stex-master.sty: sTeX 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

1 Introduction

TODO

2 User commands

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

3 Implementation

```
⟨*package⟩
% TODO
\newif\if@modules@html@\@modules@html@true
\DeclareOption{omdocmode}{\@modules@html@false}
% Modules:
\newif\ifmod@show\mod@showfalse
\DeclareOption{showmods}{\mod@showtrue}
% sref:
\newif\ifextrefs\extrefsfalse
\DeclareOption{extrefs}{\extrefstrue}
%
\ProcessOptions

\RequirePackage{standalone}
\RequirePackage{xspace}
\RequirePackage{metakeys}
```

3.1 sTeX base

The sTeX logo:

```
\protected\def\stex{%
  \@ifundefined{texorpdfstring}%
  {\let\texorpdfstring\@firstoftwo}%
  {}%
  \texorpdfstring{\raisebox{-.5ex}S\kern-.5ex\TeX}{sTeX}\xspace%
}
\def\sTeX{\stex}
```

and a conditional for LaTeXML:

```
\newif\if@latexml\@latexmlfalse
```

3.2 Paths and URIs

```
\RequirePackage{xstring}
\RequirePackage{etoolbox}
```

`\defpath` `\defpath[optional argument]{macro name}{base path}` defines a new macro which can take another path to formal 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/smglom/sets}`

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

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

3.2.1 Path Canonicalization

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

```
\def\pathsuris@setcatcodes{%
    \edef\pathsuris@oldcatcode@hash{\the\catcode`\#}%
    \catcode`\#=12\relax%
    \edef\pathsuris@oldcatcode@slash{\the\catcode`\/}%
    \catcode`\/=12\relax%
    \edef\pathsuris@oldcatcode@colon{\the\catcode`\:}%
    \catcode`\:=12\relax%
    \edef\pathsuris@oldcatcode@qm{\the\catcode`\?}%
    \catcode`\?=12\relax%
}
\def\pathsuris@resetcatcodes{%
    \catcode`\#\pathsuris@oldcatcode@hash\relax%
    \catcode`\/\pathsuris@oldcatcode@slash\relax%
    \catcode`\:\pathsuris@oldcatcode@colon\relax%
    \catcode`\?\pathsuris@oldcatcode@qm\relax%
}
```

We define some macros for later comparison.

```
\def\@ToTop{..}
\def\@Slash{/}
\def\@Colon{:}
\def\@Space{ }
\def\@QuestionMark{?}
\def\@Dot{.}
\catcode`\&=12
\def\@Ampersand{&}
\catcode`\&=4
\pathsuris@setcatcodes
\def\@Fragment{#}
\pathsuris@resetcatcodes
\catcode`\.=0
.catcode`.\=12
.let.@BackSlash\
.catcode`.\=0
\catcode`\.=12
\edef\old@percent@catcode{\the\catcode`\%}
\catcode`\%=12
\let\@Percent%
\catcode`\%=\old@percent@catcode
```

`\@cpath` Canonicalizes (file) paths:

```
\def\@cpath#1{%
    \edef\pathsuris@cpath@temp{#1}%
    \def\@CanPath{}%
    \IfBeginWith\pathsuris@cpath@temp\@Slash{%
      \@cpath@loop%
      \edef\@CanPath{\@Slash\@CanPath}%
    }{%
        \IfBeginWith\pathsuris@cpath@temp{\@Dot\@Slash}{%
            \StrGobbleLeft\pathsuris@cpath@temp2[\pathsuris@cpath@temp]%
            \@cpath@loop%
        }{%
            \ifx\pathsuris@cpath@temp\@Dot\else%
            \@cpath@loop\fi%
        }%
    }%
    \IfEndWith\@CanPath\@Slash{%
      \ifx\@CanPath\@Slash\else%
        \StrGobbleRight\@CanPath1[\@CanPath]%
      \fi%
    }{}%
}

\def\@cpath@loop{%
    \IfSubStr\pathsuris@cpath@temp\@Slash{%
        \StrCut\pathsuris@cpath@temp\@Slash\pathsuris@cpath@temp@a\pathsuris@cpath@temp%
        \ifx\pathsuris@cpath@temp@a\@ToTop%
            \ifx\@CanPath\@empty%
                \edef\@CanPath{\@ToTop}%
            \else%
                \edef\@CanPath{\@CanPath\@Slash\@ToTop}%
            \fi%
            \@cpath@loop%
        \else%
        \ifx\pathsuris@cpath@temp@a\@Dot%
            \@cpath@loop%
        \else%
        \IfBeginWith\pathsuris@cpath@temp\@ToTop{%
            \StrBehind{\pathsuris@cpath@temp}{\@ToTop}[\pathsuris@cpath@temp]%
            \IfBeginWith\pathsuris@cpath@temp\@Slash{%
                \edef\pathsuris@cpath@temp{\@CanPath\pathsuris@cpath@temp}%
            }{%
                \ifx\@CanPath\@empty\else%
                    \edef\pathsuris@cpath@temp{\@CanPath\@Slash\pathsuris@cpath@temp}
                \fi%
            }%
            \def\@CanPath{}%
            \@cpath@loop%
        }{%
            \ifx\@CanPath\@empty%
                \edef\@CanPath{\pathsuris@cpath@temp@a}%
            \else%
                \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp@a}%
            \fi%
            \@cpath@loop
        }%
        \fi\fi%
    }{
        \ifx\@CanPath\@empty%
            \edef\@CanPath{\pathsuris@cpath@temp}%
        \else%
            \edef\@CanPath{\@CanPath\@Slash\pathsuris@cpath@temp}%
        \fi%
    }%
}
```

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.

```
\newcommand\cpath[1]{%
    \@cpath{#1}%
    \@CanPath%
}
```

`\path@filename`

```
\def\path@filename#1#2{%
    \edef\filename@oldpath{#1}%
    \StrCount\filename@oldpath\@Slash[\filename@lastslash]%
    \ifnum\filename@lastslash>0%
        \StrBehind[\filename@lastslash]\filename@oldpath\@Slash[\filename@oldpath]%
        \edef#2{\filename@oldpath}%
    \else%
        \edef#2{\filename@oldpath}%
    \fi%
}
```

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:

```
\newif\if@iswindows@\@iswindows@false
\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:

```
\newif\if@windowstopath@inpath@
\def\windows@to@path#1{
    \@windowstopath@inpath@false
    \def\windows@temp{}
    \edef\windows@path{#1}
    \ifx\windows@path\@empty\else
        \expandafter\windows@path@loop\windows@path\windows@path@end
    \fi
    \let#1\windows@temp
}
\def\windows@path@loop#1#2\windows@path@end{
    \def\windows@temp@b{#2}
    \ifx\windows@temp@b\@empty
        \def\windows@continue{}
    \else
        \def\windows@continue{\windows@path@loop#2\windows@path@end}
    \fi
    \if@windowstopath@inpath@
        \ifx#1\@BackSlash
            \edef\windows@temp{\windows@temp\@Slash}
        \else
            \edef\windows@temp{\windows@temp#1}
        \fi
    \else
        \ifx#1:
            \edef\windows@temp{\@Slash\windows@temp}
            \@windowstopath@inpath@true
        \else
            \edef\windows@temp{\windows@temp#1}
        \fi
    \fi
    \windows@continue
}
```

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:

```
\def\path@to@windows#1{
    \@windowstopath@inpath@false
    \def\windows@temp{}
    \edef\windows@path{#1}
    \edef\windows@path{\expandafter\@gobble\windows@path}
    \ifx\windows@path\@empty\else
        \expandafter\path@windows@loop\windows@path\windows@path@end
    \fi
    \let#1\windows@temp
}
\def\path@windows@loop#1#2\windows@path@end{
    \def\windows@temp@b{#2}
    \ifx\windows@temp@b\@empty
```

```
        \def\windows@continue{}
    \else
        \def\windows@continue{\path@windows@loop#2\windows@path@end}
    \fi
    \if@windowstopath@inpath@
        \ifx#1/
            \edef\windows@temp{\windows@temp\@BackSlash}
        \else
            \edef\windows@temp{\windows@temp#1}
        \fi
    \else
        \ifx#1/
            \edef\windows@temp{\windows@temp:\@BackSlash}
            \@windowstopath@inpath@true
        \else
            \edef\windows@temp{\windows@temp#1}
        \fi
    \fi
    \windows@continue
}
```

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

3.2.3 Auxiliary methods

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

```
\def\trimstring#1{%
    \edef\pathsuris@trim@temp{#1}%
    \IfBeginWith\pathsuris@trim@temp\@Space{%
        \StrGobbleLeft\pathsuris@trim@temp1[#1]%
        \trimstring{#1}%
    }{%
        \IfEndWith\pathsuris@trim@temp\@Space{%
            \StrGobbleRight\pathsuris@trim@temp1[#1]%
            \trimstring{#1}%
        }{%
            \edef#1{\pathsuris@trim@temp}%
        }%
    }%
}
```

Test:
`»bla blubb«`

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

```
\def\kpsewhich#1#2{\begingroup%
  \edef\kpsewhich@cmd{"|kpsewhich #2"}%
  \everyeof{\noexpand}%
  \catcode`\\=12%
  \edef#1{\@@input\kpsewhich@cmd\@Space}%
  \trimstring#1%
  \if@iswindows@\windows@to@path#1\fi%
  \xdef#1{\expandafter\detokenize\expandafter{#1}}%
\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:

```
\edef\pwd@cmd{\if@iswindows@ -expand-var \percent CD\percent\else -var-value PWD\fi}
\kpsewhich\stex@maindir\pwd@cmd
\edef\stex@mainfile{\stex@maindir\@Slash\jobname}
\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:

```
\def\stex@currfile@stack{}

\def\stex@currfile@push#1{%
    \edef\stex@temppath{#1}%
    \edef\stex@temppath{\expandafter\detokenize\expandafter{\stex@temppath}}%
  \edef\stex@currfile@stack{\stex@currfile\ifx\stex@currfile@stack\@empty\else,\stex@currfile@st
  \IfBeginWith\stex@temppath\@Slash{\@cpath{\stex@temppath}}{%
    \@cpath{\stex@maindir\@Slash#1}%
  }
  \let\stex@currfile\@CanPath%
  \path@filename\stex@currfile\stex@currfilename%
  \StrLen\stex@currfilename[\stex@currfile@tmp]%
  \StrGobbleRight\stex@currfile{\the\numexpr\stex@currfile@tmp+1 }[\stex@currpath]%
  \global\let\stex@currfile\stex@currfile%
  \global\let\stex@currpath\stex@currpath%
  \global\let\stex@currfilename\stex@currfilename%
}
\def\stex@currfile@pop{%
  \ifx\stex@currfile@stack\@empty%
    \global\let\stex@currfile\stex@mainfile%
    \global\let\stex@currpath\stex@maindir%
    \global\let\stex@currfilename\jobname%
  \else%
    \StrCut\stex@currfile@stack,\stex@currfile\stex@currfile@stack%
    \path@filename\stex@currfile\stex@currfilename%
    \StrLen\stex@currfilename[\stex@currfile@tmp]%
    \StrGobbleRight\stex@currfile{\the\numexpr\stex@currfile@tmp+1 }[\stex@currpath]%
    \global\let\stex@currfile\stex@currfile%
    \global\let\stex@currpath\stex@currpath%
    \global\let\stex@currfilename\stex@currfilename%
  \fi%
}
```

\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:

