The `latex-lab-math` code*

Frank Mittelbach, Joseph Wright, LaTeX Project

v0.6b 2024-08-14

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.
- BUG: `\[...\]` grabs if `\l_@@_collected_bool` is false. If it is true it falls back to `equation*` and then errors because this can't find the end.

2.1 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.

2.2 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.

3 Math tagging

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 subparts (e.g. lines) with Formula sub structures (perhaps with their own set of additional content)
  - surround elements of the equation with mathml structure elements (currently only luatex with luamml)

3.1 Inline math

The embedding code is added through the sockets

- `tagsupport/math/inline/begin`
- `tagsupport/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 sockets

- `tagsupport/math/inline/formula/begin` This socket takes the math as argument and its code should output it for typesetting. Without tagging the socket uses the identity plug. The `default` plug of the socket calls these three internal sockets for the tagging support:
  - `tagsupport/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).
  - `tagsupport/math/struct/begin` This calls `\tag_struct_begin:n`. It should also write the associated files if needed.
  - `tagsupport/math/substruct/begin` this handles subparts. TODO: does it really make sense in inline math to have that??
- `tagsupport/math/inline/formula/end` This socket ends the formula structure(s). The `default` plug calls these internal sockets:
  - `tagsupport/math/substruct/end`
  - `tagsupport/math/struct/end`

3.2 Display math

to be written

3.3 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—practically 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 3.4) 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"` [1]

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).

3.4 Automatic mathml creation with luamml

If lualatex is used, if the package `unicode-math` is used and if the package `luamml` is found[2] then it will automatically generate the file `\jobname-luamml-mathml.html` with mathml representations of all math formulas. The generation of the file can be suppressed with `math/mathml/luamml=false`. The package 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. The file is then used in subsequent compilations and works also with pdflatex.

3.5 Options

4 Known current bugs, etc.

New Section, now with subsections.
As indicated, these lists are probably incomplete.
Some of these have been addressed in a more recent branch.

4.1 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$\end{minipage}$
   ```

[1] But it is probably not needed and only blows up the PDF.
[2] Until it is on CTAN and it must be installed from https://github.com/zauguin/luamml.

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. Defining `\[` to fall back to `equation*` doesn't work if `equation*` is a grabbing environment.
6. The behaviour of `\[...\]` is faulty. See above.

4.2 Fake math

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.

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.

4.3 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.

4.4 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.

4.5 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.

5 The Implementation

1 ⟨`@@=math`⟩

2 ⟨∗kernel⟩

5.1 File declaration

Change description here?

```
\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}}
\ExplSyntaxOn
```

5.2 Setup

Loading amsmath is an absolute requirement: this avoids needing to have conditional definitions and deals with how to define `\[/\]` neatly.

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

5.3 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 tl?

`\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
```

5.4 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},
      }
```

5.5 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_bool
```

```
\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
\bool_new:N\g__tag_math_luamml_bool
\bool_gset_true:N \g__tag_math_luamml_bool
```

\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. If luatex is detected with 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\%
   \__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>
    \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.*)

`h/mathml/write/prepare` (*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.

```
\socket_new:nn {tagsupport/math/mathml/write/prepare}{0}
```

`On` (*plug*)

```
\socket_new_plug:nnn{tagsupport/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!

```
\sys_if_engine_luatex:T
 {
   \file_if_exist:nT { luamml.sty }
     {
       \RequirePackage  { luamml }
       \AddToHook{begindocument/end}
         {
           \bool_if:NT \g__tag_math_luamml_bool
             {
               \@ifpackageloaded { unicode-math }
                 { \__math_luamml_activate_write: }
                 { \msg_warning:nn { tag }{ unicode-math-missing } }
             }
         }
     }
 }
\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=false}}
  }
\cs_new_protected:Npn \__math_luamml_activate_write:
 {
```

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
   \AssignSocketPlug{tagsupport/math/struct/begin}{test-mathml}
   \AssignSocketPlug{tagsupport/math/struct/end}{test-mathml}
   \AssignSocketPlug{tagsupport/math/substruct/begin}{single}
   \AssignSocketPlug{tagsupport/math/substruct/end}{single}
   \int_set:Nn \l__luamml_pretty_int { 5 }
   \RegisterFamilyMapping\symsymbols{oms}
   \RegisterFamilyMapping\symletters{oml}
   \AssignSocketPlug{tagsupport/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
         {
```

```
          \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
```

