

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

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

November 17, 2020

## **Abstract**

TODO

---

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

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>User commands</b>	<b>3</b>
<b>3</b>	<b>Implementation</b>	<b>3</b>
3.1	sTeX base . . . . .	3
3.2	Paths and URIs . . . . .	4
3.3	Modules . . . . .	15
3.4	Inheritance . . . . .	19
3.5	Symbols and Notations . . . . .	27
3.6	Term References . . . . .	39
3.7	sref . . . . .	40
3.8	smultiling . . . . .	42
3.9	smglom . . . . .	42
3.10	mathhub . . . . .	43
3.11	omdoc/omgroup . . . . .	43
3.12	omtext . . . . .	46
<b>4</b>	<b>Mathematical Text</b>	<b>46</b>
<b>5</b>	<b>Phrase-level Markup</b>	<b>47</b>
<b>6</b>	<b>Declarations (under development)</b>	<b>48</b>
<b>7</b>	<b>Block-Level Markup</b>	<b>48</b>
<b>8</b>	<b>Index Markup</b>	<b>49</b>
<b>9</b>	<b>Miscellaneous</b>	<b>51</b>
<b>10</b>	<b>Deprecated Functionality</b>	<b>51</b>
<b>11</b>	<b>Things to deprecate</b>	<b>51</b>

# 1 Introduction

TODO

## 2 User commands

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

## 3 Implementation

```
1 \langle *package\rangle
2 % TODO
3 \newif\if@modules@html@\@modules@html@true
4 \DeclareOption{omdocmode}{\@modules@html@false}
5 % Modules:
6 \newif\ifmod@show\mod@showfalse
7 \DeclareOption{showmods}{\mod@showtrue}
8 % sref:
9 \newif\ifextrefs\extrefsfalse
10 \DeclareOption{extrefs}{\extrefstrue}
11 %
12 \ProcessOptions
13 \RequirePackage{standalone}
14 \RequirePackage{xspace}
15 \RequirePackage{metakeys}
```

### 3.1 sTeX base

The sTeX logo:

```

16 \protected\def\stex{%
17   \@ifundefined{texorpdfstring}%
18   {\let\texorpdfstring\@firstoftwo}%
19   }%
20   \texorpdfstring{\raisebox{-.5ex}{S}\kern-.5ex\TeX}{sTeX}\xspace%
21 }
22 \def\sTeX{\stex}

```

and a conditional for LaTeXML:

```

23 \newif\if@latexml\@latexmlfalse

```

## 3.2 Paths and URIs

```

24 \RequirePackage{xstring}
25 \RequirePackage{etoolbox}

```

`\defpath` `\defpath[optional argument]{macro name}{base path}` defines a new macro which can take another path to form one integrated path. For example, `\MathHub` in every `localpaths.tex` 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`.

```

26 \newrobustcmd\defpath[3][]{%
27   \expandafter\newcommand\csname #2\endcsname[1]{#3/#1}%
28 }%

```

### 3.2.1 Path Canonicalization

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

```

29 \def\pathsuris@setcatcodes{%
30   \edef\pathsuris@oldcatcode@hash{\the\catcode'\#}%
31   \catcode'\#=12\relax%
32   \edef\pathsuris@oldcatcode@slash{\the\catcode'\/%}
33   \catcode'\/=12\relax%
34   \edef\pathsuris@oldcatcode@colon{\the\catcode'\:%}
35   \catcode'\:=12\relax%
36   \edef\pathsuris@oldcatcode@qm{\the\catcode'\?}%
37   \catcode'\?=12\relax%
38 }
39 \def\pathsuris@resetcatcodes{%
40   \catcode'\#\pathsuris@oldcatcode@hash\relax%
41   \catcode'\%\pathsuris@oldcatcode@slash\relax%
42   \catcode'\:\pathsuris@oldcatcode@colon\relax%
43   \catcode'\?\pathsuris@oldcatcode@qm\relax%
44 }

```

We define some macros for later comparison.

```

45 \def\@ToTop{..}
46 \def\@Slash{/}
47 \def\@Colon{:}
48 \def\@Space{ }
49 \def\@QuestionMark{?}
50 \def\@Dot{.}
51 \catcode'\&=12
52 \def\@Ampersand{&}
53 \catcode'\&=4
54 \pathsuris@setcatcodes
55 \def\@Fragment{#}
56 \pathsuris@resetcatcodes
57 \catcode'\.=0
58 .catcode'\.=12
59 .let.\@BackSlash\
60 .catcode'\.=0
61 \catcode'\.=12
62 \edef\old@percent@catcode{\the\catcode'\%}
63 \catcode'\%=12
64 \let\@Percent%
65 \catcode'\%=\old@percent@catcode

```

\@cpath Canonicalizes (file) paths:

```

66 \def\@cpath#1{%
67   \edef\pathsuris@cpath@temp{#1}%
68   \def\@CanPath{%
69     \IfBeginWith\pathsuris@cpath@temp\@Slash{%
70       \@cpath@loop%
71       \edef\@CanPath{\@Slash\@CanPath}%
72     }{%
73       \IfBeginWith\pathsuris@cpath@temp{\@Dot\@Slash}{%
74         \StrGobbleLeft\pathsuris@cpath@temp2[\pathsuris@cpath@temp]%
75         \@cpath@loop%
76       }{%
77         \ifx\pathsuris@cpath@temp\@Dot\else%
78         \@cpath@loop\fi%
79       }%
80     }%
81     \IfEndWith\@CanPath\@Slash{%
82       \ifx\@CanPath\@Slash\else%
83       \StrGobbleRight\@CanPath1[\@CanPath]%
84       \fi%
85     }{}%
86 }
87
88 \def\@cpath@loop{%
89   \IfSubStr\pathsuris@cpath@temp\@Slash{%
90     \StrCut\pathsuris@cpath@temp\@Slash\pathsuris@cpath@temp@a\pathsuris@cpath@temp%

```

```

91     \ifx\pathsuris@cpath@temp@a\@ToTop%
92         \ifx\@CanPath\@empty%
93             \edef\@CanPath{\@ToTop}%
94         \else%
95             \edef\@CanPath{\@CanPath\@Slash\@ToTop}%
96         \fi%
97         \@cpath@loop%
98     \else%
99     \ifx\pathsuris@cpath@temp@a\@Dot%
100         \@cpath@loop%
101     \else%
102     \IfBeginWith\pathsuris@cpath@temp\@ToTop{%
103         \StrBehind{\pathsuris@cpath@temp}{\@ToTop}[\pathsuris@cpath@temp]%
104         \IfBeginWith\pathsuris@cpath@temp\@Slash{%
105             \edef\pathsuris@cpath@temp{\@CanPath\pathsuris@cpath@temp}%
106         }{%
107             \ifx\@CanPath\@empty\else%
108                 \edef\pathsuris@cpath@temp{\@CanPath\@Slash\pathsuris@cpath@temp}%
109             \fi%
110         }%
111         \def\@CanPath{}%
112         \@cpath@loop%
113     }{%
114         \ifx\@CanPath\@empty%
115             \edef\@CanPath{\pathsuris@cpath@temp@a}%
116         \else%
117             \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp@a}%
118         \fi%
119         \@cpath@loop
120     }%
121     \fi\fi%
122 }{
123     \ifx\@CanPath\@empty%
124         \edef\@CanPath{\pathsuris@cpath@temp}%
125     \else%
126         \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp}%
127     \fi%
128 }%
129 }

```

**Test:**

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

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

```

130 \newcommand\cpath[1]{%
131   \@cpath{#1}%
132   \@CanPath%
133 }
```

`\path@filename`

```

134 \def\path@filename#1#2{%
135   \edef\filename@oldpath{#1}%
136   \StrCount\filename@oldpath{@Slash[\filename@lastslash]}%
137   \ifnum\filename@lastslash>0%
138     \StrBehind[\filename@lastslash]\filename@oldpath{@Slash[\filename@oldpath]}%
139     \edef#2{\filename@oldpath}%
140   \else%
141     \edef#2{\filename@oldpath}%
142   \fi%
143 }
```

**Test:**

Path: /foo/bar/baz.tex

Filename: baz.tex

### 3.2.2 Windows

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

```

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

**Test:**

We are on windows: no.

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

```

146 \newif\if@windowstopath@inpath@
147 \def\windows@to@path#1{
```

```

148 \@windowstopath@inpath@false
149 \def\windows@temp{}
150 \edef\windows@path{#1}
151 \ifx\windows@path\@empty\else
152 \expandafter\windows@path@loop\windows@path\windows@path@end
153 \fi
154 \let#1\windows@temp
155 }
156 \def\windows@path@loop#1#2\windows@path@end{
157 \def\windows@temp@b{#2}
158 \ifx\windows@temp@b\@empty
159 \def\windows@continue{}
160 \else
161 \def\windows@continue{\windows@path@loop#2\windows@path@end}
162 \fi
163 \if@windowstopath@inpath@
164 \ifx#1\@BackSlash
165 \edef\windows@temp{\windows@temp\@Slash}
166 \else
167 \edef\windows@temp{\windows@temp#1}
168 \fi
169 \else
170 \ifx#1:
171 \edef\windows@temp{\@Slash\windows@temp}
172 \@windowstopath@inpath@true
173 \else
174 \edef\windows@temp{\windows@temp#1}
175 \fi
176 \fi
177 \windows@continue
178 }

```

#### Test:

Input: C:\foo \bar .baz

Output: /C/foo/bar.baz

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

```

179 \def\path@to@windows#1{
180 \@windowstopath@inpath@false
181 \def\windows@temp{}
182 \edef\windows@path{#1}
183 \edef\windows@path{\expandafter\@gobble\windows@path}
184 \ifx\windows@path\@empty\else
185 \expandafter\path@windows@loop\windows@path\windows@path@end
186 \fi
187 \let#1\windows@temp
188 }
189 \def\path@windows@loop#1#2\windows@path@end{
190 \def\windows@temp@b{#2}
191 \ifx\windows@temp@b\@empty

```



```

192     \def\windows@continue{}
193   \else
194     \def\windows@continue{\path@windows@loop#2\windows@path@end}
195   \fi
196   \if@windowstopath@inpath@
197     \ifx#1/
198       \edef\windows@temp{\windows@temp\@BackSlash}
199     \else
200       \edef\windows@temp{\windows@temp#1}
201     \fi
202   \else
203     \ifx#1/
204       \edef\windows@temp{\windows@temp:\@BackSlash}
205       \@windowstopath@inpath@true
206     \else
207       \edef\windows@temp{\windows@temp#1}
208     \fi
209   \fi
210   \windows@continue
211 }

```

**Test:**

Input: /C/fo0/bar.baz

Output: C:\fo0\bar.baz

### 3.2.3 Auxiliary methods

**\trimstring** Removes initial and trailing spaces from a string:

```

212 \def\trimstring#1{%
213   \edef\pathsuris@trim@temp{#1}%
214   \IfBeginWith\pathsuris@trim@temp\@Space{%
215     \StrGobbleLeft\pathsuris@trim@temp1[#1]%
216     \trimstring{#1}%
217   }{%
218     \IfEndWith\pathsuris@trim@temp\@Space{%
219       \StrGobbleRight\pathsuris@trim@temp1[#1]%
220       \trimstring{#1}%
221     }{%
222       \edef#1{\pathsuris@trim@temp}%
223     }%
224   }%
225 }

```

**Test:**

>bla blubb<

**\kpsewhich** Calls kpsewhich to get e.g. system variables:

```

226 \def\kpsewhich#1#2{\begingroup%
227   \edef\kpsewhich@cmd{"|kpsewhich #2"%
228   \everyeof{\noexpand}%

```

```

229 \catcode'\=12%
230 \edef#1{\@input\kpsewhich@cmd\@Space}%
231 \trimstring#1%
232 \if@iswindows@ \windows@to@path#1\fi%
233 \xdef#1{\expandafter\detokenize\expandafter{#1}}%
234 \endgroup}

```

**Test:**

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

### 3.2.4 sTeX input hooks

We determine the PWD of the current main document:

```

235 \edef\pwd@cmd{\if@iswindows@ -expand-var \percent CD\percent\else -var-value PWD\fi}
236 \kpsewhich\stex@maindir\pwd@cmd
237 \edef\stex@mainfile{\stex@maindir\@Slash\jobname}
238 \edef\stex@mainfile{\expandafter\detokenize\expandafter{\stex@mainfile}}

```

**Test:**

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

We keep a stack of \inputed files:

```

239 \def\stex@currfile@stack{}
240
241 \def\stex@currfile@push#1{%
242   \edef\stex@temppath{#1}%
243   \edef\stex@temppath{\expandafter\detokenize\expandafter{\stex@temppath}}%
244   \edef\stex@currfile@stack{\stex@currfile\ifx\stex@currfile@stack\@empty\else,\stex@currfile@stack}%
245   \IfBeginWith\stex@temppath\@Slash{\@cpath{\stex@temppath}}{%
246     \@cpath{\stex@maindir\@Slash#1}%
247   }
248   \let\stex@currfile\@CanPath%
249   \path@filename\stex@currfile\stex@currfilename%
250   \StrLen\stex@currfilename[\stex@currfile@tmp]%
251   \StrGobbleRight\stex@currfile{\the\numexpr\stex@currfile@tmp+1 }[\stex@currpath]%
252   \global\let\stex@currfile\stex@currfile%
253   \global\let\stex@currpath\stex@currpath%
254   \global\let\stex@currfilename\stex@currfilename%
255 }
256 \def\stex@currfile@pop{%
257   \ifx\stex@currfile@stack\@empty%
258     \global\let\stex@currfile\stex@mainfile%
259     \global\let\stex@currpath\stex@maindir%
260     \global\let\stex@currfilename\jobname%
261   \else%
262     \StrCut\stex@currfile@stack,\stex@currfile\stex@currfile@stack%
263     \path@filename\stex@currfile\stex@currfilename%
264     \StrLen\stex@currfilename[\stex@currfile@tmp]%
265     \StrGobbleRight\stex@currfile{\the\numexpr\stex@currfile@tmp+1 }[\stex@currpath]%
266     \global\let\stex@currfile\stex@currfile%
267     \global\let\stex@currpath\stex@currpath%

```

```

268     \global\let\stex@currfilename\stex@currfilename%
269     \fi%
270 }

```

**\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:

```

271 \def\stexinput#1{%
272     \stexiffileexists{#1}{%
273         \stex@currfile@push\stex@temp@path%
274         \input{\stex@currfile}%
275         \stex@currfile@pop%
276     }%
277     {%
278         \PackageError{stex}{File does not exist (#1): \stex@temp@path}{}%
279     }%
280 }
281 \def\stexiffileexists#1#2#3{%
282     \edef\stex@temp@path{#1}%
283     \if@iswindows@\path@to@windows\stex@temp@path\fi%
284     \IfFileExists\stex@temp@path{#2}{#3}%
285 }
286 \stex@currfile@pop

```

**Test:**

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

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

### 3.2.5 MathHub repositories

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

```

287 \kpsewhich\mathhub@path{--var-value MATHHUB}
288 \if@iswindows@\windows@to@path\mathhub@path\fi
289 \ifx\mathhub@path\@empty%
290     \PackageWarning{stex}{MATHHUB system variable not found or wrongly set}{%
291         \defpath{MathHub}{%
292             \else\defpath{MathHub}\mathhub@path\fi

```

**Test:**

</home/jazzpirate/work/MathHub>

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

```

293 \def\findmanifest#1{
294     \@cpath{#1}
295     \ifx\@CanPath\@Slash
296         \def\manifest@mf{}
297     \else\ifx\@CanPath\@empty
298         \def\manifest@mf{}
299     \else
300         \edef\@findmanifest@path{\@CanPath/MANIFEST.MF}

```

```

301 \if@iswindows@path@to@windows\@findmanifest@path\fi
302 \IfFileExists{\@findmanifest@path}{
303   \%message{MANIFEST.MF found at \@findmanifest@path}
304   \edef\manifest@mf{\@findmanifest@path}
305   \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
306 }{
307   \edef\@findmanifest@path{\@CanPath/META-INF/MANIFEST.MF}
308   \if@iswindows@path@to@windows\@findmanifest@path\fi
309   \IfFileExists{\@findmanifest@path}{
310     \%message{MANIFEST.MF found at \@findmanifest@path}
311     \edef\manifest@mf{\@findmanifest@path}
312     \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
313   }{
314     \edef\@findmanifest@path{\@CanPath/meta-inf/MANIFEST.MF}
315     \if@iswindows@path@to@windows\@findmanifest@path\fi
316     \IfFileExists{\@findmanifest@path}{
317       \%message{MANIFEST.MF found at \@findmanifest@path}
318       \edef\manifest@mf{\@findmanifest@path}
319       \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
320     }{
321       \findmanifest{\@CanPath/..}
322     }
323   \fi\fi
324 }

```

#### Test:

</home/jazzpirate/work/MathHub/smgglom/mv/META-INF/MANIFEST.MF>

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

```

325 \def\split@manifest@key{
326   \IfSubStr{\manifest@line}{\@Colon}{
327     \StrBefore{\manifest@line}{\@Colon}[\manifest@key]
328     \StrBehind{\manifest@line}{\@Colon}[\manifest@line]
329     \trimstring\manifest@line
330     \trimstring\manifest@key
331   }{
332     \def\manifest@key{}
333   }
334 }

```

the next helper function iterates over lines in MANIFEST.MF

```

335 \def\parse@manifest@loop{
336   \ifeof\@manifest
337   \else
338     \read\@manifest to \manifest@line\relax
339     \edef\manifest@line{\expandafter\detokenize\expandafter{\manifest@line}}
340     \split@manifest@key
341     % id
342     \IfStrEq\manifest@key{\detokenize{id}}{
343       \xdef\manifest@mf{id}\manifest@line}

```

```

344     }{
345     % narration-base
346     \IfStrEq\manifest@key{\detokenize{narration-base}}{
347         \xdef\manifest@mf@narr{\manifest@line}
348     }{
349     % namespace
350     \IfStrEq\manifest@key{\detokenize{source-base}}{
351         \xdef\manifest@mf@ns{\manifest@line}
352     }{
353     \IfStrEq\manifest@key{\detokenize{ns}}{
354         \xdef\manifest@mf@ns{\manifest@line}
355     }{
356     % dependencies
357     \IfStrEq\manifest@key{\detokenize{dependencies}}{
358         \xdef\manifest@mf@deps{\manifest@line}
359     }{
360     }}}}
361     \parse@manifest@loop
362 \fi
363 }

```

`\parsemanifest` `\parsemanifest{<macroname>}{<path>}` finds MANIFEST.MF via `\findmanifest{<path>}`, and parses the file, storing the individual fields (id, narr, ns and dependencies) in `<macroname>id`, `<macroname>narr`, etc.

```

364 \newread\@manifest
365 \def\parsemanifest#1#2{%
366     \gdef\temp@archive@dir{}%
367     \findmanifest{#2}%
368     \begingroup%
369         \gdef\manifest@mf@id{}%
370         \gdef\manifest@mf@narr{}%
371         \gdef\manifest@mf@ns{}%
372         \gdef\manifest@mf@deps{}%
373         \openin\@manifest\manifest@mf%
374         \parse@manifest@loop%
375         \closein\@manifest%
376     \endgroup%
377     \if@iswindows@ \windows@to@path\manifest@mf\fi%
378     \cslet{#1id}\manifest@mf@id%
379     \cslet{#1narr}\manifest@mf@narr%
380     \cslet{#1ns}\manifest@mf@ns%
381     \cslet{#1deps}\manifest@mf@deps%
382     \ifcvoid\manifest@mf@id{-}%
383     \cslet{#1dir}\temp@archive@dir%
384 }%
385 }

```

**Test:**

id: FOO/BAR

ns: <http://mathhub.info/FOO/BAR>

dir: FOO

`\setcurrentreposinfo` `\setcurrentreposinfo{⟨id⟩}` sets the current repository to `⟨id⟩`, checks if the MANIFEST.MF of this repository has already been read, and if not, find it, parses it and stores the values in `\currentrepos@⟨key⟩@⟨id⟩` for later retrieval.

```

386 \def\setcurrentreposinfo#1{%
387   \edef\mh@currentrepos{#1}%
388   \ifx\mh@currentrepos\@empty%
389     \edef\currentrepos@dir{\@Dot}%
390     \def\currentrepos@narr{}%
391     \def\currentrepos@ns{}%
392     \def\currentrepos@id{}%
393     \def\currentrepos@deps{}%
394   \else%
395   \ifcsdef{mathhub@dir@\mh@currentrepos}{%
396     \@inmhrepostrue
397     \edef\mh@currentrepos{#1}%
398     \expandafter\let\expandafter\currentrepos@dir\csname mathhub@dir@#1\endcsname%
399     \expandafter\let\expandafter\currentrepos@narr\csname mathhub@narr@#1\endcsname%
400     \expandafter\let\expandafter\currentrepos@ns\csname mathhub@ns@#1\endcsname%
401     \expandafter\let\expandafter\currentrepos@deps\csname mathhub@deps@#1\endcsname%
402   }{%
403     \parsemanifest{currentrepos@}\MathHub{#1}}%
404     \@setcurrentreposinfo%
405     \ifcsvoid{currentrepos@dir}\PackageError{stex}{No archive with %
406       name #1 found!}{make sure that #1 is directly in your MATHHUB folder %
407       and contains a MANIFEST.MF, either directly in #1 or in a meta-inf %
408       subfolder.}}{\@inmhrepostrue}%
409   }%
410   \fi%
411 }
412
413 \def\@setcurrentreposinfo{%
414   \edef\mh@currentrepos{\currentrepos@id}%
415   \ifcsvoid{currentrepos@dir}{%
416     \csxdef{mathhub@dir@\currentrepos@id}{\currentrepos@dir}%
417     \csxdef{mathhub@narr@\currentrepos@id}{\currentrepos@narr}%
418     \csxdef{mathhub@ns@\currentrepos@id}{\currentrepos@ns}%
419     \csxdef{mathhub@deps@\currentrepos@id}{\currentrepos@deps}%
420   }%
421 }

```

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

```

422 \newif\if@inmhrepos\@inmhreposfalse
423 \ifcsvoid{stex@maindir}{%
424   \parsemanifest{currentrepos@}\stex@maindir
425   \@setcurrentreposinfo
426   \ifcsvoid{currentrepos@dir}\PackageWarning{stex}{Not currently in a MathHub repository}{%
427     \message{Current repository: \mh@currentrepos}

```

```
428 }
429 }
```

### 3.3 Modules

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

Aux:

```
431 \def\ignorespacesandpars{\begingroup\catcode13=10\ifnextchar\relax{\endgroup}{\endgroup}}
```

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

```
432 \def\ignorespacesandparsafterend#1\ignorespaces\fi{#1\fi\ignorespacesandpars}
```

```
433 \def\ignorespacesandpars{\ifhmode\unskip\fi\ifnextchar\par{\expandafter\ignorespacesandpars\@g
```

Options for the module-environment:

```
434 \addmetakey*{module}{title}
```

```
435 \addmetakey*{module}{name}
```

```
436 \addmetakey*{module}{creators}
```

```
437 \addmetakey*{module}{contributors}
```

```
438 \addmetakey*{module}{srccite}
```

```
439 \addmetakey*{module}{ns}
```

```
440 \addmetakey*{module}{narr}
```

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

```
441 \ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%
```

```
442 \newrobustcmd\module@heading{%
```

```
443 \stepcounter{module}%
```

```
444 \ifmod@show%
```

```
445 \noindent{\textbf{Module} \thesection.\thetitle [\module@name]]%
```

```
446 \sref@label@id{Module \thesection.\thetitle [\module@name]]%
```

```
447 \ifx\module@title\empty : \quad\else\quad(\module@title)\hfill\\\fi%
```

```
448 \fi%
```

```
449 }%
```

**Test:**

**Module 3.1[Test]: Foo**

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

```
450 \newenvironment{module}[1][{}]{%
```

```
451 \begin{@module}[#1]%
```

```
452 \module@heading% make the headings
```

```
453 \ignorespacesandpars\parsemodule@maybesetcodes}{%
```

```
454 \end{@module}%
```

```
455 \ignorespacesafterend%
```

```
456 }%
```

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

Some auxiliary methods:

```
458 \def\g@addto@macro@safe#1#2{\ifx#1\relax\def#1{}\fi\g@addto@macro#1{#2}}
```

```
459 \def\addto@thismodule#1{%
```

```

460 \@ifundefined{this@module}{}{%
461   \expandafter\g@addto@macro@safe\this@module{#1}%
462 }%
463 }
464 \def\addto@thismodule#1{%
465 \ifundefined{this@module}{}{%
466   \edef\addto@thismodule@exp{#1}%
467   \expandafter\expandafter\expandafter\g@addto@macro@safe%
468   \expandafter\this@module\expandafter{\addto@thismodule@exp}%
469 }}

```

**@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`.

