The `latex-lab-math` code*

Frank Mittelbach, Joseph Wright, LaTeX Project

v0.6u 2025-09-10

Abstract

This is an experimental prototype. It captures math material (basically okay, but the interfaces for packages aren't yet there) and tags the material (which is not yet anywhere near the final state). That part is provided for experimentation and to gather feedback, etc.

Contents

*

1 Introduction

Todo: update all the documentation! Both here and
(what little there is!) in the implementation section.

Tagging math involves a variety of tasks that require that math is captured before the typesetting:

- When typesetting the math MC-tags and structure commands must be inserted at the begin and the end, and perhaps also around lines or other subparts of the equation.
- The source and/or a mathml-representation of the source must be available so that it can be (perhaps after some preprocessing) be used in an associated file or in an alternate text.
- It must be possible to measure the math for, e.g., a bbox setting.

This file implements capture of all math mode material at the outer level, i.e., a formula is captured in its entirety with inner text blocks (possibly containing further math) absorbed as part of the formula. For example,

```
\[ a \in A \text{ for all $a<5$}  \]
```

would only result in a single capture of the tokens "`a␣\in␣A␣\text{␣for␣all␣$a<5$}`".

2 Math capture

In the current setup

- `$`, `\(...\)` and `$$` grab (through a command in `\everymath`/cseverydisplay) if the boolean `\l_@@_collected_bool` is false. If the boolean is true they behave normally and can for example contain verbatim.
- All (registered) environments grab their body regardless of the state of the boolean. For `equation`, `equation*` and `math` this is a change as they no longer can contain verbatim.

3 Avoiding math capture

In most cases when an environment or command switches into math, the semantic meaning of the content *is* math and grabbing and then tagging that as a Formula is adequate.

But there are exceptions, most prominently with the math shift token `$`.

- `$` can be used to center a box with `\vcenter`, for example in the tabular code. The opening `$` is then often in a different command than the closing `$` and the content of the box should be tagged normally, including all math it contains. This means one wants to avoid that the opening `$` triggers the math grabbing and creates a Formula structure, and after the `$` one wants to switch back to normal text and math capture/tagging.
- `$` is used to place superscripts and subscripts, see the definition of `\textsuperscript` which uses `\ensuremath` internally. Inside the superscript in special cases you may want more tagging (including nested math tagging) but typically it is simple text.
- `$` is used to access a char in a math font.

  For example the `\meta` command uses internally the math commands `\langle` and `\rangle` around its argument:
  `$\langle$\textit{argument}$\rangle$` which gives ⟨*argument*⟩.

  The symbols are then clearly not math. (Beside disabling math tagging one could also use text commands like `\textlangle` and `\textrangle`: ⟨*argument*⟩. Depending on the font that could even look better, but it requires that the font supports the chars, which is not the case for various OpenType fonts.) Additional tagging inside the math should be not needed. If the symbols need an actualtext then a Span can be added around the whole material.

3.1 Options to suppressing math capture and tagging

`\MathCollectTrue`
`\MathCollectFalse`
`\m@th`
`\SuspendTagging`

We are there providing a number of options to suppress the collection/grabbing of the math and with it the tagging. These commands can be used to control locally the tagging of math. The first two are new commands. `\m@th` is a standard LaTeX command used in a number of places to set `\mathsurround` to zero; with active tagging it is also used to identify math that should not be tagged, because it has traditionally be used in exactly the places where math mode is used for its layout charactistics and not for representing a formula.[1]. Details are described below. (`\SuspendTagging` is documented in source2e.pdf.)

[1] This way even code that is not adjusted up for tagging is handled correctly if it uses `\m@th`.

To set \mathsurround without disabling math tagging, use \mathsurround\z@ directly.

3.1.1 Using the trigger token \m@th *inside* the math

If the grabbed math contains the token \m@th the tagging/processing of the math is suppressed. As the math is nevertheless grabbed, this requires that the end math shift token is not hidden in some other command. \m@th sets also \mathsurround to zero, which is often wanted in untagged math anyway.

Text tagging is *not* suppressed inside the math, e.g., \mbox{\emph{text}} will still create an Em-structure. In contrast nested math is not tagged.

To suppress the text tagging \SuspendTagging{} can be used:

- no math tagging, but \emph is tagged:
  `$\m@th\langle\mbox{\emph{if and only if} $x=y$}\rangle$`
- no internal tagging:
  `$\m@th\SuspendTagging{}\langle\mbox{\emph{if and only if}} $x=y$}\rangle$`

The method is quite suitable for symbols and also for sub- and superscripts (as long as they don't contain nested math that should be tagged).

3.1.2 Using \m@th *before* the opening $

\m@th (as defined in the latex-lab-math code) sets also the internal boolean which controls the grabbing to true and so when it is used *before* some math it disables math grabbing and tagging for all following math in the current group (including nested math). Text tagging inside the math is still active, it can be disabled as above with \SuspendTagging{}.

When \m@th is used like this it is possible to reenable the math tagging of nested math, by adding \MathCollectTrue after the opening $.

- no math tagging, but \emph is tagged:
  `{\m@th$\langle\mbox{\emph{if and only if} $x=y$}\rangle$}`
- both nested math and \emph is tagged:
  `{\m@th$\MathCollectTrue\langle\mbox{\emph{if and only if} $x=y$}\rangle$}`

The method is suitable for symbols and sub- and superscripts too (it is actually how superscripts avoid math tagging currently). It can also be used in cases where the end math shift token is hidden in some other command. But it is not so useful for larger boxes as it sets \mathsurround to zero. Grouping should be used to avoid side-effects on following math.

3.1.3 Disabling math tagging with \MathCollectFalse

Without a side-effect on \mathsurround the math grabbing can be suppressed and reenabled by setting the boolean that controls the collecting. So in the following example the math in the \mbox is properly tagged, but the angled brackets are simple text.

```
%stop math grabbing
\MathCollectFalse
$
%(optional) restart math grabbing for nested math
\MathCollectTrue
\langle\mbox{\emph{if and only if}} $x=y$}\rangle
$
%(optional) restart math grabbing for following math
% if there is no grouping
\MathCollectTrue
\quad $x=1$
```

The method is suitable if a box should be centered with `\vcenter` (and is used in array.sty for the tabular code).

4 Math capture interfaces

4.1 Code level interfaces

`\math_register_env:n` `\math_register_env:n {⟨env⟩}`
`\math_register_env:nn` `\math_register_env:nn {⟨env⟩} {⟨options⟩}`

Registers the ⟨*env*⟩ as a math environment which should be captured and made available. This is necessary for all top-level math mode environments: low-level errors may result if these are not correct set up. One or more key–value ⟨*options*⟩ may also be given:

`arg-spec` The argument specification taken by the beginning of the environment; this is used to remove non-mathematical material.

`\math_processor:n` `\math_processor:n {⟨tokens⟩}`

Declares that the captured math content should be passed to the ⟨*tokens*⟩, which will receive the environment type as `#1` and the content as `#2`. The processing is done before the typesetting. It is not applied if `\ifmeasuring@` is true.

4.2 Document level interfaces

`\RegisterMathEnvironment` `\RegisterMathEnvironment [⟨options⟩] {⟨env⟩}`

Registers the ⟨*env*⟩ as a math environment which should be captured and made available. This is necessary for all top-level math mode environments: low-level errors may result if these are not correct set up. One or more key–value ⟨*options*⟩ may also be given:

`arg-spec` The argument specification taken by the beginning of the environment; this is used to remove non-mathematical material.

`\MathCollectTrue` `\MathCollectTrue`
`\MathCollectFalse` `\MathCollectFalse`

This activates/deactivates the math collection of the math shift token. See above for cases when this can be usefull.

5 Math tagging

5.1 Code requirements

The tagging code has to handle

- the embedding into the surrounding. This means
  - closing and reopening MC-chunks
  - closing and reopening text/P-structures
  - handling interferences of the tagging code with penalties and spacing.
- the actual tagging which means to do some or all of the following tasks:
  - setup content for an associated source file
  - setup content for an associated mathml file
  - setup content for the /Alt key
  - setup content for the /ActualText key
  - setup attributes
  - add associated files
  - add a Formula structure
  - surround elements of the equation with mathml structure elements (currently only luatex with luamml)

5.2 Inline math

The embedding code is added through the tagging sockets

- `math/inline/begin`
- `math/inline/end`

The sockets simply push and pop the MC currently. Without tagging they use the noop-plug.

The actual tagging is in done through the tagging sockets

- `math/inline/formula/begin` This socket takes the math as second argument and its code should output it for typesetting. The `default` plug of the socket calls these three internal sockets for the tagging support:
  - `math/content` This should set up the various content variables (empty variables are ignored by the structure code and so can be used to suppress a setting).
  - `math/struct/begin` This calls `\tag_struct_begin:n`. It should also write the associated files if needed.
- `math/inline/formula/end` This socket ends the formula structure(s). The `default` plug calls this internal socket:
  - `tagsupport/math/struct/end`

5.3 Display math

to be written

5.4 Associated Files

The current code allows the attachment of two types of associated file to the Formula structure: the LaTeX source and a MathML representation. Technically both can be attached—AF is an array of file references—in practice there can be problems with PDF consumers: e.g., ngpdf used both and so showed the equation twice (this has been corrected in the newest version) and Foxit seems to see only the first AF in the array (so we attach the mathml as first file).

The LaTeX source can be (and is) attached automatically. It can be suppressed by an option with `math/tex/AF=false`, see below.

The MathML is attached if the files `\jobname-mathml.html` and/or `\jobname-luamml-mathml.html` are found and if they contains a suitable MathML snippet for the current formula. If the files contain more than one suitable snippet (as identified by the hash) the first one is used. `\jobname-luamml-mathml.html` is automatically generated (see below section 5.5) and read after `\jobname-mathml.html`. This means that `\jobname-mathml.html` can contain improved versions of a formula.

The MathML processing can be suppressed globally by emptying the list of mathml files with `math/mathml/sources=`. Locally for a formula `math/mathml/AF=false` can be used.

For a MathML representation a file with such representations must be provided. If the equation is numbered the numbering should be part of the MathML as the `Lbl` substructure is ignored if an MathML is used (see https://github.com/foxitsoftware/PDF_-UA-2).

The MathML representation is given in a special format. It is meant to be a valid html file that can be viewed in a browser. For this it can start with `<!DOCTYPE html><html>` and end with `</html>` It should have the extension `.html`. The ⟨*mathml*⟩ content is read with special catcodes, so can contain ambersands, hashes, comment chars and unmatched braces such as `<mo>{</mo>`

The file should contain a number of representations in this format:

```
<div>
  <h2>\mml ⟨key⟩</h2>
  <p>⟨source⟩</p>
  <p>⟨hash⟩</p>
  <math  ⟨attributes⟩ >
⟨mathml⟩
  </math>
</div>
```

The keywords `<div>`, `<h2>\mml`, `<p>`, `<math`, `</math> </div>` are required as they are used to delimit the arguments by the LaTeX code.

⟨*key*⟩ and ⟨*source*⟩ are only used for debugging, they help to identify the equation referred by this representation. The source should be used correctly escaped `&` and `<` so that if gives valid html!

⟨*attributes*⟩ is not required either, but can, e.g., contain attributes to improve the display in a browser:

```
<math alttext="\mathbf{G}" class="ltx_Math" display="inline">
```

It can also contain the name space declaration:
`xmlns="http://www.w3.org/1998/Math/MathML"`[2]

By default the code tries at the begin of the document to read a file `\jobname-mathml.html` in the `html`-format. The file name can be changed with

```
mathml/setfiles={filename1,filename2}
```

(without extension, `html` is added automatically). If there is a list, all files are loaded. If a file doesn't exist it is ignored, only an info is written to the log.