this starts the process and set flag to 3, not really needed, as it is the default but set here for clarity:

```
  \luamml_flag_process:
```

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
    }
}
```

And now a key to deactivate luamml

```

\keys_define:nn { __tag / setup }
  {
    math/mathml/luamml .bool_gset:N = \g__tag_math_luamml_bool,
    math/mathml/luamml .usage:n=preamble
  }
```

pport/math/mathml/write (*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

```
\socket_new:nn {tagsupport/math/mathml/write}{0}
```

On (*plug*)

```
\socket_new_plug:nnn{tagsupport/math/mathml/write}{On}
 {
    \iow_now:Ne \g__math_writedummy_iow
     {
      \l__math_mathml_write_before_tl
      <math></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
             }
            \AssignSocketPlug {tagsupport/math/mathml/write/prepare}{On}
            \AssignSocketPlug {tagsupport/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
            }
        }
    }
    \bool_if:NT\g__tag_math_mathml_AF_bool
      {
        \typeout{Info:~Activating~mathml~support}
        \AssignSocketPlug{tagsupport/math/struct/begin}{test-mathml}
        \AssignSocketPlug{tagsupport/math/struct/end}{test-mathml}
```

mathml handling doesn't like subparts, so we disable them for now:

```
      \AssignSocketPlug{tagsupport/math/substruct/begin}{single}
      \AssignSocketPlug{tagsupport/math/substruct/end}{single}
      \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:`.)

5.6 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`. *This function is documented on page* **??**.)

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_start: }{ \tag_stop: }
 }
\cs_generate_variant:Nn \__math_tag_if_mathstyle:nn {en}
```

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

5.7 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
  }
```

5.8 Sockets

5.8.1 Main inline math sockets

`pport/math/inline/begin` (*socket*)
`support/math/inline/end` (*socket*)
`h/inline/formula/begin` (*socket*)
`ath/inline/formula/end` (*socket*)

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!

```
\socket_new:nn {tagsupport/math/inline/begin}{0}
\socket_new:nn {tagsupport/math/inline/end}{0}
\socket_new:nn {tagsupport/math/inline/formula/begin}{1} %
\socket_new:nn {tagsupport/math/inline/formula/end}{0}
```

`MC` (*plug*)

```
\socket_new_plug:nnn
  {tagsupport/math/inline/begin}
  {MC}
  {\tag_mc_end_push:}
\socket_new_plug:nnn
  {tagsupport/math/inline/end}
  {MC}
  {\tag_mc_begin_pop:n{}}
```

We probably will want to test different tagging recipes.

`default` (*plug*)

```
\socket_new_plug:nnn
  {tagsupport/math/inline/formula/begin}
  {default}
  {
    \socket_use:n{tagsupport/math/content}
    \socket_use:n{tagsupport/math/struct/begin}
```

TODO: does inline math need subformula handling?

```
    % inner formula if multiple parts (not really implemented yet)
    \socket_use:n{tagsupport/math/substruct/begin}
    #1
    \socket_use:n{tagsupport/math/end}
```

```
  }
\socket_new_plug:nnn
  {tagsupport/math/inline/formula/end}
  {default}
  {
    \socket_use:n{tagsupport/math/substruct/end}
    \socket_use:n{tagsupport/math/struct/end}
  }
```