```

470 \newif\ifarchive@ns@empty@\archive@ns@empty@false
471 \def\set@default@ns{%
472   \edef\@module@ns@temp{\stex@currpath}%
473   \if@iswindows@\windows@to@path\@module@ns@temp\fi%
474   \archive@ns@empty@false%
475   \ifcvoid{mh@currentrepos}{\archive@ns@empty@true}%
476   {\expandafter\ifx\curname mathhub@ns@\mh@currentrepos\endcsname\@empty\archive@ns@empty@true\fi%
477   }%
478   \ifarchive@ns@empty%
479     \edef\@module@ns@tempuri{file\@Colon\@Slash\@Slash\@module@ns@temp}%
480   \else%
481     \edef\@module@filepath@temppath{\@module@ns@temp}%
482     \edef\@module@ns@tempuri{\curname mathhub@ns@\mh@currentrepos\endcsname}%
483     \edef\@module@archivedirpath{\curname mathhub@dir@\mh@currentrepos\endcsname\@Slash source}%
484     \edef\@module@archivedirpath{\expandafter\detokenize\expandafter{\@module@archivedirpath}}%
485     \IfBeginWith\@module@filepath@temppath\@module@archivedirpath{%
486       \StrLen\@module@archivedirpath[\ns@temp@length]%
487       \StrGobbleLeft\@module@filepath@temppath\@module@filepath@temprest}%
488     \edef\@module@ns@tempuri{\@module@ns@tempuri\@module@filepath@temprest}%
489   }{}%
490 \fi%
491 \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]}%
492 \setkeys{module}{ns=\@module@ns@tempuri}%
493 }

```

#### Test:

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



If the module is not given a name, `\set@next@moduleid` computes one by enumeration, e.g. `module0`, `module1`, etc.

```

494 \def\set@next@moduleid{%
495   \unless\ifcsname namespace@\module@ns @unnamedmodules\endcsname%
496     \csgdef{namespace@\module@ns @unnamedmodules}{0}%
497   \fi%
498   \edef\namespace@currnum{\csname namespace@\module@ns @unnamedmodules\endcsname}%
499   \edef\module@temp@setidname{\noexpand\setkeys{module}{name=module\namespace@currnum}}%
500   \module@temp@setidname%
501   \csxdef{namespace@\module@ns @unnamedmodules}{\the\numexpr\namespace@currnum+1}%
502 }

```

#### Test:

`module0`

`module1`

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

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

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

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.

```

503 \newenvironment{@module}[1][[]]{%
504   \metasetkeys{module}{#1}%
505   \ifcsvoid{module@name}{\let\module@name\module@id}{}% % TODO deprecate
506   \ifx\module@ns\@empty\set@default@ns\fi%
507   \ifx\module@narr\@empty%
508     \setkeys{module}{narr=\module@ns}%
509   \fi%
510   \ifcsvoid{module@name}{\set@next@moduleid}{}%
511   \let\module@id\module@name% % TODO deprecate
512   \edef\module@uri{\module@ns\@QuestionMark\module@name}%

```

```

513 \csgdef{module@names@\module@uri}{}%
514 \csgdef{module@imports@\module@uri}{}%
515 \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
516 \expandafter\global\expandafter\let\csname Module\module@name\expandafter\endcsname\csname\mo
517 \edef\this@module{%
518   \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
519 }%
520 \csdef{module@defs@\module@uri}{}%
521 \ifcvoid{mh@currentrepos}{}{%
522   \@inmhrepostrue%
523   \addto@thismodule{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\en
524     {\noexpand\mh@currentrepos}}%
525   \addto@thismodule{\noexpand\setcurrentreposinfo{\mh@currentrepos}}%
526 }%
527 }{%
528   \if@inmhrepos%
529   \@inmhreposfalse%
530   \addto@thismodule{\noexpand\setcurrentreposinfo{\expandafter\noexpand\csname mh@old@repos@\m
531   \fi%
532 }%

```

**Test:**

**Module 3.2[Foo]:**

Name: Foo

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

this@module: macro:->

**Test:**

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

**Module 3.3[Foo2]:**

Name: Foo2

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

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

**Test:**

Removing the /home/jazzpirate/work/MathHub/ system variable first:

**Module 3.4[Foo]:**

Name: Foo

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

this@module: macro:->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  
}\setcurrentreposinfo {Foo/Bar}

A module with URI  $\langle uri \rangle$  and id  $\langle id \rangle$  creates two macros  $\langle uri \rangle$  and  $\langle Module \langle id \rangle \rangle$ , that ultimately expand to  $\@invoke@module{\langle uri \rangle}$ . Currently, the only functionality is  $\@invoke@module{\langle uri \rangle}\@URI$ , which expands to the full uri of a module (i.e. via  $\langle Module \langle id \rangle \rangle \@URI$ ). In the future, this macro can be

extended with additional functionality, e.g. accessing symbols in a macro for overloaded (macro-)names.

```

533 \def\@URI{uri}
534 \def\@invoke@module#1#2{%
535   \ifx\@URI#2%
536     #1%
537   \else%
538     % TODO something else
539     #2%
540   \fi%
541 }

```

## 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 `sTeX` macros and environments as part of a module signature.

```

542 \newif\if@smsmode\@smsmodefalse
543 \def\parsemodule@escapechar@allowed{true}
544 \def\parsemodule@allow#1{
545   \expandafter\let\csname parsemodule@allowedmacro@#1\endcsname\parsemodule@escapechar@allowed
546 }
547 \def\parsemodule@allowenv#1{
548   \expandafter\let\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed
549 }
550 \def\parsemodule@escapechar@beginstring{begin}
551 \def\parsemodule@escapechar@endstring{end}

```

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

```

552 \parsemodule@allow{symdef}
553 \parsemodule@allow{abbrdef}
554 \parsemodule@allow{importmodule}
555 \parsemodule@allowenv{module}
556 \parsemodule@allow{importmhmodule}
557 \parsemodule@allow{gimport}
558 \parsemodule@allowenv{modsig}
559 \parsemodule@allowenv{mhmodsig}
560 \parsemodule@allowenv{mhmodnl}

```

```

561 \parsemodule@allowenv{modnl}
562 \parsemodule@allow{symvariant}
563 \parsemodule@allow{symi}
564 \parsemodule@allow{symii}
565 \parsemodule@allow{symiii}
566 \parsemodule@allow{symiv}
567 \parsemodule@allow{notation}
568 \parsemodule@allow{symdecl}
569 %\parsemodule@allow{defi}
570 %\parsemodule@allow{defii}
571 %\parsemodule@allow{defiii}
572 %\parsemodule@allow{defiv}
573 %\parsemodule@allow{adefi}
574 %\parsemodule@allow{adefii}
575 %\parsemodule@allow{adefiii}
576 %\parsemodule@allow{adefiv}
577 %\parsemodule@allow{defis}
578 %\parsemodule@allow{defiis}
579 %\parsemodule@allow{defiiis}
580 %\parsemodule@allow{defivs}
581 %\parsemodule@allow{Defi}
582 %\parsemodule@allow{Defii}
583 %\parsemodule@allow{Defiii}
584 %\parsemodule@allow{Defiv}
585 %\parsemodule@allow{Defis}
586 %\parsemodule@allow{Defiis}
587 %\parsemodule@allow{Defiiis}
588 %\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*).

```

589 \catcode'\.=0
590 .catcode'\.=13
591 .def.\@active@slash{\}
592 .catcode'\.<=1
593 .catcode'\.>=2
594 .catcode'\.{=12
595 .catcode'\.=12
596 .def.\@open@brace<{>
597 .def.\@close@brace<}>
598 .catcode'\.=0
599 \catcode'\.=12
600 \catcode'\{=1

```

```

601 \catcode'\}=2
602 \catcode'\<=12
603 \catcode'\>=12

```

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

`\set@parsemodule@catcodes`

```

604 \def\set@parsemodule@catcodes{%
605     \global\catcode'\=13%
606     \global\catcode'\#=12%
607     \global\catcode'\{=12%
608     \global\catcode'\}=12%
609     \global\catcode'\$=12%$
610     \global\catcode'\^=12%
611     \global\catcode'\_ =12%
612     \global\catcode'\&=12%
613     \expandafter\let\@active@slash\parsemodule@escapechar%
614 }

```

`\reset@parsemodule@catcodes`

```

615 \def\reset@parsemodule@catcodes{%
616     \global\catcode'\=0%
617     \global\catcode'\#=6%
618     \global\catcode'\{=1%
619     \global\catcode'\}=2%
620     \global\catcode'\$=3%$
621     \global\catcode'\^=7%
622     \global\catcode'\_ =8%
623     \global\catcode'\&=4%
624 }

```

`\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.

```

625 \def\parsemodule@maybesetcodes{%
626     \if@smsmode\set@parsemodule@catcodes\fi%
627 }

```

`\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:

```

628

```

```

629 \def\parsemodule@escapechar{%
630   \def\parsemodule@escape@currcls{}%
631   \parsemodule@escape@parse@nextchar%
632 }%

```

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@checkcs`.

```

633 \long\def\parsemodule@escape@parse@nextchar@#1{%
634   \ifcat a#1\relax%
635     \edef\parsemodule@escape@currcls{\parsemodule@escape@currcls#1}%
636     \let\parsemodule@do@next\parsemodule@escape@parse@nextchar%
637   \else%
638     \def\parsemodule@last@char{#1}%
639     \def\parsemodule@do@next{\parsemodule@escapechar@checkcs}%
640   \fi%
641   \parsemodule@do@next%
642 }

```

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.

```

643 \def\parsemodule@escapechar@checkcs{%
644   \ifx\parsemodule@escape@currcls\parsemodule@escapechar@beginstring%
645     \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@last@char}%
646   \else%
647     \ifx\parsemodule@escape@currcls\parsemodule@escapechar@endstring%
648       \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@last@char}%
649     \else%
650       \expandafter\ifx\csname parsemodule@allowedmacro\endcsname\parsemodule@escape@currcls\endcsname%
651         \parsemodule@escapechar@allowed%
652       \ifx\parsemodule@last@char\@open@brace%
653         \expandafter\let\expandafter\parsemodule@do@next\ii\csname\parsemodule@escape@currcls\endcsname%
654       \edef\parsemodule@do@next{\noexpand\parsemodule@converttoproperbraces\@open@brace}%
655     \else%
656       \reset@parsemodule@catcodes%
657       \edef\parsemodule@do@next{\expandafter\noexpand\csname\parsemodule@escape@currcls\endcsname}%
658     \fi%
659   \else\def\parsemodule@do@next{\relax\parsemodule@last@char}\fi%
660 \fi%
661 \fi%