Currently every MathML-snippet from a file is embedded into the PDF, it is not checked first if it is actually used (simply writing everything to the PDF is a bit easier than keeping everything in memory and also means that the snippets are one after the other in the PDF).

As mentioned above the MathML-AF can be suppressed for the equations in a group with `math/mathml/AF=false`, or completely by setting `math/mathml/sources=` in the preamble.

Files embedded in a PDF can be listed in the attachments panel of a PDF viewer. This is probably not so useful for lots of small files (but one could create collections), but as long as PDF editors or viewers don't offer proper support to access the AF it can help so have them there. The MathML are added by default, but the LaTeX source not. This can be changed with `viewer/pane/mathsource=true` (anywhere in the document) and `viewer/pane/mathml=false` (in the preamble, before the external file is read).

5.5 Automatic mathml creation with luamml

If lualatex and the package unicode-math is used, the package `luamml` is loaded and this package will then automatically generate the file `\jobname-luamml-mathml.html` with MathML representations of all math formulas. This file is then used in subsequent compilations and works also with pdflatex.

The generation of the file can be suppressed (in the preamble) with `math/mathml/luamml/write=false`.

If the package unicode-math is not used, the loading of luamml and with it the generation of the file can be forced with `math/mathml/luamml/load=true` or `math/mathml/luamml/write=true` but be aware that it is then possible that various symbols are mapped to the wrong Unicode code points.

The package luamml is still quite experimental and the output should be checked. The `\jobname-luamml-mathml.html` file may be previewed in a browser although you may need to add additional css or javascript declarations to enable browser support for all mathml constructs.

5.6 Summary of math options

The following options exist to make math more accessible:

ActualText An `ActualText` can be placed on structure elements, but can also be added in the stream on a `BDC` marker with a `Span` tag (normally an independant marker without an MCID number, it is not clear yet if it can be used on a MC-chunk). The content is a text string, typically one or a few Unicode characters. `ActualText`

[2] But it is probably not needed and only blows up the PDF.

is meant to replaces the content and should only be used on small entities, e.g., to define the semantic or the Unicode code point of a symbol. `ActualText` is not supported by all PDF reader. It is also unknown where it should be used at best (in a structure element, or on an independent Span-BDC) and what happens if it is used in more than one place.

enabled by default? False

how to enable/disable No interface yet. `ActualText` can only be added on the Formula structure element by changing the `tagsupport/math/content` or some other socket. For a BDC marker one can, e.g., use

```
\pdf_string_from_unicode:nnN{utf16/hex}{€}\l_tmpa_tl
\pdf_bdc:ee{Span}{/ActualText\l_tmpa_tl}content\pdf_emc:
```

There should be no pagebreak in the ⟨*content*⟩ and the BDC should be correctly nested into tagging, so, e.g., a `\leavevmode` should be issued before the bdc command.

Consumer support in part and in part buggy, needs tests …

Alt Like `ActualText` the `Alt` key can be used on structure elements and on `Span` in the stream. It should contain a description of the content and is mainly meant for images. PDF/UA-1, which views math formulas as illustrations, mandates the key also for `Formula` structure elements.

enabled by default? false unless PDF/UA-1 is detected, then it is enabled in the begindocument/end hook (this will reconsidered when it is clear, that the use of `Alt` does not shadow mathml). It can be enabled for all engines and PDF versions.

enable/disable `\tagpdfsetup{math/alt/use}` (local boolean, so can be used on individual equations)

default value A template text (stored in `\l_@@_content_template_tl`) starting with `LaTeX formula starts`.

user value No interface currently provided. This needs optional arguments or an external setup command. See https://github.com/latex3/tagging-project/discussions/717.

source-AF The LaTeX-source of the equation can be attached as an associated file with mime-type application/Fx-tex. The `AFRelationship` is `Source`. The source is embedded without expansion. This means that targets of references and macros are not resolved. The files are by default not shown in the EmbeddedFiles pane, this can be enabled with `viewer/pane/mathsource=true`. If an A-standard is used, it must be one that allows embedded files, e.g., A-4f.

enabled by default? true for all engines and PDF versions

enable/disable `\tagpdfsetup{math/tex/AF}` (local boolean, so can be used on individual equations)

default value source code including dollars or environment name.

consumer support Currently only ngpdf makes use of it: if there is no mathml it passes the source to mathjax.

luamml The following options make (with lualatex) use of the luamml package. luamml is currently automatically loaded (at the end of the preamble) if unicode-math has been detected. The loading can be forced or suppressed with `\tagpdfsetup{math/mathml/luamml/load=`

luamml affects all math, locally it can be stopped with `math/mathml/ignore`, or by using the commands described in the package.

mathml-AF A mathml representation of the equation can be attached to the structure. The configuration possibilities are rather complex as the keys have to control three different tasks: The *generation* of the file with the mathml fragments, the *reading* and *embedding* of the mathml fragments, and the *association* of a mathml fragment to a specific equation.

generation With pdfLaTeX mathml fragments can not be generated automatically, but a file with dummy fragments for every equation will be written if `\tagpdfsetup{math/mathml/write-dummy}` is issued in the preamble.

With luaLaTeX a file with mathml fragments will be created automatically if the package luamml has been loaded (see above).

reading and embedding By default the code will read and embed mathml from `\jobname-mathml.html` and `\jobname-luamml-mathml.html` in this order and the first fragment with a new hash value will be inserted. The list of sources and their order can be changed with the key `math/mathml/sources`, setting that to an empty value suppresses the loading mathml associated files completely. For efficiency reasons it embeds math fragments directly, there is no check yet if the fragment is actually used.

The files are by default shown in the EmbeddedFiles pane, this can be disabled with `viewer/pane/mathml=false`.

attaching A mathml fragment is currently attached as an associated file to an Formula if the hash of the source matches the hash of the fragment. This is not a perfect test: equations with the same source and so the same hash can have different mathml representation, e.g., if there are references or commands or counters in the equation. This will change in a future version. The attachment can be suppressed locally with `math/mathml/AF=false`. The mathml fragment will still be embedded in the PDF!

TODO: adapt test

mathml structure elements Mathml structure elements can be used in PDF 2.0 directly. In PDF 1.7. one could theoretically use them if one declares a role mapping first, (this can be done with `\tagpdfsetup{role/mathml-tags}`) which maps all to `Span`. But such a role mapping currently breaks reading, e.g. in Adobe, and so it is not recommended.

Automatic generation of structure elements is only possible with lualatex. It requires that the packages luamml and tagpdf have been loaded.

enabled by default? false

enable/disable `\tagpdfsetup{math/mathml/structelem}` (local setting, so can be used with grouping on individual equations).

consumer support Needs more tests.

6 Known current bugs, etc.

6.1 Capture/grabbing problems

1. Incorrect grabbing of `$`-math when there is also explicit `$`-math within a *text environment* that is itself within the math that should all be grabbed. For example,

   ```
   $a\begin{minipage}{1cm}$b$\end{minipage}$
   ```

   would only result in the capture of the tokens "`a\begin␣{minipage}{1cm}`". This can be avoided by an additional brace group:

   ```
   $a{\begin{minipage}{1cm}$b$\end{minipage}}$
   ```

2. Similar incorrect grabbing with `$$` also.

3. The grabbing, for all the display environments (and `\) \]`), needs to deal with nesting: amsmath contains code for this.

4. The math can't contain verbatim and verbatim-like commands. This is nothing new for the amsmath environments but changes `$` and `\[\]` and `equation*` (see, e.g., tagging-project issue #30).

5. Begin and end of the math or math environment can not be hidden in commands. For example `>{$}l<{$}` in a tabular would lead to errors. Therefore in a tabular a slower token-by-token grabbing is used.

6.2 Fake math

For the current state see 3 above. The text here is mainly kept for history.

In a number of places in LaTeX math commands (mainly `$`) is used only for technical reason, e.g., to access a math font, to setup a symbol or to use `\vcenter`.

The code identifies such fake math mostly by making use of the `\m@th` command where two methods are used for the automatic detection:

- After grabbing math content the code checks if the content contains the token `\m@th` and if yes it doesn't call the processor before reinserting the content and perhaps adding tagging code. This method requires that the math can be grabbed (e.g. that the end dollar is visible) and that the `\m@th` is visible. It applies for example in `\@iiiparbox` where the code from `$\vcenter` to `\m@th$` is grabbed an put back. It does not work for example for `tabular` where the dollars and the `\m@th` token are spread around over three commands. `tabular` needs therefore manual intervention.

  A look in the list of usages (in `usage-of-m@th.md`) justifies this approach. All usages are either not math at all, or related to small elements that probably shouldn't be grabbed and processed on their own.

- `\m@th` is redefined so that it sets the boolean `\l_@@_collected_bool` to true. If `\m@th` is used inside math that has been grabbed this doesn't change much as the boolean is set by the grabbing anyway. For usages outside math the benefit is not so clear: The setting avoids that in `\LaTeXe` the epsilon is processed as math, but it also prevents that the content of the amsmath command `\boxed` is processed as math. It means that if one wants to reenable math processing inside some (fake) math one has to do it after `\m@th` calls.

6.2.1 Open problems

1. The grabbing code doesn't pass the info that it detected a `\m@th` token. This means that the tagging code has to do the same check (and doesn't do this in all cases yet).

2. Commands are missing to locally disable the grabbing and processing, e.g., to handle `tabular`.

3. It must be checked if setting the boolean in `\m@th` really makes sense or if commands like `\LaTeXe` should be handled manually.

6.3 Processor

The grabbed math is at first passed to the processor. The processor is not called in a measuring phase (from the amsmath `\ifmeasuring@`) and if the `\m@th` token is detected. It is not quite clear what purpose the processor has. As it is a public interface it can't be used for internal code. And typesetting happens later and the processor can't really change this. Currently it is mostly used for debugging and messages. If the `\m@th` is found the `\l_@@_fakemath_bool` is set, so if the code is changed this must be preserved.

6.4 Other problems

1. The presence of `\m@th` in association with `\ensuremath` does not necessarily indicate fakemath. This is because wanting mathsurround to be zero is very reasonable and common, *even when the math is genuine* (and hence needs to be collected).

   TODO: this claim needs some examples.

2. User-defined environments can create problems; but this area, of new, copied and changed environments, has not yet been developed.

   Joseph wrote, inter alia:
   My thinking [regarding] `\RegisterMathEnvironment`
   - (New) Math environments should not be created-then-
   patched, but only generated by a [(future)] dedicated
   command (`\DeclareMathEnvironment`, presumably)
   - Math environments created with ltcmd [commands] should not be copied, . . .
   - Package authors should be able to manually
   set up math environments with a public boolean.

6.5 Other ToDos

1. Add (some of) the math display commands that were "lifted from plain", e.g., `\displaylines` `\eqalign`(??).

2. The breqn packages changes catcodes and that isn't yet covered by our mechanism.

3. `\intertext` is not correctly taken into account by the code splitting multiline math into subformulas.

`\MaybeStop` (temporarily) not executed, as it is unknown on Chris' system.

7 The Implementation

⟨*kernel⟩

7.1 File declaration

```
\ProvidesFile{latex-lab-math.ltx}
             [\ltlabmathdate\space
              v\ltlabmathversion\space
              Grab all the math(s) and tag it (experiments)]
```

Temp loading ...

```
\AddToHook{begindocument/before}{\RequirePackage{latex-lab-testphase-block}}
```

⟨@@=math⟩

```
\ExplSyntaxOn
```

7.2 Setup

Loading amsmath is an absolute requirement: this avoids needing to have conditional definitions and deals with how to define `\[/\]` neatly. The package is loaded at begin document to allow user to load it with options.

```
\AddToHook{begindocument/before}{ \RequirePackage { amsmath } }
```

7.3 Data structures

`\l__math_collected_bool` Tracks whether math mode material has been collected, which happens inside amsmath environments as well as those handled directly here. If true following math will not grab and/or process. See 2 for details.

```
\bool_new:N \l__math_collected_bool
```

`\l__math_fakemath_bool` Tracks whether math mode material has been identified as fake math during the grabbing phase, which happens currently if the grabbed contents contains the `\m@th` token.

```
\bool_new:N \l__math_fakemath_bool
```

Change first tl name below: 'env' => 'info'? Or do we need an extra storage?