5.8.2 Main display math sockets

`ort/math/display/begin` (*socket*)
`pport/math/display/end` (*socket*)
`/display/formula/begin` (*socket*)
`th/display/formula/end` (*socket*)

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!

```
\socket_new:nn {tagsupport/math/display/begin}{0}
\socket_new:nn {tagsupport/math/display/end}{0}
\socket_new:nn {tagsupport/math/display/formula/begin}{1} %
\socket_new:nn {tagsupport/math/display/formula/end}{0}
```

`default` (*plug*)

```
\socket_new_plug:nnn
  {tagsupport/math/display/begin}
  {default}
  { \__tag_tool_close_P:  }
\socket_new_plug:nnn
  {tagsupport/math/display/end}
  {default}
  {
  }
```

`default` (*plug*)

```
\socket_new_plug:nnn
  {tagsupport/math/display/formula/begin}
  {default}
  {
    \socket_use:n{tagsupport/math/content}
    \socket_use:n{tagsupport/math/struct/begin}
    \socket_use:n{tagsupport/math/substruct/begin}
    #1
    \socket_use:n{tagsupport/math/end}
  }
\socket_new_plug:nnn
  {tagsupport/math/display/formula/end}
  {default}
  {
    \socket_use:n{tagsupport/math/substruct/end}
    \socket_use:n{tagsupport/math/struct/end}
  }
```

5.8.3 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}
      }
   }
```

tagsupport/math/content (*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

```
\socket_new:nn {tagsupport/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*)

```
\socket_new_plug:nnn
  {tagsupport/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*)

```
\socket_new_plug:nnn
  {tagsupport/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    {}
  }

\socket_assign_plug:nn {tagsupport/math/content}{alt+source}
```

`pport/math/struct/begin` (*socket*) `upport/math/struct/end` (*socket*) For the main structure we use a socket too. This allow 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`

```
\socket_new:nn {tagsupport/math/struct/begin}{0}
\socket_new:nn {tagsupport/math/struct/end}{0}
```

`default` (*plug*) TODO: think about some naming convention ...

```
\socket_new_plug:nnn
  {tagsupport/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 }
    \tag_struct_begin:n
     {
       tag=Formula,
       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
     }
  }
```

```
\socket_new_plug:nnn
  {tagsupport/math/struct/end}
  {default}
  { \tag_struct_end: }

\socket_assign_plug:nn {tagsupport/math/struct/begin}{default}
\socket_assign_plug:nn {tagsupport/math/struct/end}{default}
```

`test-mathml` (*plug*) This (test-)socket tries to add a mathml-AF to formula. It is activated if a mathml.html has been found and loaded. Additionally it also sets an attribute (this can perhaps be done by default anyway.) 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

```
\socket_new_plug:nnn
  {tagsupport/math/struct/begin}
  {test-mathml}
  {
   \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}
       }
    }
    {
      \typeout{WARNING:~mathml~missing~for~hash~\l__math_content_hash_tl}
    }
   \socket_use:n {tagsupport/math/mathml/write/prepare}
   \socket_use:n {tagsupport/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
     }
   }
```

not really needed but looks more symmetric:

```
\socket_new_plug:nnn
  {tagsupport/math/struct/end}
  {test-mathml}
  {
    \tag_struct_end:
  }
```

t/math/substruct/begin (*socket*) This holds the code to handle subparts of the formula.

ort/math/substruct/end (*socket*)

```
\socket_new:nn {tagsupport/math/substruct/begin}{0}
\socket_new:nn {tagsupport/math/substruct/end}{0}
```

default (*plug*)

```
\socket_new_plug:nnn
  {tagsupport/math/substruct/begin}
  {default}
  { \grabaformulapartandstart }
\socket_new_plug:nnn
  {tagsupport/math/substruct/end}
  {default}
  {
    \tagmcend
    \if@subformulas
      \tagstructend
    \fi
 }
\socket_assign_plug:nn {tagsupport/math/substruct/begin}{default}
\socket_assign_plug:nn {tagsupport/math/substruct/end}{default}
```

single (*plug*) We need an option to disable subparts as it is unclear if consumers can handle them:

```
\socket_new_plug:nnn
  {tagsupport/math/substruct/begin}
  {single}
  {
    \typeout{====>subpart~splitting~deactivated}
    \typeout{====>grabbed~math=\meaning\g__math_grabbed_math_tl}
```

```
    \tag_mc_begin:n{}
  }
\socket_new_plug:nnn
  {tagsupport/math/substruct/end}
  {single}
  { \tag_mc_end:}
```

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

```
\socket_new:nn {tagsupport/math/end}{0}
```

`\__tag_math_disable:` Similar to the table code we collect the plugs that should be assigned to do nothing if we don't want tagging

```
\cs_new_protected:Npn \__tag_math_disable:
  {
    \socket_assign_plug:nn {tagsupport/math/inline/begin}{noop}
    \socket_assign_plug:nn {tagsupport/math/inline/end}{noop}
    \socket_assign_plug:nn {tagsupport/math/inline/formula/begin}{identity}
    \socket_assign_plug:nn {tagsupport/math/inline/formula/end}{noop}
    \socket_assign_plug:nn {tagsupport/math/display/begin}{noop}
    \socket_assign_plug:nn {tagsupport/math/display/end}{noop}
    \socket_assign_plug:nn {tagsupport/math/display/formula/begin}{identity}
    \socket_assign_plug:nn {tagsupport/math/display/formula/end}{noop}
  }
```

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

`\__tag_math_enable:` Similar to the table code we collect the default plugs that should be assigned if we want tagging

```
\cs_new_protected:Npn \__tag_math_enable:
  {
    \socket_assign_plug:nn {tagsupport/math/inline/begin}{MC}
    \socket_assign_plug:nn {tagsupport/math/inline/end}{MC}
    \socket_assign_plug:nn {tagsupport/math/inline/formula/begin}{default}
    \socket_assign_plug:nn {tagsupport/math/inline/formula/end}{default}
    \socket_assign_plug:nn {tagsupport/math/display/begin}{default}
    \socket_assign_plug:nn {tagsupport/math/display/end}{default}
    \socket_assign_plug:nn {tagsupport/math/display/formula/begin}{default}
    \socket_assign_plug:nn {tagsupport/math/display/formula/end}{default}
  }
```

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

At begin document we can activate:

```
\AtBeginDocument{\tag_if_active:T{\__tag_math_enable: }}
```

5.9 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* 3.)

5.10 Content grabbing

`\__math_grab_dollar:w` `\__math_grab_dollar: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_dollar:n #1
  {
```

To avoid further work with entirely empty math segments, we test before doing anything more.

```
    \tl_if_blank:nF {#1}
      {
        \__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_or:nnTF
        {\legacy_if_p:n { measuring@ }}
        { \l__math_fakemath_bool }
        { #1 $ }
        {
            \tag_socket_use:n  {math/inline/begin} %end P-MC
            \tag_socket_use:nn {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_dollar: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_dollar: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}
        \socket_use:n {tagsupport/math/display/begin}
        \socket_use:nn{tagsupport/math/display/formula/begin}{#1}
        $$
      }
  }
```

The end code is added through a `\aftergroup` so we store it inside a command.

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

TODO why is that needed? where is para-tagging disabled?

```
    \tagpdfparaOn
    \l__math_tmpa_skip \lastskip
    \socket_use:n{tagsupport/math/display/formula/end}
    \penalty \postdisplaypenalty
```

This reinserts the below display skips. It must be doubled to get the right amount:

```
  \skip_vertical:n { -\l__math_tmpa_skip * 2 }
%
  \@doendpe             % this has no \end{...} to take care of it
}

```

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

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

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

(*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`.)