```

```

662 \parsemodule@do@next%
663 }

```

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.

```

664 \expandafter\expandafter\expandafter\def%
665 \expandafter\expandafter\expandafter\parsemodule@converttoproperbraces%
666 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
667 \reset@parsemodule@catcodes%
668 \parsemodule@do@next@ii{#1}%
669 }

```

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.

```

670 \expandafter\expandafter\expandafter\def%
671 \expandafter\expandafter\expandafter\parsemodule@escapechar@checkbeginenv%
672 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
673 \expandafter\ifx\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed%
674 \reset@parsemodule@catcodes%
675 \def\parsemodule@do@next{\begin{#1}}%
676 \else%
677 \def\parsemodule@do@next{#1}%
678 \fi%
679 \parsemodule@do@next%
680 }
681 \expandafter\expandafter\expandafter\def%
682 \expandafter\expandafter\expandafter\parsemodule@escapechar@checkendenv%
683 \expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
684 \expandafter\ifx\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed%
685 \reset@parsemodule@catcodes%
686 \def\parsemodule@do@next{\end{#1}}%
687 \else%
688 \def\parsemodule@do@next{#1}%
689 \fi%
690 \parsemodule@do@next%
691 }

```

`\@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.

```

692 \newrobustcmd\@requiremodules[1]{%
693 \if@tempswa\requiremodules{#1}\fi%
694 }%

```

`\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.

```

695 \newrobustcmd\requiremodules[1]{%
696   \mod@showfalse%
697   \edef\mod@path{#1}%
698   \edef\mod@path{\expandafter\detokenize\expandafter{\mod@path}}%
699   \requiremodules@smsmode{#1}%
700 }%
```

`\requiremodules@smsmode` this reads `TEX` modules by setting the category codes for `sms` mode, `\inputting` the required file and wrapping it in a `\vbox` that gets stored away and ignored, in order to not produce any output. It also sets `\hbadness`, `\hfuzz` and friends to values that suppress overfull and underfull hbox messages.

```

701 \newbox\modules@import@tempbox
702 \def\requiremodules@smsmode#1{%
703   \setbox\modules@import@tempbox\vbox{%
704     \@smsmodetrue%
705     \set@parsemodule@catcodes%
706     \hbadness=100000\relax%
707     \hfuzz=10000pt\relax%
708     \vbadness=100000\relax%
709     \vfuzz=10000pt\relax%
710     \stexinput{#1.tex}%
711     \reset@parsemodule@catcodes%
712   }%
713   \parsemodule@maybesetcodes%
714 }
```

#### Test:

parsing `FOO/testmodule.tex`

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

### 3.4.2 importmodule

`\importmodule@bookkeeping`

```

715 \newif\if@importmodule@switchrepos\@importmodule@switchreposfalse
716 \def\importmodule@bookkeeping#1#2#3{%
717   \@importmodule@switchreposfalse%
718   \metasetkeys{importmodule}{#1}%
719   \ifcsvoid{importmodule@mhrepos}{%
720     \ifcsvoid{currentrepos@dir}{%
721       \let\importmodule@dir\stex@maindir%
722     }{%
723       \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
724     }%
725   }{%
726     \@importmodule@switchrepostrue%
```



```

727 \expandafter\let\csname importmodule@oldrepos@#2\endcsname\mh@currentrepos%
728 \setcurrentreposinfo\importmodule@mhrepos%
729 \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
730 }%
731 \StrCut{#2}\@QuestionMark\importmodule@subdir\importmodule@modulename%
732 \ifx\importmodule@modulename\@empty%
733 \let\importmodule@modulename\importmodule@subdir%
734 \let\importmodule@subdir\@empty%
735 \else%
736 \ifx\importmodule@subdir\@empty\else%
737 \edef\importmodule@dir{\importmodule@dir\@Slash\importmodule@subdir}%
738 \fi%
739 \fi%
740 \begingroup#3\endgroup%
741 \if@importmodule@switchrepos%
742 \expandafter\setcurrentreposinfo\csname importmodule@oldrepos@#2\endcsname%
743 \fi%
744 \ignorespacesandpars%
745 }

```

\importmodule

```

746 %\srefaddidkey{importmodule}
747 \addmetakey{importmodule}{mhrepos}
748 \newcommand\importmodule[2] [] {\@importmodule[#1]{#2}{export}}
749 \newcommand\@importmodule[3] [] {%
750 \importmodule@bookkeeping{#1}{#2}{%
751 \@importmodule[\importmodule@dir]\importmodule@modulename{#3}%
752 }%
753 }

```

\@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.

```

754 \newcommand\@importmodule[3] [] {%
755 {%
756 \edef\@load{#1}%
757 \edef\@importmodule@name{#2}
758 \if@smsmode\else\ifcsvoid{Module\@importmodule@name}{%
759 \stexiffileexists\@load{\requiremodules\@load}{%
760 \requiremodules{\@load\@Slash\@importmodule@name}%
761 }%
762 }{\fi%
763 \ifx\@load\@empty\else%
764 {% TODO

```

```

765 %      \edef\@path{\csname module@#2@path\endcsname}%
766 %      \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do nothing
767 %      {\PackageError{stex}% else signal an error
768 %      {Module Name Clash\MessageBreak%
769 %      A module with name #2 was already loaded under the path "\@path"\MessageBreak%
770 %      The imported path "\@load" is probably a different module with the\MessageBreak%
771 %      same name; this is dangerous -- not importing}%
772 %      {Check whether the Module name is correct}%
773 %      }%
774 %  }%
775 %\fi%
776 %\global\let\@importmodule\@load\@load%
777 %}
778 %\edef\@export{#3}\def\@export{export}%prepare comparison
779 %\ifx\@export\@export\export@defs{#2}\fi% export the module
780 %\ifx\@export\@export\addto@thismodulex{%
781 %\noexpand\@importmodule[\@importmodule@load]{#2}{noexport}%
782 %}
783 %\if@smsmode\else
784 %\ifcsvoid{this@module}{}%
785 %\ifcsvoid{module@imports@\module@uri}{%
786 %\csxdef{module@imports@\module@uri}{%
787 %\csname Module#2\endcsname\@URI%
788 %}%
789 %}{%
790 %\csxdef{module@imports@\module@uri}{%
791 %\csname Module#2\endcsname\@URI,%
792 %\csname module@imports@\module@uri\endcsname%
793 %}%
794 %}%
795 %}
796 %\fi\fi%
797 %\if@smsmode\else\activate@defs{#2}\fi% activate the module
798 %}

```

### Test:

`\importmodule {testmoduleimporta}:`

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

undefined

### Test:

`\importmodule {testmoduleimportb?importb}:`

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

macro:->\protect \bar

### Test:

macro:->\@invoke@module {http://mathhub.info/smg lom/algebra?band}

macro:->\@invoke@module {http://mathhub.info/smg lom/algebra?idempotent}

undefined

macro:->\@ifstar \@gimport@star \@gimport@nostar

`\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.

```
799 \def\activate@defs#1{%
800   \ifcsundef{Module#1}{
801     \PackageError{stex}{No module with name #1 loaded}{Probably missing an
802       \detokenize{\importmodule} (or variant) somewhere?
803   }
804 }{%
805   \ifcsundef{module@csname Module#1\endcsname\@URI @activated}%
806     {\csname module@defs@csname Module#1\endcsname\@URI\endcsname}{}%
807   \namedef{module@csname Module#1\endcsname\@URI @activated}{true}%
808 }%
809 }%
```

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

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

**Test:**

**Module 3.26**[Foo]:

**Module 3.27**[Bar]: undefined

**Module 3.28**[Baz]: undefined

macro:->\protect \bar

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

```
811 \def\inputref@preskip{}
812 \def\inputref@postskip{}
```

`\inputref` `\inputref{ $\langle path \text{ to the current file without extension} \rangle$ }` supports both absolute path and relative path, meanwhile, records the path and the extension (not for relative path).

```
813 \newrobustcmd\inputref[2] [] {%
814   \importmodule@bookkeeping{#1}{#2}{%
815     %\inputreftrue
816     \inputref@preskip%
817     \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
818     \inputref@postskip%
819   }%
820 }%
```

### 3.5 Symbols and Notations

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

```
821 \newif\if@symdeflocal\@symdeflocalfalse
```

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

```

822 \def\define@in@module#1#2{
823   \expandafter\edef\csname #1\endcsname{#2}%
824   \edef\define@in@module@temp{%
825     \def\expandafter\noexpand\csname#1\endcsname%
826       {#2}%
827   }%
828   \if@symdeflocal\else%
829     \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
830     \expandafter\endcsname\expandafter{\define@in@module@temp}%
831   \fi%
832 }
```

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

```

833 \addmetakey{symdecl}{name}%
834 \addmetakey{symdecl}{verbalization}%
835
836 % constructs a symbol name and a verbalization by splitting at exclamation
837 % points - e.g. \symdecl{symmetric!group} leads to name=symmetric-group
838 % and verbalization "symmetric group".
839 \def\symdecl@constructname#1{%
840   \def\symdecl@name{}%
841   \def\symdecl@verb{}%
842   \edef\symdecl@tempname{#1}%
843   \symdecl@constructname@loop%
844 }
845
846 \def\symdecl@constructname@loop{%
847   \ifx\symdecl@tempname\@empty\else%
848     \StrCut\symdecl@tempname!\symdecl@tempfirst\symdecl@tempname%
849     \ifx\symdecl@name\@empty%
850       \let\symdecl@name\symdecl@tempfirst%
851       \let\symdecl@verbalization\symdecl@tempfirst%
852       \symdecl@constructname@loop%
853     \else%
854       \edef\symdecl@name{\symdecl@name-\symdecl@tempfirst}%
855       \edef\symdecl@verbalization{\symdecl@verbalization\@Space\symdecl@tempfirst}%
856       \symdecl@constructname@loop%
857     \fi%
858   \fi%
859 }
860
861 \newcommand\symdecl[2][]{%
862   \ifcsdef{this@module}{%
863     \metasetkeys{symdecl}{#1}%
864     \ifcvoid{symdecl@name}{%
865       \ifcvoid{symdecl@verbalization}{%

```

```

866     \symdecl@constructname{#2}%
867   }{%
868     \edef\symdecl@name{#2}%
869   }%
870 }{%
871   \ifcsvoid\symdecl@verbalization{\edef\symdecl@verbalization{#2}}{}%
872 }%
873 \edef\symdef@uri{\module@uri\@QuestionMark\symdecl@name}%
874 \ifcsvoid\symdef@uri{%
875   \ifcsvoid\module@names@\module@uri{%
876     \csxdef\module@names@\module@uri{\symdecl@name}%
877   }{%
878     \csxdef\module@names@\module@uri{\symdecl@name,%
879       \csname module@names@\module@uri\endcsname}%
880   }%
881 }{%
882 % not compatible with circular dependencies, e.g. test/omdoc/07-modules/sms testa.tex
883 \PackageWarning{stex}{symbol already defined: \symdef@uri}{%
884   You need to pick a fresh name for your symbol%
885 }%
886 }%
887 \define@in@module\symdef@uri{\noexpand\@invoke@symbol{\symdef@uri}}%
888 \define@in@module{#2}{\noexpand\@invoke@symbol{\symdef@uri}}%
889 \global\expandafter\let\csname\symdef@uri\@Fragment verb\endcsname\symdecl@verbalization%
890 }{%
891 \PackageError{stex}{\detokenize{\symdecl} not in a module}{You need to be in a module%
892   in order to declare a new symbol}
893 }%
894 \ifinsymdef@\else\parsemodule@maybesetcodes\fi%
895 }

```

### Test:

**Module 3.29**[foo]: \symdecl {bar}

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

### 3.5.1 Notations

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

```

896 \def\modules@getURIfromName#1{%
897   \def\notation@uri{}%
898   \edef\modules@getURI@name{#1}%
899   \if@isuri\modules@getURI@name{%
900     \let\notation@uri\isuri@uri%
901   }{%
902     \ifcsvoid{this@module}{}{%
903       \expandafter\modules@getURIfromModule\expandafter{\module@uri}%
904       \ifx\notation@uri\empty%