`\g__math_grabbed_env_tl` `\g__math_grabbed_math_tl` `\g__math_grabbed_env_tl` contains the name of the math environment (`math` in the case of inline math, `\g__math_grabbed_math_tl` the math content.

```
\tl_new:N \g__math_grabbed_env_tl
\tl_new:N \g__math_grabbed_math_tl
```

\l__math_tmpa_tl
\l__math_tmpa_skip
\l__math_tmpa_str

Temporary variables

```
\tl_new:N \l__math_tmpa_tl
\skip_new:N \l__math_tmpa_skip
\str_new:N \l__math_tmpa_str
```

\l__math_content_alt_tl
\l__math_content_actual_tl
\l__math_content_AF_tl

Temporary variables to hold math content that should be used in actual or alt text and stored as AF.

```
\tl_new:N \l__math_content_alt_tl
\tl_new:N \l__math_content_actual_tl
\tl_new:N \l__math_content_AF_source_tl
\tl_new:N \l__math_content_AF_source_tmpa_tl
\tl_new:N \l__math_content_AF_mathml_tl
```

7.4 Tagging tools

The following commands implement small tagging code chunks. This should probably be collected and moved into tagpdf later.

__tag_tool_close_P:

This closes a P/text-chunk, both the MC and the structure and increases the counter manually.

```
\cs_new_protected:Npn \__tag_tool_close_P:
  {
    \tag_if_active:T
     {
       \tag_mc_end: %end P-chunk, should perhaps be \tag_mc_end_push: ...
         \__tag_gincr_para_end_int:
         \__tag_check_para_end_show:nn{red}{} %debug: show para
         \tag_struct_end:
     }
  }
```

(*End of definition for* `\__tag_tool_close_P:`.)

We add also an attribute.

```
\tl_new:N\l__math_attribute_class_tl
\tagpdfsetup
      {role/new-attribute = {inline}    {/O /Layout /Placement/Inline},
       role/new-attribute = {display}   {/O /Layout /Placement/Block},
      }
```

7.5 Code related to AF

Booleans to handle the options.

\l__tag_math_texsource_AF_bool
\l__tag_math_texsource_pane_bool
\l__tag_math_mathml_AF_bool
\g__tag_math_mathml_AF_bool
\l__tag_math_mathml_pane_bool
\l__tag_math_alt_bool
\g__tag_math_luamml_tl

The variable \g__tag_math_luamml_tl is initially 0 and the user key can set it to -1 or 1. This allows to distinguish the unset case from a value set by the user.

```
\bool_new:N\l__tag_math_texsource_AF_bool
\bool_new:N\l__tag_math_texsource_pane_bool
\bool_new:N\l__tag_math_mathml_AF_bool
\bool_new:N\g__tag_math_mathml_AF_bool
\bool_new:N\l__tag_math_mathml_pane_bool
\bool_new:N\l__tag_math_alt_bool
\tl_new:N\g__tag_math_luamml_tl
\tl_gset:Nn\g__tag_math_luamml_tl {0}
```

\g__math_mathml_total_int
\g__math_mathml_int
\g__math_math_total_int
\g__math_mathml_AF_found_int
\g__math_mathml_AF_attached_int

\g__math_mml_total_int records the mathml fragments read in. \g__math_mml_int records the mathml fragments read in with a different hash. \g__math_AF_total_int records the number of math structures that try to attach a mathml AF. \g__math_AF_found_int records the number of math structures for which a fitting mathml is found. \g__math_AF_attached_int records the number of math structures which got a mathml fragment (if mathml-AF are not disabled locally this should be the equal to the previous number.

```
\int_new:N\g__math_mathml_total_int
\int_new:N\g__math_mathml_int
\int_new:N\g__math_math_total_int
\int_new:N\g__math_mathml_AF_found_int
\int_new:N\g__math_mathml_AF_attached_int
```

\l__tag_math_mathml_files_clist

A sequence to store the file list for the mathml. We also check the luamml file.

```
\clist_new:N\l__tag_math_mathml_files_clist
\clist_put_right:Ne\l__tag_math_mathml_files_clist
  {\c_sys_jobname_str-mathml,\c_sys_jobname_str-luamml-mathml}
```

This is the internal variant of the \mml command.

__math_AF_mml:nnnn

```
\cs_new_protected:Npn \__math_AF_mml:nnnn #1 #2 #3 #4
```

```
%#1 number, #2 tex source for debugging, #3 hash, #4 mathml
  {
    \int_gincr:N \g__math_mathml_total_int
```

mathml with the same hash should be included only once:

```
    \tl_if_exist:cF { g__math_mathml_#3_tl }
     {
       \int_gincr:N \g__math_mathml_int
```

a simple Desc key, take care that it is a valid string!

```
       \pdfdict_put:nne {l_pdffile/Filespec} {Desc}{(mathml-#1)}
       \pdffile_embed_stream:nnN {#4}{mathml-#1.xml}\l__math_tmpa_tl
```

not strictly necessary but makes the files visible in the file attachment page

```
       \bool_if:NT \l__tag_math_mathml_pane_bool
        {\pdfmanagement_add:nne {Catalog/Names}{EmbeddedFiles}{\l__math_tmpa_tl}}
       \tl_new:c{g__math_mathml_#3_tl}
       \tl_gset_eq:cN{g__math_mathml_#3_tl}\l__math_tmpa_tl
     }
  }
```

(*End of definition for* `\__math_AF_mml:nnnn`.)

The html reader.

```
\cs_new_protected:Npn \__math_AF_html_reader:w#1</h2>#2<p>#3</p>#4<p>#5</p>#6<math{
  \begingroup
   \char_set_catcode_other:N\{
   \char_set_catcode_other:N\}
   \char_set_catcode_other:N\#
   \char_set_catcode_other:N\%
   \char_set_catcode_other:N\^
   \__math_AF_html_reader_verb:w{#1}{#3}{#5}<math
}

\cs_new_protected:Npn\__math_AF_html_reader_verb:w#1#2#3#4~</div>{
  \endgroup
   \__math_AF_mml:nnnn{#1}{#2}{#3}{#4}
   }
```

As with luatex we write two files we define a few constants for the shared texts.

```
\c__math_mathml_write_init_tl
\l__math_mathml_write_before_tl
\c__math_mathml_write_after_tl
\c__math_mathml_write_final_tl
\tl_const:Nn \c__math_mathml_write_init_tl
  {
    <!DOCTYPE~html>
    \iow_newline:
    <html~ xmlns="http://www.w3.org/1999/xhtml">
    \iow_newline:
  }
\tl_new:N \l__math_mathml_write_before_tl
```

```
\tl_const:Nn \c__math_mathml_write_after_tl
  {
    \iow_newline:
    </div>
    \iow_newline:
  }
\tl_const:Nn \c__math_mathml_write_final_tl
  {
    </html>
  }
```

(*End of definition for* `\c__math_mathml_write_init_tl` *and others.*)

`athml/write/prepare` (*tag socket*) To prepare the hash and the starting command we use a socket, so that both the dummy and luamml can make use of it.

```
\NewTaggingSocket{math/mathml/write/prepare}{0}
```

`On` (*plug*)

```
\NewTaggingSocketPlug{math/mathml/write/prepare}{On}
  {
    \str_set:NV\l__math_tmpa_str\l__math_content_AF_source_tl
    \str_replace_all:Nnn\l__math_tmpa_str{&}{&amp;}
    \str_replace_all:Nnn\l__math_tmpa_str{<}{&lt;}
    \tl_set:Nn \l__math_mathml_write_before_tl
      {
        <div>
        \iow_newline:
        <h2>\c_backslash_str mml\c_space_tl \int_use:N \g__math_math_total_int </h2>
        \iow_newline:
        <p>\l__math_tmpa_str</p>
        \iow_newline:
        <p>\l__math_content_hash_tl </p>
        \iow_newline:
      }
  }
```

With luatex we automatically generate mathml with luamml if the package can be loaded and unicode-math is detected. We start the process in the begindocument/end hook so that the reading from a previous compilation can happen before!

For other engines, for future name changes and in case luamml is not loaded we provide some commands

```
\cs_new_protected:Npn\__math_provide_luamml_commands:
  {
    \providecommand\luamml_flag_structelem:{}
    \cs_if_free:NT \luamml_structelem:
     {
       \cs_set_eq:NN\luamml_structelem:\luamml_flag_structelem:
     }
    \providecommand\luamml_flag_process:{}
    \cs_if_free:NT \luamml_process:
     {
```

```
      \cs_set_eq:NN\luamml_process:\luamml_flag_process:
    }
   \providecommand\luamml_flag_ignore:{}
   \cs_if_free:NT \luamml_ignore:
    {
      \cs_set_eq:NN\luamml_ignore:\luamml_flag_ignore:
    }
 }
\sys_if_engine_luatex:TF
 {
   \AddToHook{begindocument/before}
    {
      \str_case:on \g__math_luamml_load_tl
        {
          { 1 } {
                  \RequirePackage  { luamml }
                  \AddToHook{begindocument/end}
                   {
                     \__math_luamml_activate_write:
                   }
                }
          {-1 } {
                  \AddToHook{begindocument/end}
                   {
                    \msg_note:nnnn { tag }
                    { luamml-status }{ disabled }{ not~create }
                   }
                }
          { 0 }
          {
            \@ifpackageloaded { unicode-math }
             {
               \RequirePackage  { luamml }
               \AddToHook{begindocument/end}
                 {
                   \__math_luamml_activate_write:
                 }
             }
             { \msg_warning:nn { tag }{ unicode-math-missing } }
          }
        }
        \__math_provide_luamml_commands:
    }
}
{
  \AddToHook{begindocument/before}
   {
     \__math_provide_luamml_commands:
   }
}
\msg_new:nnn { tag }{ luamml-status }
 {
   luamml~has~been~#1~and~will~#2~an~MathML~file.
 }
```

```
\msg_new:nnn { tag }{ unicode-math-missing }
  {
    The~package~unicode-math~is~missing\\
    luamml~will~not~create~an~MathML~file.\\
    To~avoid~this~warning~load~unicode-math~\\
    or~disable~luamml~with~\\
    \tl_to_str:n{\tagpdfsetup{math/mathml/luamml/load=false}}\\
    or~force~luamml~with~\\
    \tl_to_str:n{\tagpdfsetup{math/mathml/luamml/load=true}}
  }
\cs_new_protected:Npn \__math_luamml_activate_write:
 {
   \bool_if:NT \g__math_luamml_write_bool
     {
```

to avoid that nothing is written in the first run, we must activate the sockets:

```
       \bool_gset_true:N\g__tag_math_mathml_AF_bool
       \AssignTaggingSocketPlug{math/struct/begin}{mathml-AF}
       \AssignTaggingSocketPlug{math/struct/end}{mathml-AF}
       \int_set:Nn \l__luamml_pretty_int { 7 }
       \RegisterFamilyMapping\symsymbols{oms}
       \RegisterFamilyMapping\symletters{oml}
       \AssignTaggingSocketPlug{math/mathml/write/prepare}{On}
       \iow_new:N   \g__math_luamml_iow
       \iow_open:Nn \g__math_luamml_iow {\c_sys_jobname_str-luamml-mathml.html}
       \iow_now:Ne  \g__math_luamml_iow { \c__math_mathml_write_init_tl  }
       \cs_new:Npn  \__math_luamml_output_hook:n  ##1
         {
           \tl_if_empty:NF \l__math_mathml_write_before_tl
             {
```

We check here if the current group level is equal to the one stored for the outer math. We only write output if that is the case.

