2040 \define@key{symdef}{noverb}[all]{}%
2041 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
2042 \define@key{symdef}{specializes}{}%
2043 \addmetakey*{symdef}{noalign}[true]
2044 \define@key{symdef}{primary}[true]{}%
2045 \define@key{symdef}{assocarg}{}%
2046 \define@key{symdef}{bvars}{}%
2047 \define@key{symdef}{bargs}{}%
2048 \addmetakey{symdef}{lang}%
2049 \addmetakey{symdef}{prec}%
2050 \addmetakey{symdef}{arity}%
2051 \addmetakey{symdef}{variant}%
2052 \addmetakey{symdef}{ns}%
2053 \addmetakey{symdef}{args}%
2054 \addmetakey{symdef}{name}%
2055 \addmetakey*{symdef}{title}%
2056 \addmetakey*{symdef}{description}%
2057 \addmetakey{symdef}{subject}%
2058 \addmetakey*{symdef}{display}%
2059 \addmetakey*{symdef}{gfc}%

```

EdN:3

3

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

```

2060 \def\symdef{\@ifnextchar[\@symdef]{\@symdef[]}}%
2061 \def\@symdef[#1]#2{\@ifnextchar[\@@symdef[#1]{#2}]{\@@symdef[#1]{#2}[0]}}%

```

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

```

2062 \def\@@symdef[#1]#2[#3]{%
2063   \insymdef@true%
2064   \metasetkeys{symdef}{#1}%
2065   \edef\symdef@tmp@optpars{\ifcvoid{symdef@name}{[]}{[name=\symdef@name]}}%
2066   \expandafter\symdecl\symdef@tmp@optpars{#2}%
2067   \insymdef@false%
2068   \notation[#1]{#2}[#3]%
2069 }% mod@show
2070 \def\symdef@type{Symbol}%
2071 \providecommand{\stDMemph}[1]{\textbf{#1}}

```

³EDNOTE: MK@MK: we need to document the binder keys above.

`\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`.

```
2072 \def\symvariant#1{%
2073   \ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}%
2074   }%
2075 \def\@symvariant#1[#2]#3#4{%
2076   \notation[#3]{#1}[#2]{#4}%
2077 %\ignorespacesandpars
2078 }%
```

`\abbrdef` The `\abbrdef` macro is a variant of `\symdef` that does the same on the L^AT_EX level.

```
2079 \let\abbrdef\symdef%
```

`\@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.

```
2080 \newif\if@importing\@importingfalse
2081 \define@key{symi}{noverb}[all]{}%
2082 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
2083 \define@key{symi}{specializes}{}%
2084 \define@key{symi}{gfc}{}%
2085 \define@key{symi}{noalign}[true]{}%
2086 \newcommand\symi{\@ifstar\@symi@star\@symi}
2087 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
2088   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces
2089   }
2090 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
2091   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignorespaces
2092   }
2093 \newcommand\symii{\@ifstar\@symii@star\@symii}
2094 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
2095   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespaces
2096   }
2097 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
2098   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\ignorespaces
2099   }
2100 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
2101 \newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%
2102   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignorespaces
2103   }
2104 \newcommand\@symiii@star[4][]{\metasetkeys{symi}{#1}%
2105   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi\ignorespaces
2106   }
2107 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
2108 \newcommand\@symiv[5][]{\metasetkeys{symi}{#1}%
2109   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi\ignorespaces
2110   }
```

```

2111 \newcommand\@symiv@star[5] [] {\metasetkeys{syml}{#1}%
2112 \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4-#5}
2113 }

```

`\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`.