5.11 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_dollar_loop:`
`\__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_dollar_loop:
  {
    \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_dollar_loop:` *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_\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). The need to cover `begin␣` is that at the start of a cell, TEX will expand `\begin` but the LATEX robust mechanism will mean this yields `begin␣`. If`\begin` were protected, that would not be needed.

```
\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 \\ : }
  {
    \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_eq:cc { __math_grab_loop_ \token_to_str:N \begin \c_space_tl : }
  { __math_grab_loop_ \token_to_str: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_dollar:n \l__math_grabbed_tl
  }
```

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

5.12 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: }
     }
  }
```

TODO: the following command is neither documented nor used. Is is needed?

```
\cs_new_protected:Npn \math_register_odd_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:c { __math_env_ #1 _end: }
% needed if we don't have $$...$$
%         \exp_not:n { \typeout{---> @kernel@math@registered@end }}
```

```
        \exp_not:n { \@kernel@math@registered@end }
      }
  }


%  FMi: compare with block change!
%
%  \DeclareRobustCommand*\begin[1]{%
%  \UseHook{env/#1/before}%
%  \@ifundefined{#1}%
%    {\def\reserved@a{\@latex@error{Environment #1 undefined}\@eha}}%
%    {\def\reserved@a{\def\@currenvir{#1}%
%        \edef\@currenvline{\on@line}%
%        \@execute@begin@hook{#1}%
%        \csname #1\endcsname}}%
%  \@ignorefalse
%  \begingroup
%  \@endpefalse  % tmp!!! is it ok to drop this here?
%  \reserved@a}


\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:
       }
  \socket_use:nn{tagsupport/math/display/formula/begin}{}
  \tagpdfparaOff
%  \typeout{==>MC1}\ShowTagging{mc-current}
}

\cs_new:Npn \@kernel@math@registered@end {
%  \typeout{==>MC2}\ShowTagging{mc-current}
  \para_raw_end:
  \tagpdfparaOn
```

```
  \socket_use:n{tagsupport/math/display/formula/end}
%  \typeout{==>MC3}\ShowTagging{mc-current}
  \@endpetrue
}

\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 3.*)

`\__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`.)

5.13 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. These functions are documented on page* ??.)

`\(` `\)`

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* `\(` *and* `\)`*. These functions are documented on page* ??.)

`\[` `\]` 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* `\[` *and* `\]`*. These functions are documented on page* ??.)

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`. *This function is documented on page* ??.)

5.14 `\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
    \iftrue  % this may have to be a settable flag!
%        \typeout{==>~ in~ everydisplay}
```

flipping the `\belowdisplay` values is done so that we get (assumption) a negative skip and not make the page bigger then we take that out, then we add the tagging code (in `\__math_tag_dollardollar_display_end`) and then we put a real `\postdisplaypenalty` in and the right skip (of which we don't know if it is short or a normal `\belowdisplayskip`). This might need some refinement if that skip is actually negative from the start (not sure it ever is and is worth bothering about)

```
        \skip_set:Nn \belowdisplayskip      {-\belowdisplayskip}
        \skip_set:Nn \belowdisplayshortskip {-\belowdisplayshortskip}
        \int_set:Nn \postdisplaypenalty {10000}
        \group_insert_after:N \__math_tag_dollardollar_display_end:
    \fi
    \bool_if:NF \l__math_collected_bool
      {
        \bool_set_true:N \l__math_collected_bool
        \__math_grab_dollardollar:w
      }
  }
```

5.15 Modifying kernel environments

We need to cover this even though it is, of course, not encouraged.

```
\math_register_env:n { eqnarray }
\math_register_env:n { eqnarray* }
```

Tabulars currently contain a $ that shouldn't trigger math tagging.

```
\RequirePackage{array}
\tl_if_in:NnT\@tabular{$}
 {
   \def\@tabular{%
   \leavevmode
```

```
    \UseTaggingSocket{tbl/hmode/begin}%
    \hbox \bgroup
    \bool_set_true:N \l__math_collected_bool
    $
    \bool_set_false:N \l__math_collected_bool
    \col@sep\tabcolsep \let\d@llarbegin\begingroup
                                      \let\d@llarend\endgroup
```

A proper switching mechanism is needed: for the present, do directly.