Currently in LaTeX, the `\math@level` is increased for every nested math mode (via `\frozen@everymath`. However, to make the code below work correctly we undo that in the case of "fake math", i.e., in math mode that is only entered to make use of super or subscript positioning of text or for `\vcenter`ing text, i.e., if it is not really a "math formula". We therefore provide a special declaration, `\UseMathForPositioningText`, to indicate that the directly following `$` represents such "fake math".

```
               \int_compare:nNnT
                { \@math@level } = { 1 }
                {
                  \iow_now:Ne \g__math_luamml_iow
                   {
                     \l__math_mathml_write_before_tl
                     ##1
                     \c__math_mathml_write_after_tl
                   }
                }
              }
         }
       \__luamml_register_output_hook:N \__math_luamml_output_hook:n
```

At the end of the document we must finish and close the file:

```
      \AddToHook{enddocument/afterlastpage}
        {
          \iow_now:Ne \g__math_luamml_iow
            { \c__math_mathml_write_final_tl }
          \iow_close:N \g__math_luamml_iow
        }
      \msg_note:nnnn { tag }
        { luamml-status }{ enabled }{ create }
    }
}
```

`\UseMathForPositioningText`

The `\UseMathForPositioningText` command indicates that a directly following `$` is not a real math formula, but that the math mode is only entered to position ordinary text with the help of built in TEX algorithms normally used for math, e.g., `\vcenter`, super and subscripts.

Signalling that special usage is necessary in the case tagged PDF is produced, because then such math mode should not generate a "formula" structure.

The command requires that it is immediately followed by `$`; anything else will result in a low-level TEX error. As it is not a user but a developer command, this seems acceptable for the sake of fast runtime execution.

The way it is implemented it can be used in legacy code in front of any `$` that switches to math mode for non-math purposes. If tagging is active it will ensure that this particular math mode content is not treated as math with respect to tagging (though nested math still is). If tagging is inactive the command is simply ignored.

```
\cs_set_protected:Npn \UseMathForPositioningText $ {
```

Before the math mode is started we ensure that its content is not collected by the grabber.

```
  \bool_if:NTF \l__math_collected_bool
    { $ }
    {
      \bool_set_true:N  \l__math_collected_bool
      $
```

Once inside math mode we reenable grabbing, so that any nested math (within text) is grabbed.

```
      \bool_set_false:N \l__math_collected_bool
```

In addition we decrement the math level (which was automatically incremented when entering math mode) so that this math mode is transparent in `\__math_luamml_activate_write:`.

```
      \int_decr:N \@math@level
    }
}
```

(*End of definition for* `\UseMathForPositioningText`. *This function is documented on page* ??.)

`\@textsuperscript` `\@textsubscript` Updates to the kernel macros.

```
\def\@textsuperscript#1{%
  {\m@th\@unreal@math{^{\mbox{\fontsize\sf@size\sf@size#1}}}}}
```

For the math subscript we have to use `\sb` because this file is processed with `\ExplSyntaxOn`. Alternatively, we could have used `\c_math_subscript_token`, but that looks odd as long as the rest is in 2e style):

```
\def\@textsubscript#1{%
  {\m@th\@unreal@math{\sb{\mbox{\fontsize\sf@size\sf@size#1}}}}}
```

(*End of definition for* `\@textsuperscript` *and* `\@textsubscript`. *These functions are documented on page* ??.)

`\@unreal@math` `\@ensured@unreal@math`

```
\protected\def\@unreal@math{%
  \ifmmode
    \expandafter\@firstofone
  \else
    \expandafter\@ensured@unreal@math
  \fi}
```

If we are not in math mode we start one. Because of `\m@th` outside this will not be collected, but inside we want to collect again in case the argument contains nested math.

```
\long\def\@ensured@unreal@math#1{
  $
  \bool_set_false:N \l__math_collected_bool
```

We also decrement the math level so that any inner math is subject to MathML handling if appropriate.

```
  \int_decr:N \@math@level
  #1 $
}
```

(*End of definition for* `\@unreal@math` *and* `\@ensured@unreal@math`. *These functions are documented on page* ??.)

And now keys to activate/deactivate luamml feature

`\g__math_luamml_load_tl` This variable will be used to suppress the loading of luamml altogether.

```
\tl_new:N  \g__math_luamml_load_tl
\tl_gset:Nn \g__math_luamml_load_tl {0}
```

`\g__math_luamml_write_bool` This variable decides if luamml writes a mathml altogether.

```
\bool_new:N  \g__math_luamml_write_bool
\bool_gset_true:N \g__math_luamml_write_bool
```

__math_luamml_ignore:
__math_luamml_structelem:

Internal variants of the luamml commands, that can be remapped if needed.

```
\cs_new:Npn\__math_luamml_structelem:{}
\cs_new:Npn\__math_luamml_ignore:{}
```

(*End of definition for* `\__math_luamml_ignore:` *and* `\__math_luamml_structelem:`.)

```
\msg_new:nnn { tag }{ PDF-2.0-recommended }
  {
    The~key~#1~will~not~work~properly~with~PDF~#2.\\
    Switching~to~PDF~2.0~is~recommended.
  }
\keys_define:nn { __tag / setup }
   {
```

At first a key to suppress the loading altogether

```
     math/mathml/luamml/load .choice: ,
     math/mathml/luamml/load/true  .code:n = {\tl_gset:Nn \g__math_luamml_load_tl{1}},
     math/mathml/luamml/load/false .code:n = {\tl_gset:Nn \g__math_luamml_load_tl{-
1}},
     math/mathml/luamml/load .default:n = true,
     math/mathml/luamml/load .usage:n=preamble,
```

A key to activate math structure elements.

```
     math/mathml/structelem .choice:,
     math/mathml/structelem/true .code:n =
      {
        \pdf_version_compare:NnT < {2.0}
         {
          \msg_warning:nnne { tag }{ PDF-2.0-recommended }
           { math/mathml/structelem }{ \pdf_version: }
         }
        \cs_set:Npn\__math_luamml_structelem:{\luamml_structelem:}
        \cs_set:Npn\__math_luamml_ignore:{\luamml_ignore:}
      },
     math/mathml/structelem/false .code:n =
      {
        \cs_set_eq:NN\__math_luamml_structelem:\prg_do_nothing:
        \cs_set_eq:NN\__math_luamml_ignore:\prg_do_nothing:
      },
     math/mathml/structelem .default:n = true,
```

and a key to call the ignore flag. This should only be used locally.

```
     math/mathml/ignore .code:n = {\luamml_ignore:},

     math/mathml/luamml/write .choice:,
     math/mathml/luamml/write/true .code:n =
      {
        \tl_gset:Nn \g__math_luamml_load_tl{1}
        \bool_gset_true:N \g__math_luamml_write_bool
      },
     math/mathml/luamml/write/false .code:n =
      {
```

```
        \bool_gset_false:N \g__math_luamml_write_bool
      },
    math/mathml/luamml/write .default:n = true,
    math/mathml/luamml/write .usage:n=preamble,
```

alias keys for compatibility

```
    math/mathml/luamml .bool_gset:N = \g__math_luamml_write_bool,
    math/mathml/luamml .usage:n=preamble
  }
```

`math/mathml/write` (*tag socket*) This writes a html-dummy with the hash and the math content. This should be optional, so it uses a socket that can be disabled

```
\NewTaggingSocket{math/mathml/write}{0}
```

`On` (*plug*)

```
\NewTaggingSocketPlug{math/mathml/write}{On}
 {
    \iow_now:Ne \g__math_writedummy_iow
     {
      \l__math_mathml_write_before_tl
      <math~ xmlns="http://www.w3.org/1998/Math/MathML"></math>
      \c__math_mathml_write_after_tl
     }
 }
```

And now a key to activate the socket.

```
\keys_define:nn { __tag / setup }
  {
    math/mathml/write-dummy .code:n =
      {
        \bool_gset_true:N \g__tag_math_mathml_AF_bool
        \tl_if_exist:NF\g__math_writedummy_iow
         {
           \iow_new:N  \g__math_writedummy_iow
           \iow_open:Nn \g__math_writedummy_iow
            {
              \c_sys_jobname_str-mathml-dummy.html
            }
           \iow_now:Ne \g__math_writedummy_iow
            {
              \c__math_mathml_write_init_tl
            }
           \AssignTaggingSocketPlug{math/mathml/write/prepare}{On}
           \AssignTaggingSocketPlug{math/mathml/write}{On}
           \AddToHook{enddocument/afterlastpage}
            {
              \iow_now:Ne \g__math_writedummy_iow
                { \c__math_mathml_write_final_tl }
              \iow_close:N \g__math_writedummy_iow
            }
```

```
          }
       },
     math/mathml/write-dummy .usage:n=preamble
   }
```

__math_AF_process_mathml_files:

```
\box_new:N\l__math_tmpa_box
\cs_new_protected:Npn \__math_AF_process_mathml_files:
 {
   \hbox_set:Nn \l__math_tmpa_box
    {
      \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
      \pdfdict_put:nne
       { l_pdffile }{Subtype}
       { \pdf_name_from_unicode_e:n{application/mathml+xml} }
      \char_set_catcode_other:N \#
      \cs_set_eq:NN\mml \__math_AF_html_reader:w
      \clist_map_inline:Nn \l__tag_math_mathml_files_clist
        {
          \file_if_exist:nTF {##1.html}
            {
              \typeout{Info:~reading~mathml~file~##1}
              \file_input:n {##1.html}
              \bool_gset_true:N\g__tag_math_mathml_AF_bool
            }
            {
              \typeout{Info:~mathml~file~##1~does~not~exist}%info message
            }
        }
    }
   \box_clear:N \l__math_tmpa_box
    \bool_if:NT\g__tag_math_mathml_AF_bool
      {
        \typeout{Info:~Activating~mathml~support}
        \AssignTaggingSocketPlug{math/struct/begin}{mathml-AF}
        \AssignTaggingSocketPlug{math/struct/end}{mathml-AF}
        \AddToHook{enddocument/info}
         {
           \iow_term:n{MathML~statistic}
           \iow_term:n{================}
           \iow_term:e{==>~\int_use:N\g__math_mathml_total_int\c_space_tl
           MathML~fragments~read}
           \iow_term:e{==>~\int_use:N\g__math_mathml_int\c_space_tl
           different~MathML~fragments}
           \iow_term:e{==>~\int_use:N\g__math_math_total_int\c_space_tl
           math~fragments~found}
           \iow_term:e{==>~\int_use:N\g__math_mathml_AF_found_int\c_space_tl
           fitting~MathML~AF~found}
           \iow_term:e{==>~\int_use:N\g__math_mathml_AF_attached_int\c_space_tl
           MathML~AF~attached}
         }
      }
 }
\AddToHook{begindocument}{\__math_AF_process_mathml_files:}
```

(*End of definition for* `\__math_AF_process_mathml_files:`.)

7.6 Mathstyle detection

In some cases we need to detect the mathstyle used in a `\mathchoice` command and to disable/enable tagging in the unused branches. This is currently only used in the amstext command `\text` but is perhaps also needed in other cases, so we create a general command.

`\l__math_mathstyle_int`
`\g__math_mathchoice_int`
`mathstyle`

```
\int_new:N \l__math_mathstyle_int
\int_new:N \g__math_mathchoice_int
\property_new:nnnn{mathstyle}{now}{-1}{\int_use:N \l__math_mathstyle_int }
```

(*End of definition for* `\l__math_mathstyle_int`, `\g__math_mathchoice_int`, *and* `mathstyle`.)