```

2114 %\srefaddidkey{importmhmodule}%
2115 \addmetakey{importmhmodule}{mhrepos}%
2116 \addmetakey{importmhmodule}{path}%
2117 \addmetakey{importmhmodule}{ext}% why does this exist?
2118 \addmetakey{importmhmodule}{dir}%
2119 \addmetakey[false]{importmhmodule}{conservative}[true]%
2120 \newcommand\importmhmodule[2] [] {%
2121 \parsemodule@maybesetcodes
2122 \metasetkeys{importmhmodule}{#1}%
2123 \ifx\importmhmodule@dir\@empty%
2124 \edef\@path{\importmhmodule@path}%
2125 \else\edef\@path{\importmhmodule@dir/#2}\fi%
2126 \ifx\@path\@empty% if module name is not set
2127 \@importmodule[] {#2}{export}%
2128 \else%
2129 \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
2130 \ifx\importmhmodule@mhrepos\@empty% if in the same repos
2131 \relax% no need to change mh@currentrepos, i.e, current directory.
2132 \else%
2133 \mathhub@setcurrentreposinfo\importmhmodule@mhrepos% change it.
2134 \addto@thismodule{x\{noexpand\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}}}%
2135 \fi%
2136 \@importmodule[\MathHub{\mh@currentrepos/source/\@path}] {#2}{export}%
2137 \mathhub@setcurrentreposinfo\mh@@repos% after importing, reset to old value
2138 \addto@thismodule{x\{noexpand\mathhub@setcurrentreposinfo{\mh@@repos}}}%
2139 \fi%
2140 %\ignorespacesandpars%
2141 }

```

`\usemhmodule`

```

2142 \addmetakey{importmhmodule}{load}
2143 \addmetakey{importmhmodule}{id}
2144 \addmetakey{importmhmodule}{dir}
2145 \addmetakey{importmhmodule}{mhrepos}
2146
2147 \addmetakey{importmodule}{load}
2148 \addmetakey{importmodule}{id}
2149
2150 \newcommand\usemhmodule[2] [] {%

```

```

2151 \metasetkeys{importmhmodule}{#1}%
2152 \ifx\importmhmodule@dir\@empty%
2153 \edef\@path{\importmhmodule@path}%
2154 \else\edef\@path{\importmhmodule@dir/#2}\fi%
2155 \ifx\@path\@empty%
2156 \usemodule[id=\importmhmodule@id]{#2}%
2157 \else%
2158 \edef\mh@@repos{\mh@currentrepos}%
2159 \ifx\importmhmodule@mhrepos\@empty%
2160 \else\mathhub@setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
2161 \usemodule{\@path\@QuestionMark#2}%
2162 %\usemodule[load=MathHub{\mh@currentrepos/source/\@path},
2163 % id=\importmhmodule@id]{#2}%
2164 \mathhub@setcurrentreposinfo\mh@@repos%
2165 \fi%
2166 %\ignorespacesandpars
2167 }

\mhinputref
2168 \newcommand\mhinputref[2][]{%
2169 \edef\mhinputref@first{#1}%
2170 \ifx\mhinputref@first\@empty%
2171 \inputref{#2}%
2172 \else%
2173 \inputref[mhrepos=\mhinputref@first]{#2}%
2174 \fi%
2175 }

\trefi*
2176 \newcommand\trefi[2][]{%
2177 \edef\trefi@mod{#1}%
2178 \ifx\trefi@mod\@empty\tref{#2}\else\tref{#1\@QuestionMark#2}\fi%
2179 }
2180 \newcommand\trefii[3][]{%
2181 \edef\trefi@mod{#1}%
2182 \ifx\trefi@mod\@empty\tref{#2-#3}\else\tref{#1\@QuestionMark#2-#3}\fi%
2183 }

\defi*
2184 \def\defii#1#2{\defi{#1!#2}}
2185 \def\Defii#1#2{\Defi{#1!#2}}
2186 \def\defiis#1#2{\defis{#1!#2}}
2187 \def\Defiis#1#2{\Defis{#1!#2}}
2188 \def\defiii#1#2#3{\defi{#1!#2!#3}}
2189 \def\Defiii#1#2#3{\Defi{#1!#2!#3}}
2190 \def\defiiis#1#2#3{\defis{#1!#2!#3}}
2191 \def\Defiiis#1#2#3{\Defis{#1!#2!#3}}
2192 \def\defiv#1#2#3#4{\defi{#1!#2!#3!#4}}
2193 \def\Defiv#1#2#3#4{\Defi{#1!#2!#3!#4}}

```

```
2194 \def\defivs#1#2#3#4{\defis{#1!#2!#3!#4}}
2195 \def\Defivs#1#2#3#4{\Defis{#1!#2!#3!#4}}
2196 \def\adefi#1#2{\defi[name=#2]{#1}}
2197 \def\adefii#1#2#3{\defi[name=#2-#3]{#1}}
2198 \def\adefiii#1#2#3#4{\defi[name=#2-#3-#4]{#1}}
2199 \def\adefiv#1#2#3#4#5{\defi[name=#2-#3-#4-#5]{#1}}
```