```

```

905         \edef\modules@getURI@modules{\csname module@imports@\module@uri\endcsname}%
906         \expandafter\@for\expandafter\@I\expandafter:\expandafter=\modules@getURI@modules\do{%
907             \ifx\notation@uri\@empty%
908                 \expandafter\modules@getURIfromModule\expandafter{\@I}%
909             \fi%
910         }%
911     \fi%
912     \ifx\notation@uri\@empty%
913         \def\notation@extract@uri@curr{cs}%
914         \notation@extracturifrommacro{#1}%
915     \fi%
916     \ifx\notation@uri\@empty%
917         \PackageError{stex}{No symbol with name, URI or macroname \detokenize{#1} found!}{}%
918     \fi%
919 }%
920 }%
921 }
922
923 \def\if@isuri#1#2#3{%
924     \StrCount{#1}\@QuestionMark[\isuri@number]%
925     \ifnum\isuri@number=1 %
926         \StrCut{#1}\@QuestionMark\@isuri@mod\@isuri@name%
927         \ifcvoid{Module\@isuri@mod}{#3}{%
928             \edef\isuri@uri{\csname Module\@isuri@mod\endcsname\@URI\@QuestionMark\@isuri@name}%
929             #2%
930         }%
931     \else%
932         \ifnum\isuri@number=2 %
933             \edef\isuri@uri{#1}#2\else#3%
934         \fi%
935     \fi%
936 }
937
938 \def\modules@getURIfromModule#1{%
939     \edef\modules@getURI@names{\csname module@names@#1\endcsname}%
940     \expandafter\@for\expandafter\@I\expandafter:\expandafter=%
941     \modules@getURI@names\do{%
942         \ifx\notation@uri\@empty%
943             \ifx\@I\modules@getURI@name%
944                 \edef\notation@uri{#1\@QuestionMark\@I}%
945             \fi%
946         \fi%
947     }%
948 }
949
950 % extracts the full URI from \foo or anything being \ifx-equal to \foo,
951 % by expanding until we reach \@invoke@symbol{<uri>}
952 \def\notation@extracturifrommacro#1{%
953     \ifcvoid{#1}{}%
954     \expandafter\let\expandafter\notation@extract@uri@nextcs\csname#1\endcsname%

```

```

955 \ifx\notation@extract@uri@nextcs\notation@extract@uri@currccs\else%
956 \let\notation@extract@uri@currccs\notation@extract@uri@nextcs%
957 \expandafter\notation@extract@uriII\notation@extract@uri@nextcs\notation@end%
958 \fi%
959 }%
960 }
961 \long\def\notation@extract@uriII#1#2\notation@end{%
962 \def\notation@extract@check@temp{#2}
963 \ifx\@invoke@symbol#1%
964 \edef\notation@uri{#2}%
965 \else%
966 \ifx\notation@extract@check@temp\@empty\else%
967 \expandafter\def\expandafter\notation@extract@uri@nextcs\expandafter{#1{#2}}%
968 \notation@extract@uri\notation@extract@uri@nextcs}%
969 \fi%
970 \fi%
971 }

```

`\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`  
 TODO with brackets, e.g. `\notation[withbrackets={\langle,\rangle}]{foo}{...}`

```

972 % parses the first two arguments:
973 \providerobustcmd\notation[2] []{%
974 \edef\notation@first{#1}%
975 \edef\notation@second{#2}%
976 \notation@%
977 }
978
979 % parses the last two arguments
980 \newcommand\notation@[2] [0]{%
981 \edef\notation@donext{\noexpand\notation@@[\notation@first]%
982 { \notation@second}[#1]}%
983 \notation@donext{#2}%
984 }
985
986 % parses the notation arguments and wraps them in
987 % \notation@assoc and \notation@argprec for flexary arguments and precedences
988 \def\notation@@[#1]#2[#3]#4{%
989 \modules@getURIfromName{#2}%
990 \notation@parse@params{#1}{#3}
991 \let\notation@curr@todo@args\notation@curr@args%
992 \def\notation@temp\notation{%
993 \StrLen\notation@curr@args[\notation@temp@arity]%
994 \expandafter\renewcommand\expandafter\notation@temp\notation%
995 \expandafter[\notation@temp@arity]{#4}%
996 % precedence
997 \IfSubStr\notation@curr@args;{%
998 \StrCut\notation@curr@args;\notation@curr@prec\notation@curr@args%
999 \ifx\notation@curr@prec\@empty\def\notation@curr@prec{0}\fi%

```

```

1000 }{%
1001   \ifx\notation@curr@prec\@empty%
1002     \ifnum\notation@temp@arity=0\relax%
1003       \edef\notation@curr@prec{\infprec}%
1004     \else%
1005       \def\notation@curr@prec{0}%
1006     \fi%
1007   \else%
1008     \edef\notation@curr@prec{\notation@curr@prec}%
1009     \def\notation@curr@prec{}%
1010   \fi%
1011 }%
1012 % arguments
1013 \def\notation@curr@extargs{}
1014 \def\notation@nextarg@index{1}%
1015 \notation@do@args%
1016 }
1017
1018 % parses additional notation components for (associative) arguments
1019 \def\notation@do@args{%
1020   \def\notation@nextarg@temp{}%
1021   \ifx\notation@curr@todo@args\@empty%
1022     \notation@after%
1023   \else%
1024     % argument precedence
1025     \IfSubStr\notation@curr@prec{x}{%
1026       \StrCut\notation@curr@prec{x}\notation@curr@argprec\notation@curr@prec%
1027     }{%
1028       \edef\notation@curr@argprec{\notation@curr@prec}%
1029       \def\notation@curr@prec{}%
1030     }%
1031     \ifx\notation@curr@argprec\@empty%
1032       \let\notation@curr@argprec\notation@curr@prec%
1033     \fi%
1034     \StrChar\notation@curr@todo@args1[\notation@argchar]%
1035     \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
1036     \expandafter\ifx\notation@argchar i%
1037       % normal argument
1038       \edef\notation@nextarg@temp{\noexpand\notation@argprec{\notation@curr@argprec}{#####\
1039       \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }
1040       \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
1041         \expandafter{\notation@nextarg@temp}%
1042       \expandafter\expandafter\expandafter\notation@do@args%
1043     \else%
1044       % associative argument
1045       \expandafter\expandafter\expandafter\notation@parse@assocarg%
1046     \fi%
1047   \fi%
1048 }
1049

```



```

1050 \def\notation@parse@assocarg#1{%
1051   \edef\notation@nextarg@temp{{\noexpand\notation@argprec{\notation@curr@argprec}}{\noexpand\notation@nextarg@temp}}
1052   \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }%
1053   \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
1054   \expandafter{\notation@nextarg@temp}%
1055   \notation@do@args%
1056 }
1057
1058 \protected\def\safe@newcommand#1{%
1059   \ifdefined#1\expandafter\renewcommand\else\expandafter\newcommand\fi#1%
1060 }
1061
1062 % finally creates the actual macros
1063 \def\notation@after{
1064   \let\ex\expandafter%
1065   \ex\ex\ex\def\ex\ex\ex\notation@temp\notation\ex\ex\ex%
1066     {\ex\notation@temp\notation\notation@curr@extargs}%
1067   \edef\notation@temp\notation{\noexpand\notation@symprec{\notation@curr@prec}}{\ex\unexpanded\ex}%
1068   \def\notation@temp@fragment{}%
1069   \ifx\notation@curr@arity\@empty\else%
1070     \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1071   \fi%
1072   \ifx\notation@curr@lang\@empty\else%
1073     \ifx\notation@temp@fragment\@empty%
1074       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1075     \else%
1076       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1077     \fi%
1078   \fi%
1079   \ifx\notation@curr@variant\@empty\else%
1080     \ifx\notation@temp@fragment\@empty%
1081       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1082     \else%
1083       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@variant}%
1084     \fi%
1085   \fi%
1086   \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
1087   \ifcsvoid{\notation@csname}{%
1088     \ex\ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
1089       \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@temp@arity\ex]%
1090     \ex{\notation@temp\notation}%
1091     \edef\symdecl@temps{%
1092       \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@temp@arity%
1093     }%
1094     \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
1095     \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp\notation}%
1096   }{%
1097     \PackageWarning{stex}{notation already defined: \notation@csname}{%
1098       Choose a different set of notation options (variant,lang,arity)%
1099   }%

```

```

1100 }%
1101 \parsemodule@maybesetcodes%
1102 }
1103
1104 % parses optional parameters
1105 \def\notation@parse@params#1#2{%
1106   \def\notation@curr@prec{%
1107     \def\notation@curr@args{%
1108       \def\notation@curr@variant{%
1109         \def\notation@curr@arity{%
1110           \def\notation@curr@provided@arity{#2}
1111           \def\notation@curr@lang{%
1112             \def\notation@options@temp{#1}
1113             \notation@parse@params@%
1114             \ifx\notation@curr@args\@empty%
1115               \ifx\notation@curr@provided@arity\@empty%
1116                 \notation@num@to@ia\notation@curr@arity%
1117               \else%
1118                 \notation@num@to@ia\notation@curr@provided@arity%
1119               \fi%
1120             \fi%
1121           }
1122           \def\notation@parse@params@{%
1123             \IfSubStr\notation@options@temp,{%
1124               \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
1125               \notation@parse@param%
1126               \notation@parse@params@%
1127             }{\ifx\notation@options@temp\@empty\else%
1128               \let\notation@option@temp\notation@options@temp%
1129               \notation@parse@param%
1130             \fi}%
1131           }
1132
1133 %parses an individual optional argument/key-value-pair
1134 \def\notation@parse@param{%
1135   \trimstring\notation@option@temp%
1136   \ifx\notation@option@temp\@empty\else%
1137     \IfSubStr\notation@option@temp={%
1138       \StrCut\notation@option@temp=\notation@key\notation@value%
1139       \trimstring\notation@key%
1140       \trimstring\notation@value%
1141       \IfStrEq\notation@key{prec}{%
1142         \edef\notation@curr@prec{\notation@value}%
1143       }{%
1144         \IfStrEq\notation@key{args}{%
1145           \edef\notation@curr@args{\notation@value}%
1146         }{%
1147           \IfStrEq\notation@key{lang}{%
1148             \edef\notation@curr@lang{\notation@value}%
1149           }{%

```

```

1150     \IfStrEq\notation@key{variant}{%
1151         \edef\notation@curr@variant{\notation@value}%
1152     }{%
1153     \IfStrEq\notation@key{arity}{%
1154         \edef\notation@curr@arity{\notation@value}%
1155     }{%
1156     }}}}%
1157 }{%
1158     \edef\notation@curr@variant{\notation@option@temp}%
1159 }%
1160 \fi%
1161 }
1162
1163 % converts an integer to a string of 'i's, e.g. 3 => iii,
1164 % and stores the result in \notation@curr@args
1165 \def\notation@num@to@ia#1{%
1166     \IfInteger{#1}{
1167         \notation@num@to@ia@#1%
1168     }{%
1169         %
1170     }%
1171 }
1172 \def\notation@num@to@ia@#1{%
1173     \ifnum#1>0%
1174         \edef\notation@curr@args{\notation@curr@args i}%
1175         \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
1176     \fi%
1177 }

```

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

```

1178 \def\notation@assoc#1#2{% function, argv
1179     \let\@tmpop=\relax% do not print the function the first time round
1180     \@for\@I:=#2\do{\@tmpop% print the function
1181         % write the i-th argument with locally updated precedence
1182         \@I%
1183     \def\@tmpop{#1}%
1184     }%
1185 }%
1186
1187 \def\notation@lparen{()
1188 \def\notation@rparen{)}
1189 \def\infpref{1000000}
1190 \def\neginfpref{-\infpref}
1191
1192 \newcount\notation@downprec
1193 \notation@downprec=\neginfpref
1194
1195 % patching displaymode
1196 \newif\if@displaymode\@displaymodefalse

