# The luamml package *

Marcel Krüger

March 7, 2025

## 1 Use case

When generating output for the web or tagged output, mathematical content should often be represented as MathML. This uses LuaTEX callbacks to automatically attempt to convert LuaTEX math mode output into MathML.

## 2 Usage

The luamml package is designed to be used in automated ways by other packages and usually should not be invoked directly by the end user. For experiments, `luamml-demo` is included which provides easier to use interfaces.

Add in your preamble

```
\usepackage[files]{luamml-demo}
```

This will trigger the output of individual files for each block of math output containing corresponding MathML.

Alternatively

```
\usepackage[l3build]{luamml-demo}
```

will generate a single file with a concatenation of all MathML blocks.

For automated use, the luamml package can be included directly, followed by enclosing blocks which should generate files with `luamml_begin_single_file:` and `luamml_end_single_file:`. The filename can be set with `luamml_set_filename:n`.

## 3 Improving MathML conversion

When using constructs which do not automatically get converted in acceptable form, conversion hints can be provided with `luamml_annotate:en`. This allows to provide a replacement MathML structure in Lua table form, for example

```
\luamml_annotate:en {
  nucleus = true,
  core = {[0] = 'mi', 'TeX'},
}{
  \hbox{\TeX}
}
```

*This document corresponds to luamml v0.5.0, dated 2025-03-06.

produces a `<mi>TeX</mi>` element in the output instead of trying to import TEX as a mathematical expression.

It it possible to add a structure around the construct, stash that structure and then to tell `luamml_annotate:en` to insert it later inside the math. For this the keys `struct` (which takes a label as argument) or `structnum` (which takes a structure number) can be used. For example

```
$a = b \quad
 \tagstructbegin{tag=mtext,stash}\tagmcbegin{}
 \luamml_annotate:en{nucleus=true,structnum=\tag_get:n{struct_num}}
 {\mbox{some~text~with~\emph{structure}}}
 \tagmcend\tagstructend
$
```

Such a construction should check that the flag for structure elements has actually been set to avoid orphaned structures if the stashed structure is ignored.

More about the table structure is explained in an appendix.

# 4 Features & Limitations

Currently all mathematical expressions which purely contain Unicode encoded math mode material without embedded non-math should get converted successfully. Usage with non-Unicode math (TEX's 8-bit math fonts) is highly experimental and undocumented. Any attempt to build complicated structures by embedding arbitrary TEX code in the middle of math mode needs to have a MathML replacement specified. We try to automate more cases in the future.

# A Luamml's representation of XML and MathML

In the following I assume basic familiarity with both LuaTEX's representation of math noads and MathML.

## A.1 Representation of XML elements

In many places, luamml passes around XML elements. Every element is represented by a Lua table. Element `0` must always be present and is a string representing the tag name. The positive integer elements of the table represent child elements (either strings for direct text content or nested tables for nested elements). All string members which do not start with a colon are attributes, whose value is the result of applying `tostring` to the field value. This implies that these values should almost always be strings, except that the value `0` (since it never needs a unit) can often be set as a number. For example the XML document

```
<math block="display">
  <mn>0</mn>
  <mo> &lt; </mo>
  <mi mathvariant="normal">x</mi>
</math>
```

would be represented by the Lua table

```
{[0] = "math", block="display",
  {[0] = "mn", "0"},
  {[0] = "mo", "<"},
  {[0] = "mi", mathvariant="normal", "x"}
}
```

## A.2 Expression cores

MathML knows the concept of "embellished operators": "T"he precise definition of an "embellished operator" is:

- an `<mo>` element;
- or one of the elements `<msub>`, `<msup>`, `<msubsup>`, `<munder>`, `<mover>`, `<munderover>`, `<mmultiscripts>`, `<mfrac>`, or `<semantics>` (§ 5.1 Annotation Framework), whose first argument exists and is an embellished operator;
- or one of the elements `<mstyle>`, `<mphantom>`, or `<mpadded>`, such that an mrow containing the same arguments would be an embellished operator;
- or an `<maction>` element whose selected sub-expression exists and is an embellished operator;
- or an `<mrow>` whose arguments consist (in any order) of one embellished operator and zero or more space-like elements.

For every embellished operator, MathML calls the `<mo>` element defining the embellished operator the "core" of the embellished operator.