```
\def\stexinput#1{%
    \stexiffileexists{#1}{%
      \stex@currfile@push\stex@temp@path%
      \input{\stex@currfile}%
      \stex@currfile@pop%
    }%
    {%
        \PackageError{stex}{File does not exist (#1): \stex@temp@path}{}%
    }%
}
\def\stexiffileexists#1#2#3{%
  \edef\stex@temp@path{#1}%
  \if@iswindows@\path@to@windows\stex@temp@path\fi%
  \IfFileExists\stex@temp@path{#2}{#3}%
}
\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:

```
\kpsewhich\mathhub@path{--var-value MATHHUB}
\if@iswindows@\windows@to@path\mathhub@path\fi
\ifx\mathhub@path\@empty%
  \PackageWarning{stex}{MATHHUB system variable not found or wrongly set}{}
  \defpath{MathHub}{}
\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.

```
\def\findmanifest#1{
  \@cpath{#1}
  \ifx\@CanPath\@Slash
    \def\manifest@mf{}
  \else\ifx\@CanPath\@empty
      \def\manifest@mf{}
  \else
    \edef\@findmanifest@path{\@CanPath/MANIFEST.MF}
    \if@iswindows@\path@to@windows\@findmanifest@path\fi
    \IfFileExists{\@findmanifest@path}{
      %\message{MANIFEST.MF found at \@findmanifest@path}
      \edef\manifest@mf{\@findmanifest@path}
      \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
    }{
    \edef\@findmanifest@path{\@CanPath/META-INF/MANIFEST.MF}
    \if@iswindows@\path@to@windows\@findmanifest@path\fi
    \IfFileExists{\@findmanifest@path}{
      %\message{MANIFEST.MF found at \@findmanifest@path}
      \edef\manifest@mf{\@findmanifest@path}
      \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
    }{
    \edef\@findmanifest@path{\@CanPath/meta-inf/MANIFEST.MF}
    \if@iswindows@\path@to@windows\@findmanifest@path\fi
    \IfFileExists{\@findmanifest@path}{
      %\message{MANIFEST.MF found at \@findmanifest@path}
      \edef\manifest@mf{\@findmanifest@path}
      \xdef\temp@archive@dir{\expandafter\detokenize\expandafter{\@CanPath}}
    }{
      \findmanifest{\@CanPath/..}
    }}}
  \fi\fi
}
```

Test:
/home/jazzpirate/work/MathHub/smglom/mv/META-INF/MANIFEST.MF

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

```
\def\split@manifest@key{
  \IfSubStr{\manifest@line}{\@Colon}{
      \StrBefore{\manifest@line}{\@Colon}[\manifest@key]
      \StrBehind{\manifest@line}{\@Colon}[\manifest@line]
      \trimstring\manifest@line
      \trimstring\manifest@key
  }{
      \def\manifest@key{}
  }
}
```

the next helper function iterates over lines in `MANIFEST.MF`

```
\def\parse@manifest@loop{
  \ifeof\@manifest
  \else
    \read\@manifest to \manifest@line\relax
    \edef\manifest@line{\expandafter\detokenize\expandafter{\manifest@line}}
    \split@manifest@key
    % id
    \IfStrEq\manifest@key{\detokenize{id}}{
        \xdef\manifest@mf@id{\manifest@line}
    }{
    % narration-base
    \IfStrEq\manifest@key{\detokenize{narration-base}}{
        \xdef\manifest@mf@narr{\manifest@line}
    }{
    % namespace
    \IfStrEq\manifest@key{\detokenize{source-base}}{
        \xdef\manifest@mf@ns{\manifest@line}
    }{
    \IfStrEq\manifest@key{\detokenize{ns}}{
        \xdef\manifest@mf@ns{\manifest@line}
    }{
    % dependencies
    \IfStrEq\manifest@key{\detokenize{dependencies}}{
        \xdef\manifest@mf@deps{\manifest@line}
    }{
    }}}}}
    \parse@manifest@loop
  \fi
}
```

`\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.

```
\newread\@manifest
\def\parsemanifest#1#2{%
  \gdef\temp@archive@dir{}%
  \findmanifest{#2}%
  \begingroup%
    \gdef\manifest@mf@id{}%
    \gdef\manifest@mf@narr{}%
    \gdef\manifest@mf@ns{}%
    \gdef\manifest@mf@deps{}%
    \openin\@manifest\manifest@mf%
    \parse@manifest@loop%
    \closein\@manifest%
  \endgroup%
  \if@iswindows@\windows@to@path\manifest@mf\fi%
  \cslet{#1id}\manifest@mf@id%
  \cslet{#1narr}\manifest@mf@narr%
  \cslet{#1ns}\manifest@mf@ns%
  \cslet{#1deps}\manifest@mf@deps%
  \ifcsvoid{manifest@mf@id}{}{%
    \cslet{#1dir}\temp@archive@dir%
  }%
}
```

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.

```
\def\setcurrentreposinfo#1{%
  \edef\mh@currentrepos{#1}%
  \ifx\mh@currentrepos\@empty%
    \edef\currentrepos@dir{\@Dot}%
    \def\currentrepos@narr{}%
    \def\currentrepos@ns{}%
    \def\currentrepos@id{}%
    \def\currentrepos@deps{}%
  \else%
  \ifcsdef{mathhub@dir@\mh@currentrepos}{%
    \@inmhrepostrue
    \edef\mh@currentrepos{#1}%
    \expandafter\let\expandafter\currentrepos@dir\csname mathhub@dir@#1\endcsname%
    \expandafter\let\expandafter\currentrepos@narr\csname mathhub@narr@#1\endcsname%
    \expandafter\let\expandafter\currentrepos@ns\csname mathhub@ns@#1\endcsname%
    \expandafter\let\expandafter\currentrepos@deps\csname mathhub@deps@#1\endcsname%
  }{%
    \parsemanifest{currentrepos@}{\MathHub{#1}}%
    \@setcurrentreposinfo%
    \ifcsvoid{currentrepos@dir}{\PackageError{stex}{No archive with %
      name #1 found!}{make sure that #1 is directly in your MATHHUB folder %
      and contains a MANIFEST.MF, either directly in #1 or in a meta-inf %
      subfolder.}}{\@inmhrepostrue}%
  }%
  \fi%
}

\def\@setcurrentreposinfo{%
  \edef\mh@currentrepos{\currentrepos@id}%
  \ifcsvoid{currentrepos@dir}{}{%
    \csxdef{mathhub@dir@\currentrepos@id}{\currentrepos@dir}%
    \csxdef{mathhub@narr@\currentrepos@id}{\currentrepos@narr}%
    \csxdef{mathhub@ns@\currentrepos@id}{\currentrepos@ns}%
    \csxdef{mathhub@deps@\currentrepos@id}{\currentrepos@deps}%
  }%
}
```

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

```
\newif\if@inmhrepos\@inmhreposfalse
\ifcsvoid{stex@maindir}{}{
\parsemanifest{currentrepos@}\stex@maindir
\@setcurrentreposinfo
\ifcsvoid{currentrepos@dir}{\PackageWarning{stex}{Not currently in a MathHub repository}{}}{%
  \message{Current repository: \mh@currentrepos}
}
}
```

3.3 Modules

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

Aux:

```
\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`

```
\def\ignorespacesandparsafterend#1\ignorespaces\fi{#1\fi\ignorespacesandpars}
\def\ignorespacesandpars{\ifhmode\unskip\fi\@ifnextchar\par{\expandafter\ignorespacesandpars\@g
```

Options for the `module`-environment:

```
\addmetakey*{module}{title}
\addmetakey*{module}{name}
\addmetakey*{module}{creators}
\addmetakey*{module}{contributors}
\addmetakey*{module}{srccite}
\addmetakey*{module}{ns}
\addmetakey*{module}{narr}
```

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

```
\ifdef{\thesection}{\newcounter{module}[section]}{\newcounter{module}}%
\newrobustcmd\module@heading{%
  \stepcounter{module}%
  \ifmod@show%
  \noindent{\textbf{Module} \thesection.\themodule [\module@name]}%
  \sref@label@id{Module \thesection.\themodule [\module@name]}%
    \ifx\module@title\@empty :\quad\else\quad(\module@title)\hfill\\\fi%
  \fi%
}%
```

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.

```
\newenvironment{module}[1][]{%
  \begin{@module}[#1]%
  \module@heading% make the headings
  \ignorespacesandpars\parsemodule@maybesetcodes}{%
  \end{@module}%
  \ignorespacesafterend%
}%
\ifmod@show\surroundwithmdframed{module@om@common}\fi%
```

Some auxiliary methods:

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

```
  \@ifundefined{this@module}{}{%
    \expandafter\g@addto@macro@safe\this@module{#1}%
  }%
}
\def\addto@thismodulex#1{%
\@ifundefined{this@module}{}{%
  \edef\addto@thismodule@exp{#1}%
  \expandafter\expandafter\expandafter\g@addto@macro@safe%
  \expandafter\this@module\expandafter{\addto@thismodule@exp}%
}}
```

`@module` A variant of the `module` environment that does not create printed representations (in particular no frames).