For now internal, but perhaps will need a public version. The command should be used in every branch of a `\mathchoice` (with the correct mathstyle number) and with an unique label (which should be the same in every branch). `\g__math_mathchoice_int` can be, e.g., increased before the mathchoice and then used.

`\__math_tag_if_mathstyle:nn`

```
\cs_new_protected:Npn \__math_tag_if_mathstyle:nn #1 #2
 %#1 refers to label
 %#2 is a number for the mathstyle (typically 0,2,4,6)
 {
   \int_set:Nn \l__math_mathstyle_int {#2}
   \property_record:nn {#1} { mathstyle }
   \int_compare:nNnTF { \property_ref:nn {#1}{ mathstyle} } = { #2 }
    { \tag_resume:n{\mathchoice} }{ \tag_suspend:n{\mathchoice} }
 }
\cs_generate_variant:Nn \__math_tag_if_mathstyle:nn {en}
```

(*End of definition for* `\__math_tag_if_mathstyle:nn`.)

7.7 Tagging options

```
\keys_define:nn { __tag / setup }
  {
   math/mathml/sources .clist_set:N = \l__tag_math_mathml_files_clist,
   math/alt/use        .bool_set:N  = \l__tag_math_alt_bool,
   viewer/pane/mathml      .bool_set:N = \l__tag_math_mathml_pane_bool,
   viewer/pane/mathml      .initial:n = true,
   viewer/pane/mathsource  .bool_set:N = \l__tag_math_texsource_pane_bool,
   math/mathml/AF .bool_set:N = \l__tag_math_mathml_AF_bool,
   math/mathml/AF  .initial:n = true,
   math/tex/AF    .bool_set:N = \l__tag_math_texsource_AF_bool,
   math/tex/AF    .initial:n = true
  }
```

alt is required for pdf/UA-1. TODO: l3pdfmeta should support this test.

```
\AddToHook{begindocument/end}
```

```
{
  \str_if_eq:eeT
   {1}
   {
       \exp_last_unbraced:Ne\use_i:nn
        {\GetDocumentProperties{document/pdfstandard-UA}}
        \c_empty_tl\c_empty_tl
   }
   {
     \bool_if:NF \l__tag_math_alt_bool
      {
        \typeout{PDF/UA-1~detected.~Enabling~alt~text~on~Formula}
      }
     \bool_set_true:N\l__tag_math_alt_bool
   }
}
```

7.7.1 Meta keys

The `math/setup` key accepts a list with the values `mathml-SE`, `mathml-AF` and `tex-AF`. It is a fast way to set the main option. It at first disables them all, to get a clean state.

```
\keys_define:nn {__tag / setup}
 {
   math/setup .code:n =
    {
      %deactivate loading of luamml
      \tl_gset:Nn \g__math_luamml_load_tl{-1}
      \keys_set:nn {__tag / setup}
       {
         %deactivate tex source AF
         math/tex/AF = false,
         %deactivate reading of mathml-AF
         math/mathml/sources=,
         math/mathml/AF=false,
         %deactivate structelem
         math/mathml/structelem=false,
         %handle value
       }
      \clist_map_inline:nn { #1}
       {
         \keys_set:nn {__tag/ setup}{math/__setup/##1}
       }
    },
   math/__setup / mathml-SE .code:n =
    {
     \tl_gset:Nn \g__math_luamml_load_tl{1}
     \keys_set:nn {__tag / setup}
       {
         math/mathml/structelem=true
       }
    },
   math/__setup / mathml-AF .code:n =
    {
     \tl_gset:Nn \g__math_luamml_load_tl{1}
```

```
      \clist_put_right:Ne\l__tag_math_mathml_files_clist
       {\c_sys_jobname_str-mathml,\c_sys_jobname_str-luamml-mathml}
      \keys_set:nn {__tag / setup}
        {
          math/mathml/AF=true
        }
     },
    math/__setup / tex-AF .code:n =
     {
      \keys_set:nn {__tag / setup}
        {
          math/tex/AF =true
        }
     },
  }
```

7.8 Sockets

7.8.1 Main inline math sockets

`math/inline/begin` (*tag socket*)
`math/inline/end` (*tag socket*)
`nline/formula/begin` (*tag socket*)
`/inline/formula/end` (*tag socket*)

These sockets are already declared in lttagging and only documented here. The first two sockets are meant to embed inline math into the surrounding (so to close/reopen, e.g., MC-chunks). The other two implement the actual formula structure. The formula sockets are despite their naming not symmetric: the begin socket is issued after the math has started, while the end socket is after the math!

```
%\NewTaggingSocket{math/inline/begin}{0}
%\NewTaggingSocket{math/inline/end}{0}
%\NewTaggingSocket{math/inline/formula/begin}{2} %
%\NewTaggingSocket{math/inline/formula/end}{0}
```

`MC` (*plug*)

```
\NewTaggingSocketPlug
  {math/inline/begin}
  {MC}
  {\tag_mc_end_push:}
\NewTaggingSocketPlug
  {math/inline/end}
  {MC}
  {\tag_mc_begin_pop:n{}}
```

We probably will want to test different tagging recipes.

`default` (*plug*)

```
\NewTaggingSocketPlug
  {math/inline/formula/begin}
  {default}


  { \tagpdfparaOff
    \__math_luamml_structelem:
    \tag_socket_use:n{math/content}
```

```
    \tag_socket_use:n{math/struct/begin}
    #2
    \tag_socket_use:n{math/end}
  }
\NewTaggingSocketPlug
  {math/inline/formula/end}
  {default}
  {
    \tag_socket_use:n{math/struct/end}
  }
```

7.8.2 Main display math sockets

`math/display/begin` (*tag socket*)
`math/display/end` (*tag socket*)
`splay/formula/begin` (*tag socket*)
`display/formula/end` (*tag socket*)

These sockets are already declared in lttagging and only documented here. The first two sockets are meant to embed display math into the surrounding (so to close/reopen, e.g., MC-chunks and P-structure). The other two implement the actual formula structure. The formula sockets are despite their naming not symmetric: the begin socket is issued after the math has started, while the end socket is after the math!

```
%\NewTaggingSocket{math/display/begin}{0}
%\NewTaggingSocket{math/display/end}{0}
%\NewTaggingSocket{math/display/formula/begin}{2} %
%\NewTaggingSocket{math/display/formula/end}{0}
```

`default` (*plug*)

```
\NewTaggingSocketPlug
  {math/display/begin}
  {default}
  { \__tag_tool_close_P:  }
\NewTaggingSocketPlug
  {math/display/end}
  {default}
  {
  }
```

`default` (*plug*)

```
\NewTaggingSocketPlug
  {math/display/formula/begin}
  {default}
  {
    \tagpdfparaOff
    \__math_luamml_structelem:
    \tag_socket_use:n{math/content}
    \tag_socket_use:n{math/struct/begin}
    #2
    \tag_socket_use:n{math/end}
  }
\NewTaggingSocketPlug
  {math/display/formula/end}
  {default}
  {
    \tag_socket_use:n{math/struct/end}
  }
```

7.8.3 Sockets plugs for tags (labels)

`h/display/tag/begin` (*tag socket*)
`ath/display/tag/end` (*tag socket*)

These sockets are already declared in lttagging and only documented here. These sockets are used in `\maketag__math@` to tag labels as Lbl. luamml changes the plug to move the Lbl into the math structure with an intent.

```
%\NewTaggingSocket{math/display/tag/begin}{0}
%\NewTaggingSocket{math/display/tag/end}{0}
```

`default` (*plug*)

```
\NewTaggingSocketPlug
  {math/display/tag/begin}
  {default}
  {
    \tag_mc_end:
    \tag_struct_begin:n {tag=Lbl}
    \tag_mc_begin:n {}
  }
\NewTaggingSocketPlug
  {math/display/tag/end}
  {default}
  {
    \tag_mc_end:
    \tag_struct_end:
    \tag_mc_begin:n{}
  }
\AssignTaggingSocketPlug{math/display/tag/begin}{default}
\AssignTaggingSocketPlug{math/display/tag/end}{default}
```

7.8.4 Internal sockets

`\l__math_content_template_tl`

The default text used as alt or actual text.

```
\tl_new:N\l__math_content_template_tl
\tl_set:Nn \l__math_content_template_tl
   {
       LaTeX~ formula~ starts~
       \exp_not:N\begin{\g__math_grabbed_env_tl}
       \c_space_tl
       \exp_not:V\g__math_grabbed_math_tl
       \c_space_tl
       \exp_not:N\end{\g__math_grabbed_env_tl}
       \c_space_tl LaTeX~ formula~ ends~
   }
```

\l__math_texsource_template_tl

The default text used as texsource

```
\tl_new:N\l__math_texsource_template_tl
\tl_const:Nn\c__math_inline_env_tl {math}
\tl_set:Nn \l__math_texsource_template_tl
   {
     \tl_if_eq:NNTF\g__math_grabbed_env_tl\c__math_inline_env_tl
      {
       $
         \exp_not:V\g__math_grabbed_math_tl
       $
      }
      {
       \exp_not:N\begin{\g__math_grabbed_env_tl}
       \exp_not:V\g__math_grabbed_math_tl
       \exp_not:N\end{\g__math_grabbed_env_tl}
      }
   }
```

math/content (*tag socket*) The math content is stored in associated files and used for actual and alternative text. As the exact text is still unclear we use a socket to be able to test variants. The socket should set all four tl vars above, if needed to identical values. It can use the two variables \g__math_grabbed_env_tl and \g__math_grabbed_math_tl

```
\NewTaggingSocket{math/content}{0}
```

Some default sockets to set the contents. TODO: think about naming convention. TODO: think how this should organized so that one has options to change from the outside and so that there are less repetitions.

actual+source (*plug*)

```
\NewTaggingSocketPlug
  {math/content}
  {actual+source}
  {
   \tl_set:Ne\l__math_content_actual_tl
    {
       \l__math_content_template_tl
    }
   \tl_set:Ne \l__math_content_AF_source_tl
    {
      \l__math_texsource_template_tl
    }
   \tl_set:Nn    \l__math_content_AF_mathml_tl {}
   \tl_set:Nn    \l__math_content_alt_tl    {}
  }
```

alt+source (*plug*)

```
\NewTaggingSocketPlug
```

```
  {math/content}
  {alt+source}
  {
   \tl_set:Ne\l__math_content_alt_tl
    {
       \l__math_content_template_tl
    }
   \tl_set:Ne \l__math_content_AF_source_tl
    {
       \l__math_texsource_template_tl
    }
   \tl_set:Nn    \l__math_content_AF_mathml_tl {}
   \tl_set:Nn    \l__math_content_actual_tl    {}
  }

\AssignTaggingSocketPlug{math/content}{alt+source}
```

`math/struct/begin` (*tag socket*)
`math/struct/end` (*tag socket*)

For the main structure we use a socket too. This allows, e.g., to create a special one for luamml which setups additional objects. The begin socket can use the two variables `\g__math_grabbed_env_tl` and `\g__math_grabbed_math_tl`

```
\NewTaggingSocket{math/struct/begin}{0}
\NewTaggingSocket{math/struct/end}{0}
```

`default` (*plug*)