```

```

1197 \expandafter\everydisplay\expandafter{\the\everydisplay\@displaymodetrue}
1198 \let\old@displaystyle\displaystyle
1199 \def\displaystyle{\old@displaystyle\@displaymodetrue}
1200
1201 \def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
1202   \def\notation@innertmp{#1}%
1203   \let\ex\expandafter%
1204   \if@displaymode%
1205     \ex\ex\ex\left\ex\ex\ex\notation@lparen%
1206     \ex\notation@resetbrackets\ex\notation@innertmp%
1207     \ex\right\notation@rparen%
1208   \else%
1209     \ex\ex\ex\notation@lparen%
1210     \ex\notation@resetbrackets\ex\notation@innertmp%
1211     \notation@rparen%
1212   \fi%
1213 }
1214
1215 \def\withbrackets#1#2#3{%
1216   \edef\notation@lparen{#1}%
1217   \edef\notation@rparen{#2}%
1218   #3%
1219   \notation@resetbrackets%
1220 }
1221
1222 \def\notation@resetbrackets{%
1223   \def\notation@lparen{()%
1224   \def\notation@rparen{)%}%
1225 }
1226
1227 \def\notation@symprec#1#2{%
1228   \ifnum#1>\notation@downprec\relax%
1229     \notation@resetbrackets#2%
1230   \else%
1231     \ifnum\notation@downprec=\infprec\relax%
1232       \notation@resetbrackets#2%
1233     \else
1234       \if@inarray@
1235         \notation@resetbrackets#2
1236       \else\dobrackets{#2}\fi%
1237   \fi\fi%
1238 }
1239
1240 \newif\if@inarray@\@inarray@false
1241
1242 \def\notation@argprec#1#2{%
1243   \def\notation@innertmp{#2}
1244   \edef\notation@downprec@temp{\number#1}%
1245   \notation@downprec=\expandafter\notation@downprec@temp%
1246   \expandafter\relax\expandafter\notation@innertmp%

```

```

1247 \expandafter\notation@downprec\expandafter=\number\notation@downprec\relax%
1248 }

```

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

```

1249 \protected\def\@invoke@symbol#1{%
1250   \def\@invoke@symbol@first{#1}%
1251   \symbol@args%
1252 }

```

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

```

1253 \newcommand\symbol@args[1][]{%
1254   \notation@parse@params{#1}{}%
1255   \def\notation@temp@fragment{}%
1256   \ifx\notation@curr@arity\@empty\else%
1257     \edef\notation@temp@fragment{arity=\notation@curr@arity}%
1258   \fi%
1259   \ifx\notation@curr@lang\@empty\else%
1260     \ifx\notation@temp@fragment\@empty%
1261       \edef\notation@temp@fragment{lang=\notation@curr@lang}%
1262     \else%
1263       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
1264     \fi%
1265   \fi%
1266   \ifx\notation@curr@variant\@empty\else%
1267     \ifx\notation@temp@fragment\@empty%
1268       \edef\notation@temp@fragment{variant=\notation@curr@variant}%
1269     \else%
1270       \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@variant}%
1271     \fi%
1272   \fi%
1273   %
1274   \ifmmode\def\invoke@symbol@next{\@invoke@symbol@math\@invoke@symbol@first\notation@temp@fragment}%
1275   \else\def\invoke@symbol@next{\@invoke@symbol@text\@invoke@symbol@first\notation@temp@fragment}%
1276   \invoke@symbol@next%
1277 }

```

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

```

1278 \def\@invoke@symbol@math#1#2{%
1279   \csname #1\@Fragment#2\endcsname%
1280 }

```

TODO:

```

1281 \def\@invoke@symbol@text#1#2{%
1282   \csname #1\@Fragment verb\ifx#2\@empty\else#2\fi\endcsname%
1283 }

```

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

```

1284 \def\setstexlang#1{%
1285   \def\stex@lang{#1}%
1286 }%
1287 \setstexlang{en}
1288 \def\setstexvariant#1#2{%
1289   % TODO
1290 }
1291 \def\setstexvariants#1{%
1292   \def\stex@variants{#1}%
1293 }

```

**Test:**

**Module** 3.30[FooBar]: \symdecl {barbar}  
\notation [arity=0]{barbar}{\psi }  
\notation [prec=50;\infprec ]{barbar}[1]{\barbar [arity=0]\dobrackets {##1}}  
  
\notation [arity=0,variant=cap]{barbar}{\Psi }  
\notation [variant=cap]{barbar}[1]{\barbar [arity=0,variant=cap]\dobrackets {##1}}

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

```

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

```

```

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

```

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

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

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

foo bar

### 3.6 Term References

`\ifhref`

```
1294 \newif\ifhref\hreffalse%
1295 \AtBeginDocument{%
1296   \@ifpackageloaded{hyperref}{%
1297     \hreftrue%
1298   }{%
1299     \hreffalse%
1300   }%
1301 }
```

`\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.

```
1302 \def\termref@maketarget#1#2{%
1303   % #1: symbol URI
1304   % #2: text
1305   \ifhref%
1306     \hypertarget{sref@#1@target}{#2}%
1307   \fi%
1308   \expandafter\edef\csname sref@#1\endcsname##1{%
1309     \ifhref\noexpand\hyperlink{sref@#1@target}{##1}\fi%
1310   }%
1311 }
```

`\@termref`

```
1312 \def\@termref#1#2{%
1313   % #1: symbol URI
1314   % #2: text
1315   \ifcvoid{#1}{%
1316     \StrCut{#1}\@QuestionMark\termref@mod\termref@name%
1317     \ifcvoid{\termref@mod}{%
1318       \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
1319     }{%
1320       \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
1321         contains no symbol with name \termref@name.%
1322       }{%
1323       }%
1324     }{%
1325       \ifcvoid{sref@#1}{%
1326         % TODO: No reference point exists!
1327       }{%
1328         \csname sref@#1\endcsname{#2}%
1329       }%
1330     }%
1331 }
```

```

1329   }%
1330 }%
1331 }

```

### 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
1332 \newif\ifhhref\hreffalse%
1333 \AtBeginDocument{%
1334   \ifpackageloaded{hyperref}{%
1335     \hreftrue%
1336   }{%
1337     \hreffalse%
1338   }%
1339 }%
1340 \newcommand\sref@href@ifh[2]{%
1341   \ifhhref%
1342     \href{#1}{#2}%
1343   \else%
1344     #2%
1345   \fi%
1346 }%
1347 \newcommand\sref@hlink@ifh[2]{%
1348   \ifhhref%
1349     \hyperlink{#1}{#2}%
1350   \else%
1351     #2%
1352   \fi%
1353 }%
1354 \newcommand\sref@target@ifh[2]{%
1355   \ifhhref%
1356     \hypertarget{#1}{#2}%
1357   \else%
1358     #2%
1359   \fi%
1360 }%

```

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.

```

1361 \def\sref@target{%
1362   \ifx\sref@id\@empty%
1363     \relax%
1364   \else%

```



```

1365 \edef\@target{sref@\ifcsundef{sref@part}{}\sref@part @}\sref@id @target}%
1366 \sref@target@ifh\@target{}%
1367 \fi%
1368 }%

\srefaddidkey \srefaddidkey[\langle keyval \rangle]{\langle group \rangle} extends the metadata keys of the group
\langle group \rangle with an id key. In the optional key/value pairs in \langle keyval \rangle the
prefix key can be used to specify a prefix. Note that the id key defined by
\srefaddidkey[\langle keyval \rangle]{\langle group \rangle} not only defines \sref@id, which is used for
referencing by the sref package, but also \langle group \rangle@id, which is used for showing
metadata via the showmeta option of the metakeys package.
1369 \addmetakey{srefaddidkey}{prefix}
1370 \newcommand\srefaddidkey[2][]{%
1371 \metasetkeys{srefaddidkey}{#1}%
1372 \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
1373 \metakeys@ext@clear@keys{#2}{id}{}%
1374 \metakeys@ext@showkeys{#2}{id}%
1375 \define@key{#2}{id}{%
1376 \edef\sref@id{\srefaddidkey@prefix ##1}%
1377 %\expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
1378 \csedef{#2@id}{\srefaddidkey@prefix ##1}%
1379 }%
1380 }%

\@sref@def This macro stores the value of its last argument in a custom macro for reference.
1381 \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.
1382 \ifextrefs%
1383 \newwrite\refs@file%
1384 \else%
1385 \def\refs@file{\@auxout}%
1386 \fi%

\sref@def This macro writes an \@sref@def command to the current aux file and also exe-
cutes it.
1387 \newcommand\sref@def[3]{%
1388 \protected@write\refs@file{}\string\@sref@def{#1}{#2}{#3}}%
1389 }%

\sref@label The \sref@label macro writes a label definition to the auxfile.
1390 \newcommand\sref@label[2]{%
1391 \sref@def{\ifcsundef{sref@part}{}\sref@part @}{#2}{page}{\thepage}%
1392 \sref@def{\ifcsundef{sref@part}{}\sref@part @}{#2}{label}{#1}%
1393 }%

\sreflabel The \sreflabel macro is a semantic version of \label, it combines the catego-
rization given in the first argument with LATEX's \@currentlabel.
1394 \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.

```

1395 \def\sref@id{} % make sure that defined
1396 \newcommand\sref@label@id[1]{%
1397   \ifx\sref@id\@empty%
1398     \relax%
1399   \else%
1400     \sref@label{#1}{\sref@id}%
1401   \fi%
1402 }%
```

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

```

1403 \newcommand\sref@label@id@arg[2]{%
1404   \def\@id{#2}
1405   \ifx\@id\@empty%
1406     \relax%
1407   \else%
1408     \sref@label{#1}{\@id}%
1409   \fi%
1410 }%
```

### 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`.

```

1411 \newenvironment{modsig}[2] []{\def\@test{#1}%
1412 \ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
1413 \expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
1414 \ignorespacesandpars}
1415 {\end{module}\ignorespacesandparsafterend}
```

### 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 `\mhcurrentrepos`.

```

1416 \def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
1417 \newrobustcmd\@gimport@star[2][\def\@test{#1}%
1418 \edef\mh@@repos{\mh@currentrepos}%
1419 \ifx\@test\@empty%
1420 \importmhmodule[conservative,mhrepos=\mh@@repos,path=#2]{#2}%
1421 \else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
1422 \setcurrentreposinfo{\mh@@repos}%
1423 \ignorespacesandpars\parsemodule@maybesetcodes}
1424 \newrobustcmd\@gimport@nostar[2][\def\@test{#1}%
1425 \edef\mh@@repos{\mh@currentrepos}%
1426 \ifx\@test\@empty%
1427 \importmhmodule[mhrepos=\mh@@repos,path=#2]{#2}%
1428 \else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
1429 \setcurrentreposinfo{\mh@@repos}%
1430 \ignorespacesandpars\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.

```

1431 \def\modules@@first#1/#2;{#1}
1432 \newcommand\libinput[1]{%
1433 \ifcsvoid{mh@currentrepos}{%
1434   \PackageError{mathhub}{current MathHub repository not found}{}%
1435   {}
1436 \edef\mh@group{\expandafter\modules@@first\mh@currentrepos;}
1437 \let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
1438 \def\mh@inffile{\MathHub{\mh@group/meta-inf/lib/#1}}
1439 \def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
1440 \IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
1441 \IfFileExists\mh@inffile{\IfFileExists\mh@libfile}{%
1442   {\PackageError{mathhub}
1443     {Library file missing; cannot input #1.tex\MessageBreak%
1444       Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
1445       do not exist}%
1446     {Check whether the file name is correct}}}%
1447 \IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
1448 \let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}

```

### 3.11 omdoc/omgroup

```

1449 \newcount\section@level
1450
1451 \section@level=2
1452 \ifdefstring{omdoc@sty@class}{book}{\section@level=0}{

```

```

1453 \ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{}
1454 \ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{}
1455 \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>.