To compute the ⟨*uri*⟩ 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`.

```
\newif\ifarchive@ns@empty@\archive@ns@empty@false
\def\set@default@ns{%
  \edef\@module@ns@temp{\stex@currpath}%
  \if@iswindows@\windows@to@path\@module@ns@temp\fi%
  \archive@ns@empty@false%
  \ifcsvoid{mh@currentrepos}{\archive@ns@empty@true}%
  {\expandafter\ifx\csname mathhub@ns@\mh@currentrepos\endcsname\@empty\archive@ns@empty@true\f
  }%
  \ifarchive@ns@empty@%
    \edef\@module@ns@tempuri{file\@Colon\@Slash\@Slash\@module@ns@temp}%
  \else%
    \edef\@module@filepath@temppath{\@module@ns@temp}%
    \edef\@module@ns@tempuri{\csname mathhub@ns@\mh@currentrepos\endcsname}%
    \edef\@module@archivedirpath{\csname mathhub@dir@\mh@currentrepos\endcsname\@Slash source}%
    \edef\@module@archivedirpath{\expandafter\detokenize\expandafter{\@module@archivedirpath}}%
    \IfBeginWith\@module@filepath@temppath\@module@archivedirpath{%
      \StrLen\@module@archivedirpath[\ns@temp@length]%
      \StrGobbleLeft\@module@filepath@temppath\ns@temp@length[\@module@filepath@temprest]%
      \edef\@module@ns@tempuri{\@module@ns@tempuri\@module@filepath@temprest}%
    }{}%
  \fi%
  \IfEndWith\@module@ns@tempuri\@Slash{\StrGobbleRight\@module@ns@tempuri1[\@module@ns@tempuri]
  \setkeys{module}{ns=\@module@ns@tempuri}%
}
```

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.

```
\def\set@next@moduleid{%
  \unless\ifcsname namespace@\module@ns @unnamedmodules\endcsname%
      \csgdef{namespace@\module@ns @unnamedmodules}{0}%
  \fi%
  \edef\namespace@currnum{\csname namespace@\module@ns @unnamedmodules\endcsname}%
  \edef\module@temp@setidname{\noexpand\setkeys{module}{name=module\namespace@currnum}}%
  \module@temp@setidname%
  \csxdef{namespace@\module@ns @unnamedmodules}{\the\numexpr\namespace@currnum+1}%
}
```

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 ⟨*name*⟩ (`\module@name`) and uri ⟨*uri*⟩ (`\module@uri`), this defines the following macros:

- `\module@defs@`⟨*uri*⟩ 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 unexpended form `\this@module` that expands to `\module@defs@`⟨*uri*⟩; we define it first and then initialize `\module@defs@`⟨*uri*⟩ as empty.
- `\module@names@`⟨*uri*⟩ will store all symbol names declared in this module.
- `\module@imports@`⟨*uri*⟩ will store the URIs of all modules direclty included in this module
- `\`⟨*uri*⟩ that expands to `\invoke@module{`⟨*uri*⟩`}` (see below).
- `\Module`⟨*name*⟩ that expands to `\`⟨*uri*⟩.

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

```
\newenvironment{@module}[1][]{%
  \metasetkeys{module}{#1}%
  \ifcsvoid{module@name}{\let\module@name\module@id}{}% % TODO deprecate
  \ifx\module@ns\@empty\set@default@ns\fi%
  \ifx\module@narr\@empty%
    \setkeys{module}{narr=\module@ns}%
  \fi%
  \ifcsvoid{module@name}{\set@next@moduleid}{}%
  \let\module@id\module@name% % TODO deprecate
  \edef\module@uri{\module@ns\@QuestionMark\module@name}%
  \csgdef{module@names@\module@uri}{}%
  \csgdef{module@imports@\module@uri}{}%
  \csxdef{\module@uri}{\noexpand\@invoke@module{\module@uri}}%
  \expandafter\global\expandafter\let\csname Module\module@name\expandafter\endcsname\csname\mod
  \edef\this@module{%
    \expandafter\noexpand\csname module@defs@\module@uri\endcsname%
  }%
  \csdef{module@defs@\module@uri}{}%
  \ifcsvoid{mh@currentrepos}{}{%
    \@inmhrepostrue%
    \addto@thismodulex{\expandafter\edef\expandafter\noexpand\csname mh@old@repos@\module@uri\en
      {\noexpand\mh@currentrepos}}%
    \addto@thismodulex{\noexpand\setcurrentreposinfo{\mh@currentrepos}}%
  }%
}{%
  \if@inmhrepos%
  \@inmhreposfalse%
  \addto@thismodulex{\noexpand\setcurrentreposinfo{\expandafter\noexpand\csname mh@old@repos@\mo
  \fi%
}%
```

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

```
\def\@URI{uri}
\def\@invoke@module#1#2{%
  \ifx\@URI#2%
    #1%
  \else%
    % TODO something else
    #2%
  \fi%
}
```

3.4 Inheritance

3.4.1 Selective Inclusion

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

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

`\parsemodule@allow*` The first step is setting up a functionality for registering `\sTeX` macros and environments as part of a module signature.

```
\newif\if@smsmode\@smsmodefalse
\def\parsemodule@escapechar@allowed{true}
\def\parsemodule@allow#1{
  \expandafter\let\csname parsemodule@allowedmacro@#1\endcsname\parsemodule@escapechar@allowed
}
\def\parsemodule@allowenv#1{
  \expandafter\let\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed
}
\def\parsemodule@escapechar@beginstring{begin}
\def\parsemodule@escapechar@endstring{end}
```

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

```
\parsemodule@allow{symdef}
\parsemodule@allow{abbrdef}
\parsemodule@allow{importmodule}
\parsemodule@allowenv{module}
\parsemodule@allow{importmhmodule}
\parsemodule@allow{gimport}
\parsemodule@allowenv{modsig}
\parsemodule@allowenv{mhmodsig}
\parsemodule@allowenv{mhmodnl}
```

```
\parsemodule@allowenv{modnl}
\parsemodule@allow{symvariant}
\parsemodule@allow{symi}
\parsemodule@allow{symii}
\parsemodule@allow{symiii}
\parsemodule@allow{symiv}
\parsemodule@allow{notation}
\parsemodule@allow{symdecl}
%\parsemodule@allow{defi}
%\parsemodule@allow{defii}
%\parsemodule@allow{defiii}
%\parsemodule@allow{defiv}
%\parsemodule@allow{adefi}
%\parsemodule@allow{adefii}
%\parsemodule@allow{adefiii}
%\parsemodule@allow{adefiv}
%\parsemodule@allow{defis}
%\parsemodule@allow{defiis}
%\parsemodule@allow{defiiis}
%\parsemodule@allow{defivs}
%\parsemodule@allow{Defi}
%\parsemodule@allow{Defii}
%\parsemodule@allow{Defiii}
%\parsemodule@allow{Defiv}
%\parsemodule@allow{Defis}
%\parsemodule@allow{Defiis}
%\parsemodule@allow{Defiiis}
%\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*).

```
\catcode`\.=0
.catcode`.\=13
.def.@active@slash{\}
.catcode`.<=1
.catcode`.>=2
.catcode`.{=12
.catcode`.}=12
.def.@open@brace<{>
.def.@close@brace<}>
.catcode`.\=0
\catcode`\.=12
\catcode`\{=1
\catcode`\}=2
\catcode`\<=12
\catcode`\>=12
```

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

`\set@parsemodule@catcodes`

```
  \def\set@parsemodule@catcodes{%
      \global\catcode`\\=13%
      \global\catcode`\#=12%
      \global\catcode`\{=12%
      \global\catcode`\}=12%
      \global\catcode`\$=12%$
      \global\catcode`\^=12%
      \global\catcode`\_=12%
      \global\catcode`\&=12%
      \expandafter\let\@active@slash\parsemodule@escapechar%
  }
```

`\reset@parsemodule@catcodes`

```
  \def\reset@parsemodule@catcodes{%
      \global\catcode`\\=0%
      \global\catcode`\#=6%
      \global\catcode`\{=1%
      \global\catcode`\}=2%
      \global\catcode`\$=3%$
      \global\catcode`\^=7%
      \global\catcode`\_=8%
      \global\catcode`\&=4%
  }
```

`\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.

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

`\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@currcs` 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:

```
\def\parsemodule@escapechar{%
    \def\parsemodule@escape@currcs{}%
    \parsemodule@escape@parse@nextchar@%
}%
```

The next macro simply reads the next character and checks whether it has category code 11. If so, it stores it in `\parsemodule@escape@currcs`. Otherwise, the macro name is complete, it stores the last character in `\parsemodule@last@char` and calls `\parsemodule@escapechar@checkcs`.

```
\long\def\parsemodule@escape@parse@nextchar@#1{%
    \ifcat a#1\relax%
        \edef\parsemodule@escape@currcs{\parsemodule@escape@currcs#1}%
        \let\parsemodule@do@next\parsemodule@escape@parse@nextchar@%
    \else%
      \def\parsemodule@last@char{#1}%
      \def\parsemodule@do@next{\parsemodule@escapechar@checkcs}%
    \fi%
    \parsemodule@do@next%
}
```

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@checkendei` respectively, to check whether the environment being openend/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.

```
\def\parsemodule@escapechar@checkcs{%
    \ifx\parsemodule@escape@currcs\parsemodule@escapechar@beginstring%
        \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkbeginenv\parsemodule@las
    \else%
        \ifx\parsemodule@escape@currcs\parsemodule@escapechar@endstring%
          \edef\parsemodule@do@next{\noexpand\parsemodule@escapechar@checkendenv\parsemodule@las
        \else%
            \expandafter\ifx\csname parsemodule@allowedmacro@\parsemodule@escape@currcs\endcsnar
                \parsemodule@escapechar@allowed%
              \ifx\parsemodule@last@char\@open@brace%
                \expandafter\let\expandafter\parsemodule@do@next@ii\csname\parsemodule@escape@cu
                \edef\parsemodule@do@next{\noexpand\parsemodule@converttoproperbraces\@open@brac
              \else%
                \reset@parsemodule@catcodes%
                \edef\parsemodule@do@next{\expandafter\noexpand\csname\parsemodule@escape@currcs
              \fi%
            \else\def\parsemodule@do@next{\relax\parsemodule@last@char}\fi%
        \fi%
    \fi%
    \parsemodule@do@next%
}
```

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.

```
\expandafter\expandafter\expandafter\def%
\expandafter\expandafter\expandafter\parsemodule@converttoproperbraces%
\expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
  \reset@parsemodule@catcodes%
  \parsemodule@do@next@ii{#1}%
}
```

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.

```
\expandafter\expandafter\expandafter\def%
\expandafter\expandafter\expandafter\parsemodule@escapechar@checkbeginenv%
\expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
    \expandafter\ifx\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed%
        \reset@parsemodule@catcodes%
        \def\parsemodule@do@next{\begin{#1}}%
    \else%
        \def\parsemodule@do@next{#1}%
    \fi%
    \parsemodule@do@next%
}
\expandafter\expandafter\expandafter\def%
\expandafter\expandafter\expandafter\parsemodule@escapechar@checkendenv%
\expandafter\@open@brace\expandafter#\expandafter1\@close@brace{%
    \expandafter\ifx\csname parsemodule@allowedenv@#1\endcsname\parsemodule@escapechar@allowed%
        %\reset@parsemodule@catcodes%
        \def\parsemodule@do@next{\end{#1}}%
    \else%
      \def\parsemodule@do@next{#1}%
    \fi%
    \parsemodule@do@next%
}
```

`\@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.

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

`\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.