TODO: think about some naming convention ...

```
\NewTaggingSocketPlug
  {math/struct/begin}
  {default}
  {
    \bool_if:NTF\l__tag_math_texsource_AF_bool
     { \tl_set_eq:NN \l__math_content_AF_source_tmpa_tl \l__math_content_AF_source_tl }
     { \tl_clear:N \l__math_content_AF_source_tmpa_tl }
    \tl_if_eq:NnTF\g__math_grabbed_env_tl {math}
          {
            \tl_set:Nn\l__math_attribute_class_tl{inline}
          }
          {
            \tl_set:Nn\l__math_attribute_class_tl{display}
          }
    \bool_if:NF\l__tag_math_alt_bool
      { \tl_set:Nn \l__math_content_alt_tl{} }
    \tag_struct_begin:n
     {
       tag=Formula,
       attribute-class=\l__math_attribute_class_tl,
       texsource   = \l__math_content_AF_source_tmpa_tl,
       title-o     = \g__math_grabbed_env_tl,
       actualtext  = \l__math_content_actual_tl,
       alt         = \l__math_content_alt_tl
     }
    \typeout{====>grabbed~math=\meaning\g__math_grabbed_math_tl}
    \tag_mc_begin:n{}
  }
```

```
\NewTaggingSocketPlug
  {math/struct/end}
  {default}
  { \tag_mc_end: \tag_struct_end: }

\AssignTaggingSocketPlug{math/struct/begin}{default}
\AssignTaggingSocketPlug{math/struct/end}{default}
```

`mathml-AF` (*plug*) This socket tries to add a mathml-AF to formula. It is activated if a mathml.html has been found and loaded. As it disturbs the reading of the AF it currently deactivates the /Alt key, unless it has been reenabled with `math/alt/use=true`

```
\cs_generate_variant:Nn \str_mdfive_hash:n {o}
\tl_new:N\l__math_content_hash_tl
```

we need to save the grabbed math:

```
\tl_new:N\l__math_grabbed_math_tl
```

the socket definition

```
\NewTaggingSocketPlug
  {math/struct/begin}
  {mathml-AF}
  {
   \int_gincr:N\g__math_math_total_int
   \tl_set:Ne\l__math_content_hash_tl
    {\str_mdfive_hash:o { \l__math_content_AF_source_tl }}
   \tl_set_eq:NN\l__math_grabbed_math_tl\g__math_grabbed_math_tl
   \tl_if_eq:NnTF\g__math_grabbed_env_tl {math}
     {
       \tl_set:Nn\l__math_attribute_class_tl{inline}
     }
     {
       \tl_set:Nn\l__math_attribute_class_tl{display}
     }
   \bool_if:NF\l__tag_math_alt_bool
     { \tl_set:Nn \l__math_content_alt_tl{} }
```

debugging option. TODO: hide in debug key.

```
   \tl_if_exist:cTF { g__math_mathml_ \l__math_content_hash_tl _tl }
     {
       \int_gincr:N\g__math_mathml_AF_found_int
       \bool_if:NTF \l__tag_math_mathml_AF_bool
        {
          \int_gincr:N\g__math_mathml_AF_attached_int
          \typeout {Inserting~mathml~with~Hash~\l__math_content_hash_tl}
        }
        {
          \typeout {Ignoring~mathml~with~Hash~\l__math_content_hash_tl}
        }
     }
```

```
    {
      \bool_if:NT \l__tag_math_mathml_AF_bool
       {
         \typeout {WARNING:~mathml~missing~for~hash~\l__math_content_hash_tl}
       }
    }
  \tag_socket_use:n {math/mathml/write/prepare}
  \tag_socket_use:n {math/mathml/write} % write hash if request
   \bool_if:NTF\l__tag_math_texsource_AF_bool
    { \tl_set_eq:NN \l__math_content_AF_source_tmpa_tl \l__math_content_AF_source_tl }
    { \tl_clear:N \l__math_content_AF_source_tmpa_tl }
  \tag_struct_begin:n
    {
      tag=Formula,
      attribute-class=\l__math_attribute_class_tl, %
      AFref       =
       \bool_if:NT\l__tag_math_mathml_AF_bool
        {
          \cs_if_exist_use:c {g__math_mathml_ \l__math_content_hash_tl _tl}
        },
      texsource   = \l__math_content_AF_source_tmpa_tl, % should be after mathml AF!
      title-o     = \g__math_grabbed_env_tl,    %
      alt         = \l__math_content_alt_tl
    }
  \typeout{====>grabbed~math=\meaning\g__math_grabbed_math_tl}
  \tag_mc_begin:n{}
  }
```

not really needed but looks more symmetric:

```
\NewTaggingSocketPlug
  {math/struct/end}
  {mathml-AF}
  {
    \tag_mc_end:
    \tag_struct_end:
  }
```

`math/end` (*tag socket*) A socket used at the end of the math (before the closing dollar(s)) which can, e.g., set a flag for luamml.

```
\NewTaggingSocket{math/end}{0}
```

```
\AssignTaggingSocketPlug{math/inline/begin}{MC}
\AssignTaggingSocketPlug{math/inline/end}{MC}
\AssignTaggingSocketPlug{math/inline/formula/begin}{default}
\AssignTaggingSocketPlug{math/inline/formula/end}{default}
\AssignTaggingSocketPlug{math/display/begin}{default}
\AssignTaggingSocketPlug{math/display/end}{default}
\AssignTaggingSocketPlug{math/display/formula/begin}{default}
\AssignTaggingSocketPlug{math/display/formula/end}{default}
```

7.9 Interface commands

__math_process:nn
__math_process:Vn
__math_process_auxi:nn
__math_process_auxii:nn

A no-op place-holder; the internal wrapper means that it does not need to be concerned with internals.

```
\cs_new_protected:Npn \__math_process:nn #1#2
  {
    \legacy_if:nF { measuring@ }
      {
        \tl_if_in:nnTF {#2} { \m@th }
          { \bool_set_true:N\l__math_fakemath_bool }
          { \tl_trim_spaces_apply:nN {#2} \__math_process_auxi:nn {#1} }
      }
  }
\cs_generate_variant:Nn \__math_process:nn { V }
\cs_new_protected:Npn \__math_process_auxi:nn #1#2
  {
    \tl_gset:Nn \g__math_grabbed_env_tl {#2}
    \tl_gset:Nn \g__math_grabbed_math_tl {#1}
    \__math_process_auxii:nn {#2} {#1}
  }
\cs_new_protected:Npn \__math_process_auxii:nn #1#2 { }
```

(*End of definition for* `\__math_process:nn`, `\__math_process_auxi:nn`, *and* `\__math_process_auxii:nn`.)

\math_processor:n

A simple installer

```
\cs_new_protected:Npn \math_processor:n #1
  { \cs_set_protected:Npn \__math_process_auxii:nn ##1##2 {#1} }
```

(*End of definition for* `\math_processor:n`. *This function is documented on page 5.*)

7.10 Content grabbing

\MathCollectTrue
\MathCollectFalse

```
\cs_set_protected:Npn\MathCollectTrue{\bool_set_false:N \l__math_collected_bool}
\cs_set_protected:Npn\MathCollectFalse{\bool_set_true:N \l__math_collected_bool}
```

(*End of definition for* `\MathCollectTrue` *and* `\MathCollectFalse`. *These functions are documented on page 5.*)

__math_grab_dollar:w
__math_grab:n

Top-level function to handle grabbing of inline math mode delimited by `$` tokens. We provide two different ways to do that: a token-by-token one that can be used everywhere, and a fast delimited one that does not work anywhere that the end `$` token may be hidden, most obviously in tabulars. The function here is therefore set up as a variable starting point.

```
\cs_new_protected:Npn \__math_grab_dollar:w { \__math_grab_dollar_delim:w }
```

After grabbing inline math material, there is again common processing independent of mechanism of collection.

```
\cs_new_protected:Npn \__math_grab:n #1
  {
```

We need to do processing first as this picks up "fake" math mode: that information is needed below.

```
    \__math_process:nn { math } {#1}
```

We do not want math tagging in fakemath or when measuring, We also do not want math tagging if tagging has been suspended.

```
      \bool_lazy_any:nTF
        {
          {\legacy_if_p:n { measuring@ }}
          { \l__math_fakemath_bool }
          { \tl_if_blank_p:n {#1} }
        }
        {
          \__math_luamml_ignore:
          #1 $ % $
        }
        {
            \tag_socket_use:n  {math/inline/begin} %end P-MC
```

We do no use a tagging socket here, so that the argument (the math) is not lost, tagging-project issue 661.

```
            \tag_socket_use:nnn {math/inline/formula/begin}{}{#1}
            $ % $
            \tag_socket_use:n  {math/inline/formula/end}
            \tag_socket_use:n  {math/inline/end} % restart P-MC
        }
  }
```

(*End of definition for* `\__math_grab_dollar:w` *and* `\__math_grab:n`.)

`\__math_grab_dollar_delim:w` Grab up to a single `$`, for inline math mode, suppressing any processing if the token is `\m@th` found in the content.

```
\cs_new_protected:Npn \__math_grab_dollar_delim:w #1 $ % $
  { \__math_grab:n {#1} }
```

(*End of definition for* `\__math_grab_dollar_delim:w`.)

`\__math_grab_dollardollar:w` And for the classical TEX display structure.

```
\cs_new_protected:Npn \__math_grab_dollardollar:w #1 $$ {
  \tl_if_blank:nF {#1}
    {
      \__math_process:nn { equation* } {#1}
      \tag_socket_use:n {math/display/begin}
      \tag_socket_use:nn{math/display/formula/begin}{}{#1}
    }
```

Prepare to finish the display math formula and let TEX do its job; then regain control after the formula has been contributed to the page, in order to add the tagging followed by the correct penalty and skip.

```
  \__math_prepare_display_end:
  $$
}
```

(*End of definition for* `\__math_grab_dollardollar:w`.)

`\__math_grab_inline_delim:w` Collect inline math content and deal with the need to move to math mode.

```
\cs_new_protected:Npn \__math_grab_inline_delim:w % \(
  #1 \)
  {
    \tl_if_blank:nF {#1}
      {
        $ #1 $
      }
    \bool_set_false:N \l__math_collected_bool
  }
```

(*End of definition for* `\__math_grab_inline_delim:w`.)

`\__math_grab_inline:w` See `\@@_grab_dollar:w`.

```
\cs_new_protected:Npn \__math_grab_inline:w { \__math_grab_inline_delim:w }
```

(*End of definition for* `\__math_grab_inline:w`.)

`\__math_grab_eqn:w` For the most common use of `\[/\]`: turn into an environment.

```
\cs_new_protected:Npn \__math_grab_eqn:w % \[
  #1 \]
   {
%     \typeout{collected? = \bool_if:NTF \l__math_collected_bool {true}{false}}
     \begin { equation* } #1 \end { equation* }
   }
```

(*End of definition for* `\__math_grab_eqn:w`.)

7.11 Token-by-token inline grabbing

Grabbing inline math token-by-token is more involved. The mechanism here is essentially a simplified version of that originally seen in collcell and refined in siunitx. We make use of the fact that in math mode spaces are ignored, so we have to deal with only `N`-type tokens and groups. Furthermore, there is no need to look inside groups, so the only special cases are a small selection of `N`-type tokens.

`\l__math_grabbed_tl` For collection of the material piecewise.

```
\tl_new:N \l__math_grabbed_tl
```

`\l__math_grab_env_int` Needed to count up the number of nested environments encountered.

```
\int_new:N \l__math_grab_env_int
```

__math_grab_loop_aux:
__math_grab_loop:

The lead-off here establishes a group: we need that as we will have to be careful in the way `\cr` is handled and ensure this is only manipulated whilst grabbing. The main loop is then started.

```
\cs_new_protected:Npn \__math_grab_loop_aux:
  {
    \group_begin:
      \tl_clear:N \l__math_grabbed_tl
      \__math_grab_loop:
  }
\cs_new_protected:Npn \__math_grab_loop:
  {
    \peek_remove_spaces:n
      {
        \peek_meaning:NTF \c_group_begin_token
          { \__math_grab_loop_group:n }
          { \__math_grab_loop_token:N }
      }
  }
```

(*End of definition for* `\__math_grab_loop_aux:` *and* `\__math_grab_loop:`.)