1456 \newcommand\omgroup@nonum[2]{%
1457 \ifx\hyper@anchor\@undefined\else\phantomsection\fi%
1458 \addcontentsline{toc}{#1}{#2}\@nameuse{#1}*{#2}}

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

1459 \newcommand\omgroup@num[2]{%
1460 \edef\@ID{\sref@id}
1461 \ifx\omgroup@short\@empty% no short title
1462 \@nameuse{#1}{#2}%
1463 \else% we have a short title
1464 \@ifundefined{rdfmeta@sectioning}%
1465   {\@nameuse{#1}[\omgroup@short]{#2}}%
1466   {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
1467 \fi%
1468 \sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@ID}

omgroup
1469 \def\@true{true}
1470 \def\@false{false}
1471 \srefaddidkey{omgroup}
1472 \addmetakey{omgroup}{date}
1473 \addmetakey{omgroup}{creators}
1474 \addmetakey{omgroup}{contributors}
1475 \addmetakey{omgroup}{srccite}
1476 \addmetakey{omgroup}{type}
1477 \addmetakey*{omgroup}{short}
1478 \addmetakey*{omgroup}{display}
1479 \addmetakey[false]{omgroup}{loadmodules}[true]

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

1480 \newif\if@mainmatter\@mainmattertrue
1481 \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.

1482 \addmetakey{\omdoc@sect}{name}
1483 \addmetakey[false]{\omdoc@sect}{clear}[true]
1484 \addmetakey{\omdoc@sect}{ref}

```

```

1485 \addmetakey[false]{omdoc@sect}{num}[true]
1486 \newcommand\omdoc@sectioning[3][\metasetkeys{omdoc@sect}{#1}%
1487 \ifx\omdoc@sect@clear@true\cleardoublepage\fi%
1488 \if@mainmatter% numbering not overridden by frontmatter, etc.
1489 \ifx\omdoc@sect@num@true\omgroup@num{#2}{#3}\else\omgroup@nonum{#2}{#3}\fi%
1490 \def\current@section@level{\omdoc@sect@name}%
1491 \else\omgroup@nonum{#2}{#3}%
1492 \fi}% if@mainmatter

```

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

```

1493 \newcommand\omgroup@redefine@addtocontents[1]{%
1494 %\edef\@import{#1}%
1495 %\@for\@I:=\@import\do{%
1496 %\edef\@path{\csname module@\@I @path\endcsname}%
1497 %\@ifundefined{tf@toc}\relax%
1498 % {\protected@write\tf@toc}{\string\@requiremodules{\@path}}}%
1499 %\ifx\hyper@anchor\undefined% hyperref.sty loaded?
1500 %\def\addcontentsline##1##2##3{%
1501 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}%
1502 %\else% hyperref.sty not loaded
1503 %\def\addcontentsline##1##2##3{%
1504 %\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{##1}{##3}}{\thepage}}{\c
1505 %\fi
1506 }% hyperref.sty loaded?

```

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

```

1507 \newcount\omgroup@level
1508 \newenvironment{omgroup}[2][\keys, title
1509 {\metasetkeys{omgroup}{#1}\sref@target%
1510 \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.

```

1511 \ifx\omgroup@loadmodules@true%
1512 \omgroup@redefine@addtocontents{\@ifundefined{module@id}\used@modules%
1513 {\@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`.

```

1514 \advance\section@level by 1\relax%
1515 \ifcase\section@level%
1516 \or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
1517 \or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
1518 \or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
1519 \or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
1520 \or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%

```

```

1521 \or\omdoc@sectioning[name=\omdoc@paragraph@kw,ref=this \omdoc@paragraph@kw]{paragraph}{#2}%
1522 \or\omdoc@sectioning[name=\omdoc@subparagraph@kw,ref=this \omdoc@subparagraph@kw]{paragraph}{#2}%
1523 \fi% \ifcase
1524 \at@begin@omgroup[#1]\section@level{#2}}}% for customization
1525 {\advance\section@level by -1\advance\omgroup@level by -1}

and finally, we localize the sections
1526 \newcommand\omdoc@part@kw{Part}
1527 \newcommand\omdoc@chapter@kw{Chapter}
1528 \newcommand\omdoc@section@kw{Section}
1529 \newcommand\omdoc@subsection@kw{Subsection}
1530 \newcommand\omdoc@subsubsection@kw{Subsubsection}
1531 \newcommand\omdoc@paragraph@kw{paragraph}
1532 \newcommand\omdoc@subparagraph@kw{subparagraph}

\setSGvar set a global variable
1533 \newcommand\setSGvar[1]{\@namedef{sTeX@Gvar@#1}}

\useSGvar use a global variable
1534 \newrobustcmd\useSGvar[1]{%
1535   \@ifundefined{sTeX@Gvar@#1}
1536   {\PackageError{omdoc}
1537     {The sTeX Global variable #1 is undefined}
1538     {set it with \protect\setSGvar}}
1539 \@nameuse{sTeX@Gvar@#1}}

blindomgroup
1540 \newcommand\at@begin@blindomgroup[1]{%
1541 \newenvironment{blindomgroup}
1542 {\advance\section@level by 1\at@begin@blindomgroup\setion@level}
1543 {\advance\section@level by -1}

```

### 3.12 omtext

## 4 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).

```

1544 \srefaddidkey{omtext}
1545 \addmetakey[] {omtext}{functions}
1546 \addmetakey* {omtext}{display}
1547 \addmetakey {omtext}{for}
1548 \addmetakey {omtext}{from}
1549 \addmetakey {omtext}{type}
1550 \addmetakey* {omtext}{title}
1551 \addmetakey* {omtext}{start}

```

```

1552 \addmetakey{omtext}{theory}
1553 \addmetakey{omtext}{continues}
1554 \addmetakey{omtext}{verbalizes}
1555 \addmetakey{omtext}{subject}

\st@flow We define this macro, so that we can test whether the display key has the value
flow
1556 \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.

1557 \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.

1558 \def\omtext@pre@skip{\smallskip}
1559 \def\omtext@post@skip{}
1560 \newenvironment{omtext}[1][\@in@omtexttrue%
1561 \bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
1562 \def\lec##1{\@lec{##1}}}%
1563 \omtext@pre@skip\par\noindent%
1564 \ifx\omtext@title@empty%
1565 \ifx\omtext@start@empty\else%
1566 \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
1567 \fi% end omtext@start empty
1568 \else\stDMemph{\omtext@title}:\enspace%
1569 \ifx\omtext@start@empty\else\omtext@start\enspace\fi%
1570 \fi% end omtext@title empty
1571 \ignorespacesandpars}
1572 {\egroup\omtext@post@skip\@in@omtextfalse\ignorespacesandpars}

```

## 5 Phrase-level Markup

```

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

1573 \srefaddidkey{phrase}
1574 \addmetakey{phrase}{style}
1575 \addmetakey{phrase}{class}
1576 \addmetakey{phrase}{index}
1577 \addmetakey{phrase}{verbalizes}
1578 \addmetakey{phrase}{type}
1579 \addmetakey{phrase}{only}
1580 \newcommand\phrase[2][\metasetkeys{phrase}{#1}%
1581 \ifx\phrase@only@empty\only<\phrase@only>{#2}\else #2\fi}

\coref*

1582 \providecommand\textsubscript[1]{\ensuremath{_{#1}}}

```

```

1583 \newcommand\corefs[2]{#1\textsubscript{#2}}
1584 \newcommand\coreft[2]{#1\textsuperscript{#2}}

\n*lex
1585 \newcommand\nlex[1]{\green{\sl{#1}}}
1586 \newcommand\nlcex[1]{*\green{\sl{#1}}}

sinlinequote
1587 \def\@sinlinequote#1{‘‘{\sl{#1}}}’’}
1588 \def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
1589 \newcommand\sinlinequote[2]{}
1590 {\def\@opt{#1}\ifx\@opt\@empty\@sinlinequote{#2}\else\@@sinlinequote\@opt{#2}\fi}

```

## 6 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.

```

1591 \newcommand\vdec[2][]{#2}
1592 \newcommand\vest[2][]{#2}
1593 \newcommand\vcond[2][]{#2}

EdN:1 \strucdec 1
1594 \newcommand\strucdec[2][]{#2}

EdN:2 \impdec 2
1595 \newcommand\impdec[2][]{#2}

```

## 7 Block-Level Markup

```

sblockquote
1596 \def\begin@sblockquote{\begin{quote}\sl}
1597 \def\end@sblockquote{\end{quote}}
1598 \def\begin@@sblockquote#1{\begin@sblockquote}
1599 \def\end@@sblockquote#1{\def\@lec##1{\textrm{##1}}\@lec{#1}\end@sblockquote}
1600 \newenvironment{sblockquote}[1]{}
1601 {\def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
1602 {\ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}

sboxquote
1603 \newenvironment{sboxquote}[1]{}
1604 {\def\@src{#1}\begin{mdframed}[leftmargin=.5cm,rightmargin=.5cm]}
1605 {\@lec{\textrm{\@src}}\end{mdframed}}

```

---

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

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



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.

```
1606 \providecommand{\@@lec}[1]{(#1)}
1607 \def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@@lec{#1}}
1608 \def\lec#1{\@lec{#1}\par}
```

## 8 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.

```
1609 \addmetakey{omdoc@index}{at}
1610 \addmetakey[false]{omdoc@index}{loadmodules}[true]
1611 \newcommand\omdoc@indexi[2][\ifindex%
1612 \metasetkeys{omdoc@index}{#1}%
1613 \@bsphack\begingroup\@sanitize%
1614 \protected@write\@indexfile{}\string\indexentry%
1615 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1616 \ifx\omdoc@index@loadmodules\@true%
1617 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
1618 \else #2\fi% loadmodules
1619 }\the page}}%
1620 \endgroup\@esphack\fi}%ifindex
1621 \newcommand\omdoc@indexii[3][\ifindex%
1622 \metasetkeys{omdoc@index}{#1}%
1623 \@bsphack\begingroup\@sanitize%
1624 \protected@write\@indexfile{}\string\indexentry%
1625 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1626 \ifx\omdoc@index@loadmodules\@true%
1627 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1628 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
1629 \else #2!#3\fi% loadmodules
1630 }\the page}}%
1631 \endgroup\@esphack\fi}%ifindex
1632 \newcommand\omdoc@indexiii[4][\ifindex%
1633 \metasetkeys{omdoc@index}{#1}%
1634 \@bsphack\begingroup\@sanitize%
1635 \protected@write\@indexfile{}\string\indexentry%
1636 {\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
1637 \ifx\omdoc@index@loadmodules\@true%
```

```

1638 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1639 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1640 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1641 \else #2!#3!#4\fi% loadmodules
1642 }\the page}}%
1643 \endgroup\@esphack\fi}%ifindex
1644 \newcommand\omdoc@indexiv[5] [] {\ifindex%
1645 \metasetkeys{omdoc@index}{#1}%
1646 \@bsphack\begin group\@sanitize%
1647 \protected@write\indexfile{}\string\indexentry%
1648 {\ifx\omdoc@index@at\empty\else\omdoc@index@at @\fi%
1649 \ifx\omdoc@index@loadmodules\@true%
1650 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
1651 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
1652 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
1653 \string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
1654 \else #2!#3!#4!#5\fi% loadmodules
1655 }\the page}}%
1656 \endgroup\@esphack\fi}%ifindex

```

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

**\\*indi\***