```
\newrobustcmd\requiremodules[1]{%
  \mod@showfalse%
  \edef\mod@path{#1}%
  \edef\mod@path{\expandafter\detokenize\expandafter{\mod@path}}%
  \requiremodules@smsmode{#1}%
}%
```

`\requiremodules@smsmode` this reads sTeX modules by setting the category codes for `sms` mode, `\input`ting 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.

```
\newbox\modules@import@tempbox
\def\requiremodules@smsmode#1{%
  \setbox\modules@import@tempbox\vbox{%
    \@smsmodetrue%
    \set@parsemodule@catcodes%
    \hbadness=100000\relax%
    \hfuzz=10000pt\relax%
    \vbadness=100000\relax%
    \vfuzz=10000pt\relax%
    \stexinput{#1.tex}%
    \reset@parsemodule@catcodes%
    }%
    \parsemodule@maybesetcodes%
}
```

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`

```
\newif\if@importmodule@switchrepos\@importmodule@switchreposfalse
\def\importmodule@bookkeeping#1#2#3{%
  \@importmodule@switchreposfalse%
  \metasetkeys{importmodule}{#1}%
  \ifcsvoid{importmodule@mhrepos}{%
    \ifcsvoid{currentrepos@dir}{%
      \let\importmodule@dir\stex@maindir%
    }{%
      \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
    }%
  }{%
    \@importmodule@switchrepostrue%
    \expandafter\let\csname importmodule@oldrepos@#2\endcsname\mh@currentrepos%
    \setcurrentreposinfo\importmodule@mhrepos%
    \edef\importmodule@dir{\currentrepos@dir\@Slash source}%
  }%
  \StrCut{#2}\@QuestionMark\importmodule@subdir\importmodule@modulename%
  \ifx\importmodule@modulename\@empty%
    \let\importmodule@modulename\importmodule@subdir%
    \let\importmodule@subdir\@empty%
  \else%
    \ifx\importmodule@subdir\@empty\else%
      \edef\importmodule@dir{\importmodule@dir\@Slash\importmodule@subdir}%
    \fi%
  \fi%
  \begingroup#3\endgroup%
  \if@importmodule@switchrepos%
    \expandafter\setcurrentreposinfo\csname importmodule@oldrepos@#2\endcsname%
  \fi%
  \ignorespacesandpars%
}
```

`\importmodule`

```
%\srefaddidkey{importmodule}
\addmetakey{importmodule}{mhrepos}
\newcommand\importmodule[2][]{\@@importmodule[#1]{#2}{export}}
\newcommand\@@importmodule[3][]{%
  \importmodule@bookkeeping{#1}{#2}{%
    \@importmodule[\importmodule@dir]\importmodule@modulename{#3}%
  }%
}
```

`\@importmodule` `\@importmodule[`⟨*filepath*⟩`]{`⟨*mod*⟩`}{`⟨*export?*⟩`}` loads ⟨*filepath*⟩`.tex` and activates the module ⟨*mod*⟩. If ⟨*export?*⟩ is `export`, then it also re-exports the `\symdef`s 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`.

```
\newcommand\@importmodule[3][]{%
{%
  \edef\@load{#1}%
  \edef\@importmodule@name{#2}
  \if@smsmode\else\ifcsvoid{Module\@importmodule@name}{%
    \stexiffileexists\@load{\requiremodules\@load}{%
      \requiremodules{\@load\@Slash\@importmodule@name}%
    }%
  }{}\fi%
  \ifx\@load\@empty\else%
    {% TODO
%      \edef\@path{\csname module@#2@path\endcsname}%
%      \IfStrEq\@load\@path{\relax}% if the known path is the same as the requested one do nothi
%      {\PackageError{stex}% else signal an error
%        {Module Name Clash\MessageBreak%
%          A module with name #2 was already loaded under the path "\@path"\MessageBreak%
%          The imported path "\@load" is probably a different module with the\MessageBreak%
%          same name; this is dangerous -- not importing}%
%        {Check whether the Module name is correct}%
%      }%
    }%
  \fi%
  \global\let\@importmodule@load\@load%
}%
\edef\@export{#3}\def\@@export{export}%prepare comparison
%\ifx\@export\@@export\export@defs{#2}\fi% export the module
\ifx\@export\@@export\addto@thismodulex{%
  \noexpand\@importmodule[\@importmodule@load]{#2}{noexport}%
}%
\if@smsmode\else
\ifcsvoid{this@module}{}{%
  \ifcsvoid{module@imports@\module@uri}{
    \csxdef{module@imports@\module@uri}{%
      \csname Module#2\endcsname\@URI%
    }%
  }{%
    \csxdef{module@imports@\module@uri}{%
      \csname Module#2\endcsname\@URI,%
      \csname module@imports@\module@uri\endcsname%
    }%
  }%
}%
\fi\fi%
\if@smsmode\else\activate@defs{#2}\fi% activate the module
}%
```

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/smglom/algebra?band}
macro:->\@invoke@module {http://mathhub.info/smglom/algebra?idempotent}
undefined

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

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

```
\def\activate@defs#1{%
  \ifcsundef{Module#1}{
    \PackageError{stex}{No module with name #1 loaded}{Probably missing an
      \detokenize{\importmodule} (or variant) somewhere?
    }
  }{%
    \ifcsundef{module@\csname Module#1\endcsname\@URI @activated}%
      {\csname module@defs@\csname Module#1\endcsname\@URI\endcsname}{}%
    \@namedef{module@\csname Module#1\endcsname\@URI @activated}{true}%
  }%
}%
```

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

```
\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.

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

`\inputref` `\inputref{`⟨*path to the current file without extension*⟩`}` supports both absolute path and relative path, meanwhile, records the path and the extension (not for relative path).

```
\newrobustcmd\inputref[2][]{%
  \importmodule@bookkeeping{#1}{#2}{%
    %\inputreftrue
    \inputref@preskip%
    \stexinput{\importmodule@dir\@Slash\importmodule@modulename.tex}%
    \inputref@postskip%
  }%
}%
```

3.5 Symbols and Notations

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

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

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

```
\def\define@in@module#1#2{
  \expandafter\edef\csname #1\endcsname{#2}%
  \edef\define@in@module@temp{%
    \def\expandafter\noexpand\csname#1\endcsname%
    {#2}%
  }%
  \if@symdeflocal\else%
    \expandafter\g@addto@macro@safe\csname module@defs@\module@uri%
    \expandafter\endcsname\expandafter{\define@in@module@temp}%
  \fi%
}
```

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

```
\addmetakey{symdecl}{name}%
\addmetakey{symdecl}{verbalization}%

% constructs a symbol name and a verbalization by splitting at exclamation
% points - e.g. \symdecl{symmetric!group} leads to name=symmetric-group
% and verbalization "symmetric group".
\def\symdecl@constructname#1{%
  \def\symdecl@name{}%
  \def\symdecl@verb{}%
  \edef\symdecl@tempname{#1}%
  \symdecl@constructname@loop%
}

\def\symdecl@constructname@loop{%
  \ifx\symdecl@tempname\@empty\else%
    \StrCut\symdecl@tempname!\symdecl@tempfirst\symdecl@tempname%
    \ifx\symdecl@name\@empty%
      \let\symdecl@name\symdecl@tempfirst%
      \let\symdecl@verbalization\symdecl@tempfirst%
      \symdecl@constructname@loop%
    \else%
      \edef\symdecl@name{\symdecl@name-\symdecl@tempfirst}%
      \edef\symdecl@verbalization{\symdecl@verbalization\@Space\symdecl@tempfirst}%
      \symdecl@constructname@loop%
    \fi%
  \fi%
}

\newcommand\symdecl[2][]{%
  \ifcsdef{this@module}{%
    \metasetkeys{symdecl}{#1}%
    \ifcsvoid{symdecl@name}{%
      \ifcsvoid{symdecl@verbalization}{%
```

```
        \symdecl@constructname{#2}%
      }{%
        \edef\symdecl@name{#2}%
      }%
    }{%
      \ifcsvoid{symdecl@verbalization}{\edef\symdecl@verbalization{#2}}{}%
    }%
    \edef\symdef@uri{\module@uri\@QuestionMark\symdecl@name}%
    \ifcsvoid{\symdef@uri}{
      \ifcsvoid{module@names@\module@uri}{%
        \csxdef{module@names@\module@uri}{\symdecl@name}%
      }{%
        \csxdef{module@names@\module@uri}{\symdecl@name,%
          \csname module@names@\module@uri\endcsname}%
      }%
    }{%
    % not compatible with circular dependencies, e.g. test/omdoc/07-modules/smstesta.tex
      \PackageWarning{stex}{symbol already defined: \symdef@uri}{%
        You need to pick a fresh name for your symbol%
      }%
    }%
    \define@in@module\symdef@uri{\noexpand\@invoke@symbol{\symdef@uri}}%
    \define@in@module{#2}{\noexpand\@invoke@symbol{\symdef@uri}}%
    \global\expandafter\let\csname\symdef@uri\@Fragment verb\endcsname\symdecl@verbalization%
  }{%
    \PackageError{stex}{\detokenize{\symdecl} not in a module}{You need to be in a module%
    in order to declare a new symbol}
  }%
  \if@insymdef@\else\parsemodule@maybesetcodes\fi%
}
```