```
    \cs_set_protected:Npn \__math_grab_dollar:w { \__math_grab_dollar_loop: }
    \@tabarray}
  }
```

`\__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`*. This function is documented on page* **??**.)

5.16 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
 }
```

5.17 Modifying amsmath

`\__math_amsmath_align@:nn` `\__math_amsmath_gather@:n` `\__math_amsmath_multline@:n` `\align@` `\gather@` `\multline@` Mark up all of the display environments as the content is captured anyway. We then use an internal macro in each environment type to insert the processing code. Each of these is slightly different, so we cannot use a simple loop here. The test for `\split@tag` is required as the `split` environment internally uses `gather` *when not within an* amsmath environment, for example inside `equation`. Without the precaution, we'd get two copies of the grabbed math, the second of which would start with `\split@tag`.

```


\tl_gput_right:Nn \@kernel@before@begindocument {
%
\renewenvironment{gather*}{%
  \start@gather\st@rredtrue
}
{%
```

```
% this redirection doesn't work if we alter "gather"!
  %  \endgather
% so replace it with its real meaning
  \math@cr \black@\totwidth@ \egroup
  $$\ignorespacesafterend
}
\def\common@align@ending {
  \math@cr \black@\totwidth@
  \egroup
  \ifingather@
    \restorealignstate@
    \egroup
    \nonumber
    \ifnum0=`{\fi\iffalse}\fi
  \else
    $$%
  \fi
  \ignorespacesafterend
}
\renewenvironment{alignat}{%
  \start@align\z@\st@rredfalse
}{%
  \common@align@ending
}
\renewenvironment{alignat*}{%
  \start@align\z@\st@rredtrue
}{%
  \common@align@ending
}
\renewenvironment{xalignat}{%
  \start@align\@ne\st@rredfalse
}{%
  \common@align@ending
}
\renewenvironment{xalignat*}{%
  \start@align\@ne\st@rredtrue
}{%
  \common@align@ending
}
\renewenvironment{xxalignat}{%
  \start@align\tw@\st@rredtrue
}{%
  \common@align@ending
}
\renewenvironment{align}{%
  \start@align\@ne\st@rredfalse\m@ne
}{%
  \common@align@ending
}
\renewenvironment{align*}{%
  \start@align\@ne\st@rredtrue\m@ne
}{%
  \common@align@ending
}
```

```
\renewenvironment{flalign}{%
  \start@align\tw@\st@rredfalse\m@ne
}{%
  \common@align@ending
}
\renewenvironment{flalign*}{%
  \start@align\tw@\st@rredtrue\m@ne
}{%
  \common@align@ending
}
%
\renewenvironment{multline*}{\start@multline\st@rredtrue}
{%
  \iftagsleft@ \@xp\lendmultline@ \else \@xp\rendmultline@ \fi
  \ignorespacesafterend
}
```