```

1657 \newcommand\aindi[3] [] {\{#2\}\omdoc@indexi[#1]{#3}}
1658 \newcommand\indi[2] [] {\{#2\}\omdoc@indexi[#1]{#2}}
1659 \newcommand\indis[2] [] {\{#2\}\omdoc@indexi[#1]{#2s}}
1660 \newcommand\Indi[2] [] {\{\captitalize{#2}\}\omdoc@indexi[#1]{#2}}
1661 \newcommand\Indis[2] [] {\{\capitalize{#2}\}\omdoc@indexi[#1]{#2s}}
1662
1663 \newcommand\@indii[3] [] {\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
1664 \newcommand\aindii[4] [] {\{#2\}\@indii[#1]{#3}{#4}}
1665 \newcommand\indii[3] [] {\{#2 #3\}\@indii[#1]{#2}{#3}}
1666 \newcommand\indiis[3] [] {\{#2 #3s\}\@indii[#1]{#2}{#3}}
1667 \newcommand\Indii[3] [] {\{\captitalize{#2 #3}\}\@indii[#1]{#2}{#3}}
1668 \newcommand\Indiis[3] [] {\{\capitalize{#2 #3}\}\@indii[#1]{#2}{#3}}
1669
1670 \newcommand\@indiii[4] [] {\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexii[#1]{#3}{#2 (#4)}}
1671 \newcommand\aindiii[5] [] {\{#2\}\@indiii[#1]{#3}{#4}{#5}}
1672 \newcommand\indiii[4] [] {\{#2 #3 #4\}\@indiii[#1]{#2}{#3}{#4}}
1673 \newcommand\indiis[4] [] {\{#2 #3 #4s\}\@indiii[#1]{#2}{#3}{#4}}
1674 \newcommand\Indiii[4] [] {\{\captitalize{#2 #3 #4}\}\@indiii[#1]{#2}{#3}{#4}}
1675 \newcommand\Indiis[4] [] {\{\capitalize{#2 #3 #4s}\}\@indiii[#1]{#2}{#3}{#4}}
1676
1677 \newcommand\@indiv[5] [] {\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
1678 \newcommand\aindiv[6] [] {\{#2\}\@indiv[#1]{#3}{#4}{#5}{#6}}
1679 \newcommand\indiv[5] [] {\{#2 #3 #4 #5\}\@indiv[#1]{#2}{#3}{#4}{#5}}
1680 \newcommand\indivs[5] [] {\{#2 #3 #4 #5s\}\@indiv[#1]{#2}{#3}{#4}{#5}}
1681 \newcommand\Indiv[5] [] {\{\captitalize{#2 #3 #4 #5s}\}\@indiv[#1]{#2}{#3}{#4}{#5}}
1682 \newcommand\Indivs[5] [] {\{\capitalize{#2 #3 #4 #5s}\}\@indiv[#1]{#2}{#3}{#4}{#5}}

```

## 9 Miscellaneous

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

```

1683 \newcommand\hateq{\ensuremath{\widehat{=}}\xspace}
1684 \newcommand\hatequiv{\ensuremath{\widehat{\equiv}}\xspace}
1685 \@ifundefined{ergo}%
1686 {\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1687 {\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
1688 \newcommand{\reflect@squig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%
1689 \newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
1690 \newcommand\notergo{\ensuremath{\not\leadsto}}
1691 \newcommand\notogre{\ensuremath{\not\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%

```

## 10 Deprecated Functionality

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

`\*def*`

```

1692 \newcommand\indextoo[2][]{\indi[#1]{#2}}%
1693 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead.}
1694 \newcommand\indexalt[2][]{\aindi[#1]{#2}}%
1695 \PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead.}
1696 \newcommand\twintoo[3][]{\indii[#1]{#2}{#3}}%
1697 \PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead.}
1698 \newcommand\twinalt[3][]{\aindii[#1]{#2}{#3}}%
1699 \PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead.}
1700 \newcommand\atwintoo[4][]{\indiii[#1]{#2}{#3}{#4}}%
1701 \PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instead.}
1702 \newcommand\atwinalt[4][]{\aindiii[#1]{#2}{#3}{#4}}%
1703 \PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instead.}
1704 </package>

```

`\my*graphics`

```

1705 \newcommand\mygraphics[2][]{\includegraphics[#1]{#2}}%
1706 \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics\space instead.}
1707 \newcommand\mycgraphics[2][]{\begin{center}\mygraphics[#1]{#2}\end{center}}%
1708 \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphics\space instead.}
1709 \newcommand\mybgraphics[2][]{\fbox{\mygraphics[#1]{#2}}}%
1710 \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphics\space instead.}
1711 \newcommand\mycbgraphics[2][]{\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}}%
1712 \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphics\space instead.}

```

## 11 Things to deprecate

Module options:

```

1713 \addmetakey*{module}{id} % TODO: deprecate properly
1714 \addmetakey*{module}{load}
1715 \addmetakey*{module}{path}
1716 \addmetakey*{module}{dir}
1717 \addmetakey*{module}{align}[WithTheModuleOfTheSameName]
1718 \addmetakey*{module}{noalign}[true]
1719
1720 \newif\if@insymdef@\@insymdef@false

```

**symdef:keys** The optional argument `local` specifies the scope of the function to be defined. If `local` is not present as an optional argument then `\symdef` assumes the scope of the function is global and it will include it in the pool of macros of the current module. Otherwise, if `local` is present then the function will be defined only locally and it will not be added to the current module (i.e. we cannot inherit a local function). Note, the optional key `local` does not need a value: we write `\symdef[local]{somefunction}[0]{some expansion}`. The other keys are not used in the L<sup>A</sup>T<sub>E</sub>X part.

```

1721 %\srefaddidkey{symdef}% what does this do?
1722 \define@key{symdef}{local}[true]{\@symdeflocaltrue}%
1723 \define@key{symdef}{noverb}[all]{}%
1724 \define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
1725 \define@key{symdef}{specializes}{}%
1726 \addmetakey*{symdef}{noalign}[true]
1727 \define@key{symdef}{primary}[true]{}%
1728 \define@key{symdef}{assocarg}{}%
1729 \define@key{symdef}{bvars}{}%
1730 \define@key{symdef}{bargs}{}%
1731 \addmetakey{symdef}{lang}%
1732 \addmetakey{symdef}{prec}%
1733 \addmetakey{symdef}{arity}%
1734 \addmetakey{symdef}{variant}%
1735 \addmetakey{symdef}{ns}%
1736 \addmetakey{symdef}{args}%
1737 \addmetakey{symdef}{name}%
1738 \addmetakey*{symdef}{title}%
1739 \addmetakey*{symdef}{description}%
1740 \addmetakey{symdef}{subject}%
1741 \addmetakey*{symdef}{display}%
1742 \addmetakey*{symdef}{gfc}%

```

EdN:3

3

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

```

1743 \def\symdef{\@ifnextchar[\@symdef]{\@symdef[]}}%
1744 \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.

---

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

```

1745 \def\@symdef[#1]#2[#3]{%
1746   \@insymdef@true%
1747   \metasetkeys{symdef}{#1}%
1748   \edef\symdef@tmp@optpars{\ifcvoid{symdef@name}{[]}{[name=\symdef@name]}}%
1749   \expandafter\symdecl\symdef@tmp@optpars{#2}%
1750   \@insymdef@false%
1751   \notation[#1]{#2}[#3]%
1752 }% mod@show
1753 \def\symdef@type{Symbol}%
1754 \providecommand{\stDMemph}[1]{\textbf{#1}}

\symvariant \symvariant{<sym>}[<args>]{<var>}{<cseq>} just extends the internal macro
\modules@<sym>@pres@ defined by \symdef{<sym>}[<args>]{...} with a variant
\modules@<sym>@pres@<var> which expands to <cseq>. Recall that this is called
by the macro \<sym>[<var>] induced by the \symdef.

1755 \def\symvariant#1{%
1756   \@ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}%
1757   }%
1758 \def\@symvariant#1[#2]#3#4{%
1759   \notation[#3]{#1}[#2]#4}%
1760 \ignorespacesandpars}%

\abbrdef The \abbrdef macro is a variant of \symdef that does the same on the LATEX
level.

1761 \let\abbrdef\symdef%

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

1762 \newif\if@importing\@importingfalse
1763 \define@key{symi}{noverb}[all]{}%
1764 \define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
1765 \define@key{symi}{specializes}{}%
1766 \define@key{symi}{gfc}{}%
1767 \define@key{symi}{noalign}[true]{}%
1768 \newcommand\symi{\@ifstar\@symi@star\@symi}
1769 \newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
1770   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces}
1771 \newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
1772   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\ignorespaces}
1773 \newcommand\symii{\@ifstar\@symii@star\@symii}
1774 \newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
1775   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespaces}
1776 \newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
1777   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\ignorespaces}
1778 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
1779 \newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%
1780   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignorespaces}
1781 \newcommand\@symiii@star[4][]{\metasetkeys{symi}{#1}%
1782   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\fi\ignorespaces}

```

```

1783 \newcommand\symiv{\@ifstar\@symiv@star\@symiv}
1784 \newcommand\@symiv[5] [] {\metasetkeys{symi}{#1}%
1785   \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4-#5}\fi\ign
1786 \newcommand\@symiv@star[5] [] {\metasetkeys{symi}{#1}%
1787   \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4-#5}

```

`\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`.

```

1788 %\srefaddidkey{importmhmodule}%
1789 \addmetakey{importmhmodule}{mhrepos}%
1790 \addmetakey{importmhmodule}{path}%
1791 \addmetakey{importmhmodule}{ext}% why does this exist?
1792 \addmetakey{importmhmodule}{dir}%
1793 \addmetakey[false]{importmhmodule}{conservative}[true]%
1794 \newcommand\importmhmodule[2] [] {%
1795   \parsemodule@maybesetcodes
1796   \metasetkeys{importmhmodule}{#1}%
1797   \ifx\importmhmodule@dir\@empty%
1798     \edef\@path{\importmhmodule@path}%
1799   \else\edef\@path{\importmhmodule@dir/#2}\fi%
1800   \ifx\@path\@empty% if module name is not set
1801     \@importmodule[] {#2}{export}%
1802   \else%
1803     \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
1804     \ifx\importmhmodule@mhrepos\@empty% if in the same repos
1805       \relax% no need to change mh@currentrepos, i.e., current directory.
1806     \else%
1807       \setcurrentreposinfo\importmhmodule@mhrepos% change it.
1808       \addto@thismodule{x}\noexpand\setcurrentreposinfo{\importmhmodule@mhrepos}}%
1809     \fi%
1810     \@importmodule[\MathHub{\mh@currentrepos/source/\@path}] {#2}{export}%
1811     \setcurrentreposinfo\mh@@repos% after importing, reset to old value
1812     \addto@thismodule{x}\noexpand\setcurrentreposinfo{\mh@@repos}}%
1813   \fi%
1814   \ignorespacesandpars%
1815 }

```

`\usemhmodule`

```

1816 \addmetakey{importmhmodule}{load}
1817 \addmetakey{importmhmodule}{id}
1818 \addmetakey{importmhmodule}{dir}
1819 \addmetakey{importmhmodule}{mhrepos}
1820
1821 \addmetakey{importmodule}{load}
1822 \addmetakey{importmodule}{id}

```

```

1823
1824 \newcommand\usemhmodule[2][]{%
1825 \metasetkeys{importmhmodule}{#1}%
1826 \ifx\importmhmodule@dir\@empty%
1827 \edef\@path{\importmhmodule@path}%
1828 \else\edef\@path{\importmhmodule@dir/#2}\fi%
1829 \ifx\@path\@empty%
1830 \usemodule[id=\importmhmodule@id]{#2}%
1831 \else%
1832 \edef\mh@crepos{\mh@currentrepos}%
1833 \ifx\importmhmodule@mhrepos\@empty%
1834 \else\setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
1835 \usemodule{\@path\@QuestionMark#2}%
1836 %\usemodule[load=\MathHub{\mh@currentrepos/source/\@path},
1837 % id=\importmhmodule@id]{#2}%
1838 \setcurrentreposinfo\mh@crepos%
1839 \fi%
1840 \ignorespacesandpars}

\mhinputref
1841 \newcommand\mhinputref[2][]{%
1842 \edef\mhinputref@first{#1}%
1843 \ifx\mhinputref@first\@empty%
1844 \inputref{#2}%
1845 \else%
1846 \inputref[mhrepos=\mhinputref@first]{#2}%
1847 \fi%
1848 }

```