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:

```
\def\modules@getURIfromName#1{%
  \def\notation@uri{}%
  \edef\modules@getURI@name{#1}%
  \if@isuri\modules@getURI@name{%
    \let\notation@uri\isuri@uri%
  }{%
    \ifcsvoid{this@module}{}{%
      \expandafter\modules@getURIfromModule\expandafter{\module@uri}%
      \ifx\notation@uri\@empty%
      \edef\modules@getURI@modules{\csname module@imports@\module@uri\endcsname}%
      \expandafter\@for\expandafter\@I\expandafter:\expandafter=\modules@getURI@modules\do{%
        \ifx\notation@uri\@empty%
          \expandafter\modules@getURIfromModule\expandafter{\@I}%
        \fi%
      }%
    \fi%
    \ifx\notation@uri\@empty%
      \def\notation@extract@uri@currcs{}%
      \notation@extracturifrommacro{#1}%
    \fi%
    \ifx\notation@uri\@empty%
      \PackageError{stex}{No symbol with name, URI or macroname \detokenize{#1} found!}{}%
    \fi%
  }%
 }%
}

\def\if@isuri#1#2#3{%
 \StrCount{#1}\@QuestionMark[\isuri@number]%
 \ifnum\isuri@number=1 %
   \StrCut{#1}\@QuestionMark\@isuri@mod\@isuri@name%
   \ifcsvoid{Module\@isuri@mod}{#3}{%
     \edef\isuri@uri{\csname Module\@isuri@mod\endcsname\@URI\@QuestionMark\@isuri@name}%
     #2%
   }%
 \else%
   \ifnum\isuri@number=2 %
     \edef\isuri@uri{#1}#2\else#3%
   \fi%
 \fi%
}

\def\modules@getURIfromModule#1{%
 \edef\modules@getURI@names{\csname module@names@#1\endcsname}%
 \expandafter\@for\expandafter\@I\expandafter:\expandafter=%
 \modules@getURI@names\do{%
   \ifx\notation@uri\@empty%
     \ifx\@I\modules@getURI@name%
       \edef\notation@uri{#1\@QuestionMark\@I}%
     \fi%
   \fi%
 }%
}

% extracts the full URI from \foo or anything being \ifx-equal to \foo,
% by expanding until we reach \@invoke@symbol{<uri>}
\def\notation@extracturifrommacro#1{%
 \ifcsvoid{#1}{}{%
   \expandafter\let\expandafter\notation@extract@uri@nextcs\csname#1\endcsname%
```

```
    \ifx\notation@extract@uri@nextcs\notation@extract@uri@currcs\else%
      \let\notation@extract@uri@currcs\notation@extract@uri@nextcs%
      \expandafter\notation@extract@uriII\notation@extract@uri@nextcs\notation@end%
    \fi%
  }%
}
\long\def\notation@extract@uriII#1#2\notation@end{%
  \def\notation@extract@check@temp{#2}
  \ifx\@invoke@symbol#1%
    \edef\notation@uri{#2}%
  \else%
    \ifx\notation@extract@check@temp\@empty\else%
      \expandafter\def\expandafter\notation@extract@uri@nextcs\expandafter{#1{#2}}%
      \notation@extract@uri{notation@extract@uri@nextcs}%
    \fi%
  \fi%
}
```

`\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}{...}`

```
% parses the first two arguments:
\providerobustcmd\notation[2][]{%
  \edef\notation@first{#1}%
  \edef\notation@second{#2}%
  \notation@%
}

% parses the last two arguments
\newcommand\notation@[2][0]{%
  \edef\notation@donext{\noexpand\notation@@[\notation@first]%
    {\notation@second}[#1]}%
  \notation@donext{#2}%
}

% parses the notation arguments and wraps them in
% \notation@assoc and \notation@argprec for flexary arguments and precedences
\def\notation@@[#1]#2[#3]#4{%
  \modules@getURIfromName{#2}%
  \notation@parse@params{#1}{#3}
  \let\notation@curr@todo@args\notation@curr@args%
  \def\notation@temp@notation{}%
  \StrLen\notation@curr@args[\notation@temp@arity]%
  \expandafter\renewcommand\expandafter\notation@temp@notation%
    \expandafter[\notation@temp@arity]{#4}%
  % precedence
  \IfSubStr\notation@curr@precs;{%
    \StrCut\notation@curr@precs;\notation@curr@prec\notation@curr@precs%
    \ifx\notation@curr@prec\@empty\def\notation@curr@prec{0}\fi%
  }{%
    \ifx\notation@curr@precs\@empty%
      \ifnum\notation@temp@arity=0\relax%
        \edef\notation@curr@prec{\infprec}%
      \else%
        \def\notation@curr@prec{0}%
      \fi%
    \else%
      \edef\notation@curr@prec{\notation@curr@precs}%
      \def\notation@curr@precs{}%
    \fi%
  }%
  % arguments
  \def\notation@curr@extargs{}
  \def\notation@nextarg@index{1}%
  \notation@do@args%
}

% parses additional notation components for (associative) arguments
\def\notation@do@args{%
  \def\notation@nextarg@temp{}%
  \ifx\notation@curr@todo@args\@empty%
    \notation@after%
  \else%
    % argument precedence
    \IfSubStr\notation@curr@precs{x}{%
      \StrCut\notation@curr@precs{x}\notation@curr@argprec\notation@curr@precs%
    }{%
      \edef\notation@curr@argprec{\notation@curr@precs}%
      \def\notation@curr@precs{}%
    }%
    \ifx\notation@curr@argprec\@empty%
      \let\notation@curr@argprec\notation@curr@prec%
    \fi%
    \StrChar\notation@curr@todo@args1[\notation@argchar]%
    \StrGobbleLeft\notation@curr@todo@args1[\notation@curr@todo@args]%
    \expandafter\ifx\notation@argchar i%
      % normal argument
      \edef\notation@nextarg@temp{{\noexpand\notation@argprec{\notation@curr@argprec}{########\
      \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }
      \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
        \expandafter{\notation@nextarg@temp}%
      \expandafter\expandafter\expandafter\notation@do@args%
    \else%
      % associative argument
      \expandafter\expandafter\expandafter\notation@parse@assocarg%
    \fi%
  \fi%
}

```

```
\def\notation@parse@assocarg#1{%
  \edef\notation@nextarg@temp{{\noexpand\notation@argprec{\notation@curr@argprec}{\noexpand\nota
  \edef\notation@nextarg@index{\the\numexpr\notation@nextarg@index+1 }%
  \expandafter\g@addto@macro@safe\expandafter\notation@curr@extargs%
  \expandafter{\notation@nextarg@temp}%
  \notation@do@args%
}

\protected\def\safe@newcommand#1{%
  \ifdefined#1\expandafter\renewcommand\else\expandafter\newcommand\fi#1%
}

% finally creates the actual macros
\def\notation@after{
  \let\ex\expandafter%
  \ex\ex\ex\def\ex\ex\ex\notation@temp@notation\ex\ex\ex%
    {\ex\notation@temp@notation\notation@curr@extargs}%
  \edef\notation@temp@notation{\noexpand\notation@symprec{\notation@curr@prec}{\ex\unexpanded\ex
  \def\notation@temp@fragment{}%
  \ifx\notation@curr@arity\@empty\else%
    \edef\notation@temp@fragment{arity=\notation@curr@arity}
  \fi%
  \ifx\notation@curr@lang\@empty\else%
    \ifx\notation@temp@fragment\@empty%
      \edef\notation@temp@fragment{lang=\notation@curr@lang}%
    \else%
      \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}%
    \fi%
  \fi%
  \ifx\notation@curr@variant\@empty\else%
    \ifx\notation@temp@fragment\@empty%
      \edef\notation@temp@fragment{variant=\notation@curr@variant}%
    \else%
      \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
    \fi%
  \fi%
  \edef\notation@csname{\notation@uri\@Fragment\notation@temp@fragment}%
  \ifcsvoid{\notation@csname}{%
    \ex\ex\ex\ex\ex\ex\ex\newcommand\ex\ex\ex\csname\ex\ex\ex\notation@csname%
      \ex\ex\ex\endcsname\ex\ex\ex[\ex\notation@temp@arity\ex]%
      \ex{\notation@temp@notation}%
    \edef\symdecl@temps{%
      \noexpand\safe@newcommand\ex\noexpand\csname\notation@csname\endcsname[\notation@temp@ari
    }%
    \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\symdecl@temps}%
    \ex\g@addto@macro@safe\csname module@defs@\module@uri\ex\endcsname\ex{\ex{\notation@temp@no
  }{%
    \PackageWarning{stex}{notation already defined: \notation@csname}{%
      Choose a different set of notation options (variant,lang,arity)%
    }%
  }%
  \parsemodule@maybesetcodes%
}

% parses optional parameters
\def\notation@parse@params#1#2{%
  \def\notation@curr@precs{}%
  \def\notation@curr@args{}%
  \def\notation@curr@variant{}%
  \def\notation@curr@arity{}%
  \def\notation@curr@provided@arity{#2}
  \def\notation@curr@lang{}%
  \def\notation@options@temp{#1}
  \notation@parse@params@%
  \ifx\notation@curr@args\@empty%
    \ifx\notation@curr@provided@arity\@empty%
      \notation@num@to@ia\notation@curr@arity%
    \else%
      \notation@num@to@ia\notation@curr@provided@arity%
    \fi%
  \fi%
}
\def\notation@parse@params@{%
  \IfSubStr\notation@options@temp,{%
    \StrCut\notation@options@temp,\notation@option@temp\notation@options@temp%
    \notation@parse@param%
    \notation@parse@params@%
  }{\ifx\notation@options@temp\@empty\else%
    \let\notation@option@temp\notation@options@temp%
    \notation@parse@param%
  \fi}%
}

%parses an individual optional argument/key-value-pair
\def\notation@parse@param{%
  \trimstring\notation@option@temp%
  \ifx\notation@option@temp\@empty\else%
    \IfSubStr\notation@option@temp={%
      \StrCut\notation@option@temp=\notation@key\notation@value%
      \trimstring\notation@key%
      \trimstring\notation@value%
      \IfStrEq\notation@key{prec}{%
        \edef\notation@curr@precs{\notation@value}%
      }{%
      \IfStrEq\notation@key{args}{%
        \edef\notation@curr@args{\notation@value}%
      }{%
      \IfStrEq\notation@key{lang}{%
        \edef\notation@curr@lang{\notation@value}%
      }{%
      \IfStrEq\notation@key{variant}{%
        \edef\notation@curr@variant{\notation@value}%
      }{%
      \IfStrEq\notation@key{arity}{%
        \edef\notation@curr@arity{\notation@value}%
      }{%
      }}}}}%
    }{%
        \edef\notation@curr@variant{\notation@option@temp}%
    }%
\fi%
}

% converts an integer to a string of 'i's, e.g. 3 => iii,
% and stores the result in \notation@curr@args
\def\notation@num@to@ia#1{%
  \IfInteger{#1}{
    \notation@num@to@ia@#1%
  }{%
    %
  }%
  }
\def\notation@num@to@ia@#1{%
  \ifnum#1>0%
    \edef\notation@curr@args{\notation@curr@args i}%
    \expandafter\notation@num@to@ia@\expandafter{\the\numexpr#1-1\@Space}%
  \fi%
  }
```

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

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

\def\notation@lparen{(}
\def\notation@rparen{)}
\def\infprec{1000000}
\def\neginfprec{-\infprec}