__math_grab_loop_group:n
__math_grab_loop_store:n

Handling of grabbed groups is pretty easy.

```
\cs_new_protected:Npn \__math_grab_loop_group:n #1
  { \__math_grab_loop_store:n { {#1} } }
\cs_new_protected:Npn \__math_grab_loop_store:n #1
  {
    \tl_put_right:Nn \l__math_grabbed_tl {#1}
    \__math_grab_loop:
  }
```

(*End of definition for* `\__math_grab_loop_group:n` *and* `\__math_grab_loop_store:n`.)

__math_grab_loop_token:N
__math_grab_loop_$:
__math_grab_loop_\\:
__math_grab_loop_\begin:
__math_grab_loop_\end:
__math_grab_loop_\ignorespaces:
__math_grab_loop_\unskip:
__math_grab_loop_\textonly@unskip:

Filter out the special cases: for performance reasons, use a hash table approach rather than a loop (*cf.* collcell).

```
\cs_new_protected:Npn \__math_grab_loop_token:N #1
  {
    \cs_if_exist_use:cF
      { __math_grab_loop_ \token_to_str:N #1 : }
      { \__math_grab_loop_store:n {#1} }
  }
\cs_new_protected:cpn { __math_grab_loop_ \token_to_str:N $ : }
  { \__math_grab_loop_end: }
\cs_new_protected:cpn { __math_grab_loop_ \token_to_str:N \) : }
  { \__math_grab_loop_end: }
\cs_new_protected:cpn { __math_grab_loop_ \token_to_str:N \\ : }
  {
    \int_compare:nNnTF \l__math_grab_env_int = 0
       { \__math_grab_loop_newline: }
       { \__math_grab_loop_store:n { \\ } }
  }
```

In contrast to collcell, nesting is tracked by counting \begin/\end pairs: this is needed in case there is a tabular-like construct containing \\ inside a cell. As a result, the end-of-tabular can be detected without checking the name argument: if \end is encountered at nesting level 0, we've hit the end of a cell. In that case, end the row and leave the environment to clean up.

```
\cs_new_protected:cpn { __math_grab_loop_ \token_to_str:N \begin : }
  {
    \int_incr:N \l__math_grab_env_int
    \__math_grab_loop_store:n { \begin }
  }
\cs_new_protected:cpn { __math_grab_loop_ \token_to_str:N \end : }
  {
    \int_compare:nNnTF \l__math_grab_env_int = 0
       {
         \__math_grab_loop_newline:
         \end
       }
       {
         \int_decr:N \l__math_grab_env_int
         \__math_grab_loop_store:n { \end }
       }
  }
\tl_map_inline:nn { \ignorespaces \unskip \textonly@unskip }
  {
    \cs_new_protected:cpn { __math_grab_loop_ \token_to_str:N #1 : }
      { \__math_grab_loop: }
  }
```

(*End of definition for* __math_grab_loop_token:N *and others.*)

__math_grab_loop_newline: To allow collection of tokens in the part of the \halign template after #, we need TEX to see the primitive with the loop token in the right place. That is done by re-defining \cr at present. Ideally there would be a socket in the definition of tabular, etc., to handle this: there is also the need to examine in interaction with longtable, which also redefines \cr.

```
\cs_new_protected:Npn \__math_grab_loop_newline:
  {
    \if_false: { \fi:
    \cs_set_protected:Npn \cr
      {
        \__math_grab_loop:
        \tex_cr:D
      }
    \if_false: } \fi:
    \\
  }
```

(*End of definition for* __math_grab_loop_newline:.)

__math_grab_loop_end: Clean up and pass on.

```
\cs_new_protected:Npn \__math_grab_loop_end:
```

```
  {
    \exp_args:NNV \group_end:
    \__math_grab:n \l__math_grabbed_tl
  }
```

(*End of definition for* `\__math_grab_loop_end:`.)

7.12 Marking math environments

A general mechanism for math mode environments that do not grab their content (*cf.* most amsmath environments).

`\l__math_env_name_tl` To allow us to carry out "special effects"

```
\tl_new:N \l__math_env_name_tl
```

Here we set up specialised handling of environments. The idea for the `arg-spec` key is that if an environment takes arguments, we don't worry during the main grabbing. Rather, we remove the arguments from the grabbed content and forward only the payload. That is done by (ab)using ltcmd.

```
\keys_define:nn { __math }
  {
     arg-spec .code:n =
       {
         \ExpandArgs { c } \DeclareDocumentCommand
           { __math_env \l__math_env_name_tl _aux: }
           {#1}
           { \__math_env_forward:w }
       }
  }
```

`\math_register_env:nn` `\math_register_env:n` `\RegisterMathEnvironment` Set up to capture environment content and make available.

```
\cs_new_protected:Npn \math_register_env:nn #1#2
  {
    \tl_set:Nn \l__math_env_name_tl {#1}
    \keys_set:nn { __math } {#2}
    \cs_gset_eq:cc { __math_env_ #1 _begin: } {#1}
    \cs_gset_eq:cc { __math_env_ #1 _end: } { end #1 }
%
    \ExpandArgs { nne } \RenewDocumentEnvironment {#1} { b }
      {
        \exp_not:N \bool_if:NTF \exp_not:N \l__math_collected_bool
          {
%            \typeout{===>B1}
          }
          {
%            \typeout{===>B2}
            \cs_if_exist:cTF { __math_env #1 _aux: }
              {
                \exp_not:c { __math_env #1 _aux: }
                  ##1 \exp_not:N \__math_env_end: {#1}
```

```
              }
              { \exp_not:N \__math_process:nn {#1} {##1} }
            \exp_not:n { \@kernel@math@registered@begin }
            \bool_set_true:N \exp_not:N \l__math_collected_bool
          }
%        \exp_not:N \tracingall
        \exp_not:c { __math_env_ #1 _begin: }
        ##1
        \exp_not:c { __math_env_ #1 _end: }
%        \exp_not:N \tracingnone
    }
    {
    }
  }

\cs_new_protected:Npn \math_register_halign_env:nn #1#2
  {
    \tl_set:Nn \l__math_env_name_tl {#1}
    \keys_set:nn { __math } {#2}
    \cs_gset_eq:cc { __math_env_ #1 _begin: } {#1}
    \cs_gset_eq:cc { __math_env_ #1 _end: } { end #1 }
%
    \ExpandArgs { nnee } \RenewDocumentEnvironment {#1} { b }
      {
        \exp_not:N \bool_if:NTF \exp_not:N \l__math_collected_bool
          {
%            \typeout{===>B1}
          }
          {
%            \typeout{===>B2}
            \cs_if_exist:cTF { __math_env #1 _aux: }
              {
                \exp_not:c { __math_env #1 _aux: }
                  ##1 \exp_not:N \__math_env_end: {#1}
              }
              { \exp_not:N \__math_process:nn {#1} {##1} }
            \exp_not:n { \@kernel@math@registered@begin }
            \bool_set_true:N \exp_not:N \l__math_collected_bool
          }
%        \exp_not:N \tracingall
        \exp_not:c { __math_env_ #1 _begin: }
        ##1
%        \exp_not:N \tracingnone
    }
    {
      \exp_not:c { __math_env_ #1 _end: }
    }
  }


\cs_new:Npn \@kernel@math@registered@begin {
%  \ShowTagging{struct-stack}
%\typeout{==>A1}\ShowTagging{struct-stack,mc-current}
```

```
  \mode_if_vertical:TF
      {
%         \legacy_if:nTF { @endpe }
%           { \legacy_if_set_false:n { @endpe } }
%           { \__block_list_beginpar_vmode: }
%
%         \typeout{==>~ at:~ \g__tag_struct_tag_tl}
%
        \tag_if_active:T
          {
            \exp_args:Noo\str_if_eq:nnF \g__tag_struct_tag_tl { \l__tag_para_main_tag_tl }    %
             {
%               \typeout{==>A2}
              \__block_beginpar_vmode:
             }              % needs correction!
          }
      }
      {
%         \typeout{==>A3}
        \__tag_tool_close_P:
      }
  \tag_socket_use:nn{math/display/formula/begin}{}{}
%  \typeout{==>MC1}\ShowTagging{mc-current}
}

\cs_new_protected:Npn \math_register_env:n #1
  { \math_register_env:nn {#1} { } }

\NewDocumentCommand \RegisterMathEnvironment { O{} m }
  { \math_register_env:nn {#2} {#1} }
```

(*End of definition for* `\math_register_env:nn`, `\math_register_env:n`, *and* `\RegisterMathEnvironment`. *These functions are documented on page* 5.)

`\__math_env_forward:w`

```
\cs_new_protected:Npn \__math_env_forward:w #1 \__math_env_end: #2
  { \__math_process:nn {#2} {#1} }
```

(*End of definition for* `\__math_env_forward:w`.)

7.13 Regaining control after a display has finished with $$

The tagging structures have to be added after the formula content but before TEX is allowed to break a page. But TEX will automatically add `\postdisplaypenalty` followed by `\belowdisplayskip` or `\belowdisplayshortskip` without giving us any chance to take control before these are added.

We therefore implement the following approach:

- Just before we end the formula we save away the current value of `\postdisplaypenalty` in case it was altered within the formula. Then we set it to 10000 so that what is inserted by TEX is not allowing for a page break at this point

- At this point We also normally flip the sign of `\belowdisplayskip` and `\belowdisplayshortskip` so that they become negative and record that we have done this, by setting the token list `\g__math_skip_sign_tl` to `-`.
- However, we only do that if both are non-negative. If either of them is negative we assume that this was deliberately done by the user to adjust the spacing around this particular display. Again, we record in `\g__math_skip_sign_tl` that we haven't done the sign flip by setting the variable to empty.
- Making these two skips negative is essential, because the formula will be added using the special `\postdisplaypenalty` value that doesn't allow a page break even though the real value (that we use later on) will probably do that. Thus the below skip added by TEX will add to the size of the display and not vanish into the bottom margin and if it is positive TEX might decide to move the whole formula onto the next page even though it might well fit.
- Once TEX has finished processing the closing `$$` it has added something like

```
\penalty 100000        % our special value for \postdisplaypenalty
\glue -10.0 plus -3pt  % the \belowdisplayskip (with sign flipped
```

after the formula. This means that at this that point we can retrieve this skip by using `\lastskip` and we just have have to flip the sign again to know how to correct it (no need to know whether the normal or the short skip was added by TEX) and the right penalty is available to us too as we have saved it away in `\g__math_postdisplaypenalty_int`.

`\__math_prepare_display_end:`

Prepare to finish the formula and give control to TEX. This is normally used directly in front of `$$` (`\eqno` or `\leqno`.

```
\cs_new_protected:Npn \__math_prepare_display_end: {
  \bool_lazy_or:nnTF
    { \dim_compare_p:nNn \belowdisplayskip  < {0pt} }
    { \dim_compare_p:nNn \belowdisplayshortskip  < {0pt} }
```

No sign flipping if one or both of the skips are already negative.

```
    { \tl_gclear:N \g__math_skip_sign_tl  }
```

Otherwise flip the sign of both.

```
    {
      \tl_gset:Nn \g__math_skip_sign_tl {-}
      \skip_set:Nn \belowdisplayskip      {-\belowdisplayskip}
      \skip_set:Nn \belowdisplayshortskip {-\belowdisplayshortskip}
    }
```

For the skip it is enough to flip the sign, because we get the value back through `\lastskip` but for the `\postdisplaypenalty` change we have to save the current value somewhere, because it may have been changed within the display formula. This has to be done globally, because it is used after the formula has ended.

```
  \int_gset_eq:NN \g__math_postdisplaypenalty_int \postdisplaypenalty
  \int_set_eq:NN  \postdisplaypenalty \@M
}
```

(*End of definition for* `\__math_prepare_display_end:`.)