Also for `false`?

```
\def\measuring@true{\let\ifmeasuring@\iftrue\tag_stop:}
%
  \math_register_halign_env:nn {align}{}
  \math_register_halign_env:nn {align*}{}
  \math_register_halign_env:nn {alignat}{}
  \math_register_halign_env:nn {alignat*}{}
  \math_register_halign_env:nn {flalign}{}
  \math_register_halign_env:nn {flalign*}{}
  \math_register_halign_env:nn {gather}{}
  \math_register_halign_env:nn {gather*}{}
  \math_register_halign_env:nn {multline}{}
  \math_register_halign_env:nn {multline*}{}
  \math_register_halign_env:nn {xalignat}{}
  \math_register_halign_env:nn {xalignat*}{}
  \math_register_halign_env:nn {xxalignat}{}
  %
  \@namedef{maketag @ @ @} #1{%
%    \typeout{--->maketag @ @ @}
    \ifmeasuring@
      \hbox{\m@th\normalfont#1}%
    \else
      \tagmcend \tagstructbegin{tag=Lbl}%
      \tagmcbegin{tag=Lbl}%
      \hbox{\m@th\normalfont#1}%
      \tagmcend \tagstructend \tagmcbegin{}%
    \fi
  }
\@namedef{math@cr @ @ @ gather}{%
    \ifst@rred\nonumber\fi
   &\relax
    \make@display@tag
%
    \maybestartnewformulatag
%
    \ifst@rred\else\global\@eqnswtrue\fi
    \global\advance\row@\@ne
```

```
    \cr
}
\@namedef{math@cr @ @ @ align}{%
  \ifst@rred\nonumber\fi
  \if@eqnsw \global\tag@true \fi
  \global\advance\row@\@ne
  \add@amps\maxfields@
  \omit
  \kern-\alignsep@
  \iftag@
    \setboxz@h{\@lign\strut@{\make@display@tag}}%
    \place@tag
  \fi
%
    \maybestartnewformulatag
%
  \ifst@rred\else\global\@eqnswtrue\fi
  \global\lineht@\z@
  \cr
}
\def\restore@math@cr{\@namedef{math@cr @ @ @}{
%
    \maybestartnewformulatag
%
    \cr}}
\restore@math@cr
}
```

(*End of definition for* `\__math_amsmath_align@:nn` *and others. These functions are documented on page* ??.)

`\__math_split_at_nl:NN` This splits grabbed math at newlines.

```
\cs_new:Npn \__math_split_at_nl:NN #1#2 {
  \tl_set:Nf \l__math_tmpa_tl {
      \exp_after:wN \__math_split_at_nl_first:w #1 \\ \q_nil \\ \s_stop }
  \exp_after:wN \__math_split_at_nl_aux:nnNN \l__math_tmpa_tl #1 #2
}
```

and the auxiliary commands

```
\cs_new:Npn \__math_split_at_nl_first:w #1 \\ #2 \\ #3 \s_stop
  {
    \quark_if_nil:nTF {#2}
      { {#1} {  } }
      {
        \__math_split_chk_if_begin:ww #1 \begin \q_nil \s_mark
          #2 \\ #3 \s_stop
      }
  }

\cs_new_protected:Npn \__math_split_at_nl_aux:nnNN #1 #2 #3 #4
  {
    \tl_gset:Nn #4 {#1}
    \tl_gset:Nn #3 {#2}
```

```
  }

\cs_new:Npn \__math_split_chk_if_begin:ww
   #1 \begin #2 #3 \s_mark #4 \\ \q_nil \\ \s_stop
  {
    \quark_if_nil:nTF {#2}
      { {#1} {#4} }
      {
        \exp_after:wN \__math_split_collect_one_end:w
          \__math_split_cleanup_begin_q_nil:w #1 \begin{#2} #3 \\ #4 \s_stop
            { } { 1 }
      }
  }

\cs_new:Npn \__math_split_cleanup_begin_q_nil:w #1 \begin \q_nil {#1}

\cs_new:Npn \__math_split_collect_one_end:w #1 \end #2 #3 \s_stop #4 #5
  {
    \exp_args:Nf \__math_split_check_count_begins:nnnn
      { \__math_split_count_begins:n { #4 #1 } } {#5}
      { #4 #1 \end{#2} } {#3}
  }
\cs_new:Npn \__math_split_count_begins:n #1
  { \int_eval:n { 0 \__math_split_count_begins:w #1 \begin \q_nil } }

\cs_new:Npn \__math_split_count_begins:w #1 \begin #2
  { \quark_if_nil:nF {#2} { +1 \__math_split_count_begins:w } }

\cs_new:Npn \__math_split_check_count_begins:nnnn #1 #2 #3 #4
  {
    \int_compare:nNnTF {#1} = {#2}
      {
        \exp_last_unbraced:Nf \__math_split_final_cleanup:nn
          { \__math_split:n { \__math_split_guard:n {#3} #4 } }
      }
      {
        \exp_args:No \use_ii_i:nn
          { \exp_after:wN { \int_value:w \int_eval:n { #2 + 1 } } }
          { \__math_split_collect_one_end:w #4 \s_stop {#3} }
      }
  }
\cs_new:Npn \__math_split_final_cleanup:nn #1 #2
  {
      \exp:w \__math_split_final_cleanup:w #1
        \__math_split_guard:n \q_nil \s_mark { }
      {#2}
  }
\cs_new:Npn \__math_split_final_cleanup:w #1 \__math_split_guard:n #2 #3 \s_mark #4
  {
    \quark_if_nil:nTF {#2}
      { \exp_end: { #4 #1 } }
      { \__math_split_final_cleanup:w #3 \s_mark { #4 #1 #2 } }
  }