\newcount\notation@downprec
\notation@downprec=\neginfprec

% patching displaymode
\newif\if@displaymode\@displaymodefalse
```

```
\expandafter\everydisplay\expandafter{\the\everydisplay\@displaymodetrue}
\let\old@displaystyle\displaystyle
\def\displaystyle{\old@displaystyle\@displaymodetrue}

\def\dobrackets#1{% avoiding groups at all costs to ensure \parray still works!
  \def\notation@innertmp{#1}%
  \let\ex\expandafter%
  \if@displaymode%
    \ex\ex\ex\left\ex\ex\ex\notation@lparen%
    \ex\notation@resetbrackets\ex\notation@innertmp%
    \ex\right\notation@rparen%
  \else%
    \ex\ex\ex\notation@lparen%
    \ex\notation@resetbrackets\ex\notation@innertmp%
    \notation@rparen%
  \fi%
}

\def\withbrackets#1#2#3{%
  \edef\notation@lparen{#1}%
  \edef\notation@rparen{#2}%
  #3%
  \notation@resetbrackets%
}

\def\notation@resetbrackets{%
  \def\notation@lparen{(}%
  \def\notation@rparen{)}%
}

\def\notation@symprec#1#2{%
  \ifnum#1>\notation@downprec\relax%
    \notation@resetbrackets#2%
  \else%
    \ifnum\notation@downprec=\infprec\relax%
      \notation@resetbrackets#2%
    \else
      \if@inparray@
        \notation@resetbrackets#2
      \else\dobrackets{#2}\fi%
  \fi\fi%
}

\newif\if@inparray@\@inparray@false

\def\notation@argprec#1#2{%
  \def\notation@innertmp{#2}
  \edef\notation@downprec@temp{\number#1}%
  \notation@downprec=\expandafter\notation@downprec@temp%
  \expandafter\relax\expandafter\notation@innertmp%
  \expandafter\notation@downprec\expandafter=\number\notation@downprec\relax%
}
```

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

```
\protected\def\@invoke@symbol#1{%
  \def\@invoke@symbol@first{#1}%
  \symbol@args%
}
```

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

```
\newcommand\symbol@args[1][]{%
  \notation@parse@params{#1}{}%
  \def\notation@temp@fragment{}%
  \ifx\notation@curr@arity\@empty\else%
    \edef\notation@temp@fragment{arity=\notation@curr@arity}%
  \fi%
  \ifx\notation@curr@lang\@empty\else%
    \ifx\notation@temp@fragment\@empty%
      \edef\notation@temp@fragment{lang=\notation@curr@lang}%
    \else%
      \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand lang=\notation@curr@lang}
    \fi%
  \fi%
  \ifx\notation@curr@variant\@empty\else%
    \ifx\notation@temp@fragment\@empty%
      \edef\notation@temp@fragment{variant=\notation@curr@variant}%
    \else%
      \edef\notation@temp@fragment{\notation@temp@fragment\@Ampersand variant=\notation@curr@va
    \fi%
  \fi%
  %
  \ifmmode\def\invoke@symbol@next{\@invoke@symbol@math\@invoke@symbol@first\notation@temp@fragm
  \else\def\invoke@symbol@next{\@invoke@symbol@text\@invoke@symbol@first\notation@temp@fragment
  \invoke@symbol@next%
}
```

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

```
\def\@invoke@symbol@math#1#2{%
  \csname #1\@Fragment#2\endcsname%
}
```

TODO:

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

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

```
\def\setstexlang#1{%
  \def\stex@lang{#1}%
}%
\setstexlang{en}
\def\setstexvariant#1#2{%
  % TODO
}
\def\setstexvariants#1{%
  \def\stex@variants{#1}%
}
```

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}}

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

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

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

$\times {\frac \vara \varb ,\plus {\frac \vara {\frac \vara \varb },\times {\varc ,\plus {\vard ,\vare }}}}$:
$\frac{a}{b} \cdot (\frac{a}{\frac{a}{b}} + c \cdot (d + e))$

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

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

foo bar

3.6 Term References

\ifhref

```
\newif\ifhref\hreffalse%
\AtBeginDocument{%
  \@ifpackageloaded{hyperref}{%
    \hreftrue%
  }{%
    \hreffalse%
  }%
}
```

\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`.

```
\def\termref@maketarget#1#2{%
  % #1: symbol URI
  % #2: text
  \ifhref%
    \hypertarget{sref@#1@target}{#2}%
  \fi%
  \expandafter\edef\csname sref@#1\endcsname##1{%
    \ifhref\noexpand\hyperlink{sref@#1@target}{##1}\fi%
  }%
}
```

\@termref

```
\def\@termref#1#2{%
  % #1: symbol URI
  % #2: text
  \ifcvoid{#1}{%
    \StrCut{#1}\@QuestionMark\termref@mod\termref@name%
    \ifcsvoid{\termref@mod}{%
      \PackageError{stex}{Term reference: Module with URI \termref@mod\ not found}{}%
    }{%
      \PackageError{stex}{Term reference: Module \termref@mod\ exists, but %
        contains no symbol with name \termref@name.%
      }{}%
    }%
  }{%
    \ifcsvoid{sref@#1}{%
      % TODO: No reference point exists!
    }{%
      \csname sref@#1\endcsname{#2}%
    }%
  }%
}
```

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`

```
\newif\ifhref\hreffalse%
\AtBeginDocument{%
  \@ifpackageloaded{hyperref}{%
    \hreftrue%
  }{%
    \hreffalse%
  }%
}%
\newcommand\sref@href@ifh[2]{%
  \ifhref%
    \href{#1}{#2}%
  \else%
    #2%
  \fi%
}%
\newcommand\sref@hlink@ifh[2]{%
  \ifhref%
    \hyperlink{#1}{#2}%
  \else%
    #2%
  \fi%
}%
\newcommand\sref@target@ifh[2]{%
  \ifhref%
    \hypertarget{#1}{#2}%
  \else%
    #2%
  \fi%
}%
```

Then we provide some macros for sTEX-specific crossreferencing

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

```
\def\sref@target{%
  \ifx\sref@id\@empty%
    \relax%
  \else%
```

```
    \edef\@target{sref@\ifcsundef{sref@part}{}{\sref@part @}\sref@id @target}%
    \sref@target@ifh\@target{}%
  \fi%
}%
```

\srefaddidkey `\srefaddidkey[`⟨*keyval*⟩`]{`⟨*group*⟩`}` extends the metadata keys of the group ⟨*group*⟩ with an `id` key. In the optional key/value pairs in ⟨*keyval*⟩ the `prefix` key can be used to specify a prefix. Note that the `id` key defined by `\srefaddidkey[`⟨*keyval*⟩`]{`⟨*group*⟩`}` not only defines `\sref@id`, which is used for referencing by the `sref` package, but also `\`⟨*group*⟩`@id`, which is used for showing metadata via the `showmeta` option of the `metakeys` package.

```
\addmetakey{srefaddidkey}{prefix}
\newcommand\srefaddidkey[2][]{%
  \metasetkeys{srefaddidkey}{#1}%
  \@metakeys@ext@clear@keys{#2}{sref@id}{}% id cannot have a default
  \metakeys@ext@clear@keys{#2}{id}{}%
  \metakeys@ext@showkeys{#2}{id}%
  \define@key{#2}{id}{%
    \edef\sref@id{\srefaddidkey@prefix ##1}%
    %\expandafter\edef\csname #2@id\endcsname{\srefaddidkey@prefix ##1}%
    \csedef{#2@id}{\srefaddidkey@prefix ##1}%
  }%
}%
```

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

```
\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.

```
\ifextrefs%
  \newwrite\refs@file%
\else%
  \def\refs@file{\@auxout}%
\fi%
```

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

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

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

```
\newcommand\sref@label[2]{%
  \sref@def{\ifcsundef{sref@part}{}{\sref@part @}#2}{page}{\thepage}%
  \sref@def{\ifcsundef{sref@part}{}{\sref@part @}#2}{label}{#1}%
}%
```

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

```
\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.

```
\def\sref@id{} % make sure that defined
\newcommand\sref@label@id[1]{%
  \ifx\sref@id\@empty%
    \relax%
  \else%
    \sref@label{#1}{\sref@id}%
  \fi%
}%
```

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

```
\newcommand\sref@label@id@arg[2]{%
  \def\@@id{#2}
  \ifx\@@id\@empty%
    \relax%
  \else%
    \sref@label{#1}{\@@id}%
  \fi%
}%
```

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`.

```
\newenvironment{modsig}[2][]{\def\@test{#1}%
\ifx\@test\@empty\begin{module}[name=#2]\else\begin{module}[name=#2,#1]\fi%
\expandafter\gdef\csname mod@#2@multiling\endcsname{true}%
\ignorespacesandpars}
{\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 \mh@currentrepos.

```
\def\gimport{\@ifstar\@gimport@star\@gimport@nostar}%
\newrobustcmd\@gimport@star[2][]{\def\@test{#1}%
\edef\mh@@repos{\mh@currentrepos}%
\ifx\@test\@empty%
\importmhmodule[conservative,mhrepos=\mh@@repos,path=#2]{#2}%
\else\importmhmodule[conservative,mhrepos=#1,path=#2]{#2}\fi%
\setcurrentreposinfo{\mh@@repos}%
\ignorespacesandpars\parsemodule@maybesetcodes}
\newrobustcmd\@gimport@nostar[2][]{\def\@test{#1}%
\edef\mh@@repos{\mh@currentrepos}%
\ifx\@test\@empty%
\importmhmodule[mhrepos=\mh@@repos,path=#2]{#2}%
\else\importmhmodule[mhrepos=#1,path=#2]{#2}\fi%
\setcurrentreposinfo{\mh@@repos}%
\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.

```
\def\modules@@first#1/#2;{#1}
\newcommand\libinput[1]{%
\ifcsvoid{mh@currentrepos}{%
  \PackageError{mathhub}{current MathHub repository not found}{}}%
  {}
\edef\@mh@group{\expandafter\modules@@first\mh@currentrepos;}
\let\orig@inffile\mh@inffile\let\orig@libfile\mh@libfile
\def\mh@inffile{\MathHub{\@mh@group/meta-inf/lib/#1}}
\def\mh@libfile{\MathHub{\mh@currentrepos/lib/#1}}%
\IfFileExists\mh@inffile{\stexinput\mh@inffile}{}%
\IfFileExists\mh@inffile{}{\IfFileExists\mh@libfile{}{%
  {\PackageError{mathhub}
    {Library file missing; cannot input #1.tex\MessageBreak%
    Both \mh@libfile.tex\MessageBreak and \mh@inffile.tex\MessageBreak%
    do not exist}%
  {Check whether the file name is correct}}}}
\IfFileExists\mh@libfile{\stexinput\mh@libfile\relax}{}
\let\mh@inffile\orig@inffile\let\mh@libfile\orig@libfile}
```

3.11 omdoc/omgroup

```
\newcount\section@level

\section@level=2
\ifdefstring{\omdoc@sty@class}{book}{\section@level=0}{}
```

```
\ifdefstring{\omdoc@sty@class}{report}{\section@level=0}{}
\ifdefstring{\omdoc@sty@topsect}{part}{\section@level=0}{}
\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*⟩.