`\__math_tag_dollardollar_display_end:`

The code to be executed after the formula has ended is injected using `\group_insert_after:N` inside of `\tex_everydisplay:D` (i.e., when the formula starts but inside the group started by the display). As `\group_insert_after:N` expects a single token we have to store that code in a command.

```
\cs_new_protected:Npn \__math_tag_dollardollar_display_end:
  {
    %  \typeout{== tag dollarldollar display end}
    %  \ShowTagging{struct-stack}
    \para_raw_end:
```

The `\postdisplaypenalty` was temporarily set to 10000 inside the display and both the `\belowdisplayskip` and the `\belowdisplayshortskip` were negated (with some exceptions, see above), so whatever was inserted it should have been a negative or possibly zero skip. Whatever was added, we pick up the value, so that we can correct the spacing after the tagging code has been inserted.

```
    \l__math_tmpa_skip \lastskip
    \tag_socket_use:n{math/display/formula/end}
```

Now we add a skip without introducing a page break possibility, that should bring the current vertical position back to the point where TEX has added the penalty and the "below skip".

```
    \nobreak
    \skip_vertical:n { -\l__math_tmpa_skip  }
```

Then we finally add the real stuff: the true `\postdisplaypenalty` (saved in `\g__math_postdisplaypenalty_int` earlier) and the negated value of the skip value we saved in `\l__math_tmpa_skip`. It may look strange that we have two identical negated skips next to each other, but if you think about it, that is correct: the first cancels the "below skip" that TEX has added and the second puts the same amount of skip after the penalty (which is where it should be).

```
    \penalty \g__math_postdisplaypenalty_int
    \skip_vertical:n
```

Actually we do use the skip without flipping the sign when our records show that it wasn't flipped earlier i.e., when the negative value was due to the user specifying it inside the formula.

```
      { \g__math_skip_sign_tl \l__math_tmpa_skip  }
```

As we are now in vertical mode the situation is different from the way TEX would handle things after a display: TEX would internally switch to horizontal mode without adding a \parskip. But this is not possible to do on the macro level. Therefore we have to neutralize the upcoming \parskip since we can't prevent it from being added.

Actually, we could combine this correction with the previous \skip_vertical:n, which would produce marginally smaller PDFs; but that will make \showoutput tracing somewhat less easy to understand, so I haven't done that (yet).

```
%    \typeout{------->~ add~ negative~ parskip~ (to~ cancel~ the~
%                       one~ that~ TeX~ will~ add)}
    \skip_vertical:n { -\tex_parskip:D }
```

We also set the @domathendpetrue flag to signal that paragraph continuation should happen after such a display.

```
    \@domathendpetrue
    \@doendpe             % this has no \end{...} to take care of it
}
```

(*End of definition for* __math_tag_dollardollar_display_end:.)

\g__math_postdisplaypenalty_int This is the internal variable in which we save the \postdisplaypenalty. It is global because we use it immediately after the formula has ended.

```
\int_new:N \g__math_postdisplaypenalty_int
```

(*End of definition for* \g__math_postdisplaypenalty_int.)

\g__math_skip_sign_tl We record whether we have flipped the signs of \belowdisplayskip, etc., so that we know what to do after the formula has ended. If we have it flipped the signs then variable holds -, whilst otherwise it is empty. This means we can flip the signs again by simply prefixing the value with this token list variable.

```
\tl_new:N \g__math_skip_sign_tl
```

(*End of definition for* \g__math_skip_sign_tl.)

\dollardollar@end When we do tagging we augment the kernel definition for \dollardollar@end in order to regain control at the very end of the formula (after the user may have altered \postdisplaypenalty or \belowdisplayskip).

```
\def \dollardollar@end { \__math_prepare_display_end: $$ }
```

(*End of definition for* \dollardollar@end.)

\eqno \leqno However, in case of a formula using the primitives \eqno or \leqno the \dollardollar@end comes too late as it is executed in the context of the equation number; and the values for \postdisplaypenalty, etc. set at this point are not the ones used for the formula. We therefore have to execute __math_prepare_display_end: just in front of the primitive.

```
\protected\def\eqno{
  \__math_prepare_display_end:
```

Inside the equation we set it temporarly to do nothing, because otherwise it would be executed again when `\dollardollar@end` is reached (not that this would matter, but it would just take unnecessary extra time).

```
  \@kernel@eqno
  \cs_set_eq:NN \__math_prepare_display_end: \prg_do_nothing:
  \aftergroup\ignorespaces
}
```

Same game for `\leqno`.

```
\protected\def\leqno{
  \__math_prepare_display_end:
  \@kernel@leqno
  \cs_set_eq:NN \__math_prepare_display_end: \prg_do_nothing:
  \aftergroup\ignorespaces
}
```

(*End of definition for* `\eqno` *and* `\leqno`.)

7.14 Document commands

Add one more here: `displaymath`, which is equivalent to `\[` , `\]`
and hence to the basic `equation*`.
Added in more recent branch.

`\equation`
`\__math_equation_begin:`
`\equation*`
`\__math_equation_star_begin:`
`\endequation`
`\__math_equation_end:`
`\endequation*`
`\__math_equation_star_end:`

These environments are not set up by amsmath to collect their body, so we do that here. This has to be done *after* we can be sure amsmath is loaded.

Note that with amsmath loaded, `equation*` and `equation`
are the two basics: they are used to define the other single-row
display environments, etc.

```
\tl_gput_right:Nn \@kernel@before@begindocument
  {
    \math_register_env:n { equation }
    \math_register_env:n { equation* }
% at the moment register_env can only do display math
%    \math_register_env:n { math }
    \RenewDocumentEnvironment{math} {b}{$#1$}{}
% and this one doesn't work either
%    \math_register_env:n { displaymath }
    \RenewDocumentEnvironment{displaymath} {b}{\[#1\]}{}
  }
```

(*End of definition for* `\equation` *and others.*)

`\(`
`\)`

If math mode has not been collected, we need to do that; otherwise, worry about whether we are in math mode or not. The closing command here can only occur inside a collected math block: otherwise it will be simply used as a delimiter.

```
\cs_gset_protected:Npn \( % \)
  {
    \bool_if:NTF \l__math_collected_bool
      {
        \mode_if_math:TF
          { \@badmath }
          { $ }
      }
      {
        \__math_grab_inline:w
      }
  } % \(
\cs_gset_protected:Npn \)
  {
    \mode_if_math:TF
      { $ }
      { \@badmath }
  }
```

(*End of definition for* `\(` *and* `\)`.)

`\[` `\]` Again, we need to watch for when amsmath is loaded after this code. The flag usage here is to cover the case where `\[/\]` is hidden inside another environment. In this case the grabbing happens on the outer level and should not be repeated.

```
\tl_gput_right:Nn \@kernel@before@begindocument
  {
    \cs_gset_protected:Npn \[ % \]
       {
         \__math_grab_eqn:w
 %       \bool_if:NTF \l__math_collected_bool
%          { \begin { equation* } }
%          { \__math_grab_eqn:w }
      } % \[
    \cs_gset_protected:Npn \]
      {
        \@badmath
%        \bool_if:NTF \l__math_collected_bool
%          { \end{ equation* } }
%          { \@badmath }
      }
  }
```

(*End of definition for* `\[` *and* `\]`.)

why does ensuremath need handling at all?

Indeed! Currently, this is setup to process the math that it has anyways already captured as its argument; thus it is more efficient than leaving the capture to be repeated by the `\everymath`

A bit of nesting fun to make sure we collect only if required.

```
%\cs_gset_protected:Npn \ensuremath #1
%  {
%    \mode_if_math:TF
%      {#1}
%      {
%        \bool_if:NTF \l__math_collected_bool
%          { \@ensuredmath {#1} }
```

```
%           {
%             \bool_set_true:N \l__math_collected_bool
%             \__math_process:nn { math } {#1}
%             \@ensuredmath {#1}
%             \bool_set_false:N \l__math_collected_bool
%           }
%      }
%  }
```

(*End of definition for* `\ensuremath`.)

7.15 `\everymath` and `\everydisplay`

The business end for grabbing inline math and "raw" TEX display. Most display math mode is actually handled elsewhere, as we have macro control.

```
\exp_args:No \tex_everymath:D
  {
    \tex_the:D \tex_everymath:D
    \bool_if:NF \l__math_collected_bool
      {
        \bool_set_true:N \l__math_collected_bool
        \__math_grab_dollar:w
      }
  }

\exp_args:No \tex_everydisplay:D
  {
    \tex_the:D \tex_everydisplay:D
%   \typeout{==>~ in~ everydisplay}
```

We need to attach tagging after the display math has been processed by TEX, and we also need to correct the penalty and spacing that will get added there. This is done by the following code through which we regain control after the display math group ends.

```
    \group_insert_after:N \__math_tag_dollardollar_display_end:
    \bool_if:NF \l__math_collected_bool
      {
        \bool_set_true:N \l__math_collected_bool
        \__math_grab_dollardollar:w
      }
  }
```

7.16 Modifying kernel environments

We need to cover this even though it is, of course, not encouraged. Registration is delayed until begindocument to avoid error with redefinition in, e.g., fleqn.clo.

```
\tl_gput_right:Nn \@kernel@before@begindocument
  {
    \math_register_env:n { eqnarray }
    \math_register_env:n { eqnarray* }
  }
```

Tabulars currently contain a $ that shouldn't trigger math tagging. Also we do need to change the grabbing method to the slow loop-method.

```
\RequirePackage{array}
\tl_if_exist:NT\@kernel@tabular@init
 {
   \tl_put_right:Nn\@kernel@tabular@init
    {
      \cs_set_protected:Npn \__math_grab_dollar:w { \__math_grab_loop_aux: }
      \cs_set_protected:Npn \__math_grab_inline:w { $ }
    }
 }
```

`\__math_m@th:` `\m@th` Handle non-math use of math mode. At present nesting isn't supported as `\m@th` pops up in a few places that *are* math mode!

```
\cs_new_eq:NN \__math_m@th: \m@th
\cs_gset_protected:Npn \m@th
  {
    \bool_set_true:N \l__math_collected_bool
    \__math_m@th:
  }
```

(*End of definition for* `\__math_m@th:` *and* `\m@th`.)

7.17 Modifying kernel commands

`\mathpalette` The `\mathpalette` command is a problem if lualatex is used and math structure elements are created as the math is then processed more than once. We therefore redefine it to use `\mathstyle`.

```
\sys_if_engine_luatex:T
 {
  \def\mathpalette#1#2{%
   \ifcase\mathstyle
    #1\displaystyle{#2}\or
    #1\displaystyle{#2}\or
    #1\textstyle{#2}\or
    #1\textstyle{#2}\or
    #1\scriptstyle{#2}\or
    #1\scriptstyle{#2}\or
    #1\scriptscriptstyle{#2}\or
    #1\scriptscriptstyle{#2}
   \fi}
  }
```

(*End of definition for* `\mathpalette`.)

`\mathsm@sh` Similar to the phantom commands in latex-lab-text, `\mathsm@sh` must be redefined to include luamml sockets.

```
\def\mathsm@sh#1#2{%
  \setbox\z@\hbox{$\m@th#1{#2}
    \UseTaggingSocket{math/luamml/save/nNn}{{mathsmash} #1 {mpadded}}
    $}%
  \UseTaggingSocket{math/luamml/finsm@sh}{}{{}\finsm@sh}}
```

(*End of definition for* `\mathsm@sh`.)

7.18 Disable math grabbing in the begindocument hook

For example amsart uses math to measure text there.

```
\tl_gput_right:Nn\@kernel@before@begindocument
  {
    \bool_set_true:N\l__math_collected_bool
  }
\tl_gput_right:Nn\@kernel@after@begindocument
 {
   \bool_set_false:N\l__math_collected_bool
 }

\ExplSyntaxOff

⟨@@=⟩

⟨/kernel⟩
```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

C

D

E

F

G

N

U

V