```

```
\cs_new:Npn \__math_split:n #1 {
    \__math_split_at_nl_first:w #1 \\ \q_nil \\ \s_stop }

% this looks unused.
%\NewDocumentCommand \splitnl { mm +m }
%  {
%    \tl_set:Nf \l__math_tmpa_tl { \split:n {#3} }
%    \show \l__math_tmpa_tl
%    \exp_after:wN \__splitnl_aux:nnNN \l__math_tmpa_tl #1 #2
%  }
```

(*End of definition for* `\__math_split_at_nl:NN`.)

`\maybestartnewformulatag`

```
\newif\if@subformulas
\tl_new:N \result

\cs_new_protected:Npn\grabaformulapartandstart {
  \__math_split_at_nl:NN  \g__math_grabbed_math_tl \result
  \typeout{====>first-result=\meaning\result}
  \typeout{====>first-tmpmathcontent=\meaning\g__math_grabbed_math_tl}
  \tl_if_empty:NTF \g__math_grabbed_math_tl
     {
       \typeout{====>formula~ has~ no~ subparts}
       \global\@subformulasfalse
     }
     {
       \typeout{====>formula~ has~ subparts}
       \global\@subformulastrue
       \edef\resulttitle{\g__math_grabbed_env_tl\space (part)}
       \tagstructbegin{tag=Formula,
```

For now we don't put real content in /alt or /ActualText on subformulas but we add a short text to satisfy the pdf/ua-2 validator

```
%         alt=\result,
          alt = subformula,
          title-o=\resulttitle
        }
    }
    \tagmcbegin{}
}

\cs_new_protected:Npn\grabaformulapartandmayberestart {
  \__math_split_at_nl:NN  \g__math_grabbed_math_tl \result
  \typeout{====>result=\meaning\result}
  \typeout{====>tmpmathcontent=\meaning\g__math_grabbed_math_tl}
%  \tl_if_empty:NTF \g__math_grabbed_math_tl
%     {
%       \typeout{====>tmpmathcontent=empty}
%     }
%     {
%       \typeout{====>tmpmathcontent=not-empty}
       \edef\resulttitle{\g__math_grabbed_env_tl\space (part)}
       \tagstructbegin{tag=Formula,
```

```
          alt=\result,
          title-o=\resulttitle
        }
%    }
    \tagmcbegin{}
}
```

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

```
\def\maybestartnewformulatag {
\if@subformulas
 \ifmeasuring@\else
%
  \tl_if_empty:NF \g__math_grabbed_math_tl
     {
       \tagmcend
       \tagstructend
       \grabaformulapartandmayberestart
     }
 \fi
\fi
}
```

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

```
%\AddToHook{package/breqn/after}{
%  \typeout{===>~ in~ hook}
%  \math_register_halign_env:nn {dmath}{}
%  \math_register_halign_env:nn {dgroup*}{}
%}
```

```
\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.

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V