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

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

```
\newcommand\omgroup@num[2]{%
\edef\@@ID{\sref@id}
\ifx\omgroup@short\@empty% no short title
\@nameuse{#1}{#2}%
\else% we have a short title
\@ifundefined{rdfmeta@sectioning}%
  {\@nameuse{#1}[\omgroup@short]{#2}}%
  {\@nameuse{rdfmeta@#1@old}[\omgroup@short]{#2}}%
\fi%
\sref@label@id@arg{\omdoc@sect@name~\@nameuse{the#1}}\@@ID}
```

`omgroup`

```
\def\@true{true}
\def\@false{false}
\srefaddidkey{omgroup}
\addmetakey{omgroup}{date}
\addmetakey{omgroup}{creators}
\addmetakey{omgroup}{contributors}
\addmetakey{omgroup}{srccite}
\addmetakey{omgroup}{type}
\addmetakey*{omgroup}{short}
\addmetakey*{omgroup}{display}
\addmetakey[false]{omgroup}{loadmodules}[true]
```

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

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

```
\addmetakey{omdoc@sect}{name}
\addmetakey[false]{omdoc@sect}{clear}[true]
\addmetakey{omdoc@sect}{ref}
```

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

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

```
\newcommand\omgroup@redefine@addtocontents[1]{%
%\edef\@@import{#1}%
%\@for\@I:=\@@import\do{%
%\edef\@path{\csname module@\@I  @path\endcsname}%
%\@ifundefined{tf@toc}\relax%
%     {\protected@write\tf@toc{}{\string\@requiremodules{\@path}}}}
%\ifx\hyper@anchor\@undefined% hyperref.sty loaded?
%\def\addcontentsline##1##2##3{%
%\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{#1}{##3}}{\thepage}}}
%\else% hyperref.sty not loaded
%\def\addcontentsline##1##2##3{%
%\addtocontents{##1}{\protect\contentsline{##2}{\string\withusedmodules{#1}{##3}}{\thepage}{\@cu
%\fi
}% hypreref.sty loaded?
```

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

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

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

```
\advance\section@level by 1\relax%
\ifcase\section@level%
\or\omdoc@sectioning[name=\omdoc@part@kw,clear,num]{part}{#2}%
\or\omdoc@sectioning[name=\omdoc@chapter@kw,clear,num]{chapter}{#2}%
\or\omdoc@sectioning[name=\omdoc@section@kw,num]{section}{#2}%
\or\omdoc@sectioning[name=\omdoc@subsection@kw,num]{subsection}{#2}%
\or\omdoc@sectioning[name=\omdoc@subsubsection@kw,num]{subsubsection}{#2}%
```

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

and finally, we localize the sections

```
\newcommand\omdoc@part@kw{Part}
\newcommand\omdoc@chapter@kw{Chapter}
\newcommand\omdoc@section@kw{Section}
\newcommand\omdoc@subsection@kw{Subsection}
\newcommand\omdoc@subsubsection@kw{Subsubsection}
\newcommand\omdoc@paragraph@kw{paragraph}
\newcommand\omdoc@subparagraph@kw{subparagraph}
```

`\setSGvar` set a global variable

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

`\useSGvar` use a global variable

```
\newrobustcmd\useSGvar[1]{%
  \@ifundefined{sTeX@Gvar@#1}
  {\PackageError{omdoc}
    {The sTeX Global variable #1 is undefined}
    {set it with \protect\setSGvar}}
\@nameuse{sTeX@Gvar@#1}}
```

`blindomgroup`

```
\newcommand\at@begin@blindomgroup[1]{}
\newenvironment{blindomgroup}
{\advance\section@level by 1\at@begin@blindomgroup\setion@level}
{\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).

```
\srefaddidkey{omtext}
\addmetakey[]{omtext}{functions}
\addmetakey*{omtext}{display}
\addmetakey{omtext}{for}
\addmetakey{omtext}{from}
\addmetakey{omtext}{type}
\addmetakey*{omtext}{title}
\addmetakey*{omtext}{start}
```

```
\addmetakey{omtext}{theory}
\addmetakey{omtext}{continues}
\addmetakey{omtext}{verbalizes}
\addmetakey{omtext}{subject}
```

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

```
\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.

```
\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.

```
\def\omtext@pre@skip{\smallskip}
\def\omtext@post@skip{}
\newenvironment{omtext}[1][]{\@in@omtexttrue%
  \bgroup\metasetkeys{omtext}{#1}\sref@label@id{this paragraph}%
  \def\lec##1{\@lec{##1}}%
  \omtext@pre@skip\par\noindent%
  \ifx\omtext@title\@empty%
    \ifx\omtext@start\@empty\else%
      \ifx\omtext@display\st@flow\omtext@start\else\stDMemph{\omtext@start}\fi\enspace%
    \fi% end omtext@start empty
  \else\stDMemph{\omtext@title}:\enspace%
    \ifx\omtext@start\@empty\else\omtext@start\enspace\fi%
  \fi% end omtext@title empty
  \ignorespacesandpars}
{\egroup\omtext@post@skip\@in@omtextfalse\ignorespacesandpars}
```

5 Phrase-level Markup

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

```
\srefaddidkey{phrase}
\addmetakey{phrase}{style}
\addmetakey{phrase}{class}
\addmetakey{phrase}{index}
\addmetakey{phrase}{verbalizes}
\addmetakey{phrase}{type}
\addmetakey{phrase}{only}
\newcommand\phrase[2][]{\metasetkeys{phrase}{#1}%
\ifx\prhase@only\@empty\only<\phrase@only>{#2}\else #2\fi}
```

`\coref*`

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

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

\n*lex

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

sinlinequote

```
\def\@sinlinequote#1{``{\sl{#1}}''}
\def\@@sinlinequote#1#2{\@sinlinequote{#2}~#1}
\newcommand\sinlinequote[2][]
{\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.

```
\newcommand\vdec[2][]{#2}
\newcommand\vrest[2][]{#2}
\newcommand\vcond[2][]{#2}
```

EdN:1 \strucdec [1]

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

EdN:2 \impdec [2]

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

7 Block-Level Markup

sblockquote

```
\def\begin@sblockquote{\begin{quote}\sl}
\def\end@sblockquote{\end{quote}}
\def\begin@@sblockquote#1{\begin@sblockquote}
\def\end@@sblockquote#1{\def\@@lec##1{\textrm{##1}}\@lec{#1}\end@sblockquote}
\newenvironment{sblockquote}[1][]
  {\def\@opt{#1}\ifx\@opt\@empty\begin@sblockquote\else\begin@@sblockquote\@opt\fi}
  {\ifx\@opt\@empty\end@sblockquote\else\end@@sblockquote\@opt\fi}
```

sboxquote

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

[1]EdNote: document above
[2]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.

```
\providecommand{\@@lec}[1]{(#1)}
\def\@lec#1{\strut\hfil\strut\null\nobreak\hfill\@@lec{#1}}
\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 a the end of the document. If the at key is given, then we use that for sorting in the index.

```
\addmetakey{omdoc@index}{at}
\addmetakey[false]{omdoc@index}{loadmodules}[true]
\newcommand\omdoc@indexi[2][]{\ifindex%
\metasetkeys{omdoc@index}{#1}%
\@bsphack\begingroup\@sanitize%
\protected@write\@indexfile{}{\string\indexentry%
{\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
\ifx\omdoc@index@loadmodules\@true%
\string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}%
\else #2\fi% loadmodules
}{\thepage}}%
\endgroup\@esphack\fi}%ifindex
\newcommand\omdoc@indexii[3][]{\ifindex%
\metasetkeys{omdoc@index}{#1}%
\@bsphack\begingroup\@sanitize%
\protected@write\@indexfile{}{\string\indexentry%
{\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
\ifx\omdoc@index@loadmodules\@true%
\string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
\string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}%
\else #2!#3\fi% loadmodules
}{\thepage}}%
\endgroup\@esphack\fi}%ifindex
\newcommand\omdoc@indexiii[4][]{\ifindex%
\metasetkeys{omdoc@index}{#1}%
\@bsphack\begingroup\@sanitize%
\protected@write\@indexfile{}{\string\indexentry%
{\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
\ifx\omdoc@index@loadmodules\@true%
\string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
\string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
\string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
\else #2!#3!#4\fi% loadmodules
}{\thepage}}%
\endgroup\@esphack\fi}%ifindex
\newcommand\omdoc@indexiv[5][]{\ifindex%
\metasetkeys{omdoc@index}{#1}%
\@bsphack\begingroup\@sanitize%
\protected@write\@indexfile{}{\string\indexentry%
{\ifx\omdoc@index@at\@empty\else\omdoc@index@at @\fi%
\ifx\omdoc@index@loadmodules\@true%
\string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#2}!%
\string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#3}!%
\string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#4}%
\string\withusedmodules{\@ifundefined{module@id}\used@modules\module@id}{#5}%
\else #2!#3!#4!#5\fi% loadmodules
}{\thepage}}%
\endgroup\@esphack\fi}%ifindex
```

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

`\*indi*`

```
\newcommand\aindi[3][]{{#2}\omdoc@indexi[#1]{#3}}
\newcommand\indi[2][]{{#2}\omdoc@indexi[#1]{#2}}
\newcommand\indis[2][]{{#2}\omdoc@indexi[#1]{#2s}}
\newcommand\Indi[2][]{{\captitalize{#2}}\omdoc@indexi[#1]{#2}}
\newcommand\Indis[2][]{{\capitalize{#2}}\omdoc@indexi[#1]{#2s}}

\newcommand\@indii[3][]{\omdoc@indexii[#1]{#2}{#3}\omdoc@indexii[#1]{#3}{#2}}
\newcommand\aindii[4][]{#2\@indii[#1]{#3}{#4}}
\newcommand\indii[3][]{{#2 #3}\@indii[#1]{#2}{#3}}
\newcommand\indiis[3][]{{#2 #3s}\@indii[#1]{#2}{#3}}
\newcommand\Indii[3][]{{\captitalize{#2 #3}}\@indii[#1]{#2}{#3}}
\newcommand\Indiis[3][]{{\capitalize{#2 #3}}\@indii[#1]{#2}{#3}}

\newcommand\@indiii[4][]{\omdoc@indexiii[#1]{#2}{#3}{#4}\omdoc@indexii[#1]{#3}{#2 (#4)}}
\newcommand\aindiii[5][]{{#2}\@indiii[#1]{#3}{#4}{#5}}
\newcommand\indiii[4][]{{#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
\newcommand\indiiis[4][]{{#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}
\newcommand\Indiii[4][]{\captitalize{#2 #3 #4}\@indiii[#1]{#2}{#3}{#4}}
\newcommand\Indiiis[4][]{\capitalize{#2 #3 #4s}\@indiii[#1]{#2}{#3}{#4}}

\newcommand\@indiv[5][]{\omdoc@indexiv[#1]{#2}{#3}{#4}{#5}}
\newcommand\aindiv[6][]{#2\@indiv[#1]{#3}{#4}{#5}{#6}}
\newcommand\indiv[5][]{{#2 #3 #4 #5}\@indiv[#1]{#2}{#3}{#4}{#5}}
\newcommand\indivs[5][]{{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
\newcommand\Indiv[5][]{\capitalize{#2 #3 #4 #5s}\@indiv[#1]{#2}{#3}{#4}{#5}}
\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 LaTeXML.

```
\newcommand\hateq{\ensuremath{\widehat=}\xspace}
\newcommand\hatequiv{\ensuremath{\widehat\equiv}\xspace}
\@ifundefined{ergo}%
{\newcommand\ergo{\ensuremath{\leadsto}\xspace}}%
{\renewcommand\ergo{\ensuremath{\leadsto}\xspace}}%
\newcommand{\reflect@squig}[2]{\reflectbox{$\m@th#1\rightsquigarrow$}}%
\newcommand\ogre{\ensuremath{\mathrel{\mathpalette\reflect@squig\relax}}\xspace}%
\newcommand\notergo{\ensuremath{\not\leadsto}}
\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

```
\newcommand\indextoo[2][]{\indi[#1]{#2}%
\PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\indi\space instead}
\newcommand\indexalt[2][]{\aindi[#1]{#2}%
\PackageWarning{omtext}{\protect\indextoo\space is deprecated, use \protect\aindi\space instead
\newcommand\twintoo[3][]{\indii[#1]{#2}{#3}%
\PackageWarning{omtext}{\protect\twintoo\space is deprecated, use \protect\indii\space instead}
\newcommand\twinalt[3][]{\aindii[#1]{#2}{#3}%
\PackageWarning{omtext}{\protect\twinalt\space is deprecated, use \protect\aindii\space instead
\newcommand\atwintoo[4][]{\indiii[#1]{#2}{#3}{#4}%
\PackageWarning{omtext}{\protect\atwintoo\space is deprecated, use \protect\indiii\space instea
\newcommand\atwinalt[4][]{\aindii[#1]{#2}{#3}{#4}%
\PackageWarning{omtext}{\protect\atwinalt\space is deprecated, use \protect\aindiii\space instea
⟨/package⟩
```

\my*graphics

```
\newcommand\mygraphics[2][]{\includegraphics[#1]{#2}%
  \PackageWarning{omtext}{\protect\mygraphics\space is deprecated, use \protect\includegraphics
\newcommand\mycgraphics[2][]{\begin{center}\mygraphics[#1]{#2}\end{center}%
  \PackageWarning{omtext}{\protect\mycgraphics\space is deprecated, use \protect\includegraphic
\newcommand\mybgraphics[2][]{\fbox{\mygraphics[#1]{#2}}%
  \PackageWarning{omtext}{\protect\mybgraphics\space is deprecated, use \protect\includegraphic
\newcommand\mycbgraphics[2][]{\begin{center}\fbox{\mygraphics[#1]{#2}}\end{center}%
  \PackageWarning{omtext}{\protect\mycbgraphics\space is deprecated, use \protect\includegraphi
```

11 Things to deprecate

Module options:

```
\addmetakey*{module}{id} % TODO: deprecate properly
\addmetakey*{module}{load}
\addmetakey*{module}{path}
\addmetakey*{module}{dir}
\addmetakey*{module}{align}[WithTheModuleOfTheSameName]
\addmetakey*{module}{noalign}[true]

\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 LaTeX part.

```
%\srefaddidkey{symdef}% what does this do?
\define@key{symdef}{local}[true]{\@symdeflocaltrue}%
\define@key{symdef}{noverb}[all]{}%
\define@key{symdef}{align}[WithTheSymbolOfTheSameName]{}%
\define@key{symdef}{specializes}{}%
\addmetakey*{symdef}{noalign}[true]
\define@key{symdef}{primary}[true]{}%
\define@key{symdef}{assocarg}{}%
\define@key{symdef}{bvars}{}%
\define@key{symdef}{bargs}{}%
\addmetakey{symdef}{lang}%
\addmetakey{symdef}{prec}%
\addmetakey{symdef}{arity}%
\addmetakey{symdef}{variant}%
\addmetakey{symdef}{ns}%
\addmetakey{symdef}{args}%
\addmetakey{symdef}{name}%
\addmetakey*{symdef}{title}%
\addmetakey*{symdef}{description}%
\addmetakey{symdef}{subject}%
\addmetakey*{symdef}{display}%
\addmetakey*{symdef}{gfc}%
```

EdN:3 [3]

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

```
\def\symdef{\@ifnextchar[{\@symdef}{\@symdef[]}}%
\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.

[3] EdNote: MK@MK: we need to document the binder keys above.

```
\def\@@symdef[#1]#2[#3]{%
  \@insymdef@true%
  \metasetkeys{symdef}{#1}%
  \edef\symdef@tmp@optpars{\ifcsvoid{symdef@name}{[]}{[name=\symdef@name]}}%
  \expandafter\symdecl\symdef@tmp@optpars{#2}%
  \@insymdef@false%
  \notation[#1]{#2}[#3]%
}% mod@show
\def\symdef@type{Symbol}%
\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`.

```
\def\symvariant#1{%
  \@ifnextchar[{\@symvariant{#1}}{\@symvariant{#1}[0]}%
  }%
\def\@symvariant#1[#2]#3#4{%
  \notation[#3]{#1}[#2]{#4}%
\ignorespacesandpars}%
```

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

```
\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.

```
\newif\if@importing\@importingfalse
\define@key{symi}{noverb}[all]{}%
\define@key{symi}{align}[WithTheSymbolOfTheSameName]{}%
\define@key{symi}{specializes}{}%
\define@key{symi}{gfc}{}%
\define@key{symi}{noalign}[true]{}%
\newcommand\symi{\@ifstar\@symi@star\@symi}
\newcommand\@symi[2][]{\metasetkeys{symi}{#1}%
  \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2}\fi\ignorespaces
\newcommand\@symi@star[2][]{\metasetkeys{symi}{#1}%
  \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2}\fi\igno
\newcommand\symii{\@ifstar\@symii@star\@symii}
\newcommand\@symii[3][]{\metasetkeys{symi}{#1}%
  \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3}\fi\ignorespa
\newcommand\@symii@star[3][]{\metasetkeys{symi}{#1}%
  \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3}\fi\i
\newcommand\symiii{\@ifstar\@symiii@star\@symiii}
\newcommand\@symiii[4][]{\metasetkeys{symi}{#1}%
  \parsemodule@maybesetcodes\if@importing\else\par\noindent Symbol: \textsf{#2-#3-#4}\fi\ignore
\newcommand\@symiii@star[4][]{\metasetkeys{symi}{#1}%
  \parsemodule@maybesetcodes\if@importing\else\par\noindent Primary Symbol: \textsf{#2-#3-#4}\f
```

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

```
%\srefaddidkey{importmhmodule}%
\addmetakey{importmhmodule}{mhrepos}%
\addmetakey{importmhmodule}{path}%
\addmetakey{importmhmodule}{ext}% why does this exist?
\addmetakey{importmhmodule}{dir}%
\addmetakey[false]{importmhmodule}{conservative}[true]%
\newcommand\importmhmodule[2][]{%
  \parsemodule@maybesetcodes
  \metasetkeys{importmhmodule}{#1}%
  \ifx\importmhmodule@dir\@empty%
    \edef\@path{\importmhmodule@path}%
  \else\edef\@path{\importmhmodule@dir/#2}\fi%
  \ifx\@path\@empty% if module name is not set
    \@importmodule[]{#2}{export}%
  \else%
    \edef\mh@@repos{\mh@currentrepos}% remember so that we can reset it.
    \ifx\importmhmodule@mhrepos\@empty% if in the same repos
      \relax% no need to change mh@currentrepos, i.e, current directory.
    \else%
      \setcurrentreposinfo\importmhmodule@mhrepos% change it.
      \addto@thismodulex{\noexpand\setcurrentreposinfo{\importmhmodule@mhrepos}}%
    \fi%
    \@importmodule[\MathHub{\mh@currentrepos/source/\@path}]{#2}{export}%
    \setcurrentreposinfo\mh@@repos% after importing, reset to old value
    \addto@thismodulex{\noexpand\setcurrentreposinfo{\mh@@repos}}%
  \fi%
  \ignorespacesandpars%
}
```

`\usemhmodule`

```
\addmetakey{importmhmodule}{load}
\addmetakey{importmhmodule}{id}
\addmetakey{importmhmodule}{dir}
\addmetakey{importmhmodule}{mhrepos}

\addmetakey{importmodule}{load}
\addmetakey{importmodule}{id}
```

```
\newcommand\usemhmodule[2][]{%
\metasetkeys{importmhmodule}{#1}%
\ifx\importmhmodule@dir\@empty%
\edef\@path{\importmhmodule@path}%
\else\edef\@path{\importmhmodule@dir/#2}\fi%
\ifx\@path\@empty%
\usemodule[id=\importmhmodule@id]{#2}%
\else%
\edef\mh@@repos{\mh@currentrepos}%
\ifx\importmhmodule@mhrepos\@empty%
\else\setcurrentreposinfo{\importmhmodule@mhrepos}\fi%
\usemodule{\@path\@QuestionMark#2}%
%\usemodule[load=\MathHub{\mh@currentrepos/source/\@path},
%                         id=\importmhmodule@id]{#2}%
\setcurrentreposinfo\mh@@repos%
\fi%
\ignorespacesandpars}
```

\mhinputref

```
\newcommand\mhinputref[2][]{%
  \edef\mhinputref@first{#1}%
  \ifx\mhinputref@first\@empty%
    \inputref{#2}%
  \else%
    \inputref[mhrepos=\mhinputref@first]{#2}%
  \fi%
}
```