Luamml makes this slightly more general: Every expression is represented by a pair of two elements: The expression and it's core. The core is always a `<mo>`, `<mi>`, or `<mn>`, `nil` or s special marker for space like elements.

If and only if the element is a embellished operator the core is a `<mo>` element representing the core of the embellished operator. The core is a `<mi>` or a `<mn>` element if and only if the element would be an embellished operator with this core if this element where a `<mo>` element. The core is the special space like marker for space like elements. Otherwise the core is `nil`.

## A.3 Translation of math noads

A math lists can contain the following node types: noad, fence, fraction, radical, accent, style, choice, ins, mark, adjust, boundary, whatsit, penalty, disc, glue, and kern. The "noads"

### A.3.1 Translation of kernel noads

The math noads of this list contain nested kernel noads. So in the first step, we look into how kernel nodes are translated to math nodes.

**`math_char` kernel noads** First the family and character value in the `math_char` are used to lookup the Unicode character value of this `math_char`. (For `unicode-math`, this is usually just the character value. Legacy maths has to be remapped based on the family.) Then there are two cases: The digits `0` to `9` are mapped to `<mn>` elements, everything else becomes a `<mi>` element with `mathvariant` set to `normal`. (The `mathvariant` value might get suppressed if the character defaults to mathvariant `normal`.) In either case, the `tex:family` attribute is set to the family number if it's not `0`.

The core is always set to the expression itself. E.g. the `math_char` kernel noad `\fam3 a` would become (assuming no remapping for this family)

```
{[0] = 'mi',
  mathvariant = 'normal',
  ["tex:family"] = 3,
  "a"
}
```

### A.3.2 `sub_box` kernel noads

I am open to suggestions how to convert them properly.

### A.3.3 `sub_mlist` kernel noads

The inner list is converted as a `<mrow>` element, with the core being the core of the `<mrow>` element. See the rules for this later.

### A.3.4 `delim` kernel noads

If the `small_char` is zero, these get converted as space like elements of the form

```
{[0] = 'mspace',
  width = '1.196pt',
}
```

where 1.196 is replaced by the current value of `\nulldelimiterspace` converted to `bp`.

Otherwise the same rules as for `math_char` apply, except that instead of `mi` or `<mn>` elements, `mo` elements are generated, `mathvariant` is never set, `stretchy` is set to `true` if the operator is not on the list of default stretchy operators in the MathML specification nd `lspace` and `rspace` attributes are set to zero.

### A.3.5 `acc` kernel noads

Depending on the surrounding element containing the `acc` kernel noad, it is either stretchy or not. If it's stretchy, the same rules as for `delim` apply, except that `lspace` and `rspace` are not set. Otherwise the `stretchy` attribute is set to false if the operator is on the list of default stretchy operators.

# B Package Implementation

## B.1 Initialization

```
⟨@@=luamml⟩
⟨*luatex⟩
\ProvidesExplPackage {luamml} {2025-03-06} {0.5.0}
  {Automatically generate presentational MathML from LuaTeX math expressions}
⟨/luatex⟩
⟨*pdftex⟩
\ProvidesExplPackage {luamml-pdf} {2025-03-06} {0.5.0}
  {MathML generation for L u a pdfLaTeX}
⟨/pdftex⟩
```

## B.2 Initialization

These variable have to appear before the Lua module is loaded and will be used to communicate information to the callback.

Here `\tracingmathml` does not use a expl3 name since it is not intended for programming use but only as a debugging helper for the user. The other variables are internal, but we provide public interfaces for setting them later.

```
\int_new:N \l__luamml_flag_int
\int_new:N \l__luamml_pretty_int
⟨luatex⟩\tl_new:N \l__luamml_filename_tl
\tl_new:N \l__luamml_root_tl
\tl_set:Nn \l__luamml_root_tl { mrow }
\tl_new:N \l__luamml_label_tl
⟨pdftex⟩\int_new:N \g__luamml_formula_id_int
⟨luatex⟩\int_new:N \tracingmathml

\int_set:Nn \l__luamml_pretty_int { 1 }
```

Now we can load the Lua module which defines the callback. Of course until pdfTEXstarts implementing `\directlua` this is only done in LuaTEX.

```
⟨luatex⟩\lua_now:n { require'luamml-tex' }
```

## B.3 Hook

We also call a hook with arguments at the end of every MathML conversion with the result. Currently only implemented in LuaTEX since it immediately provides the output.

```
⟨*luatex⟩
\hook_new_with_args:nn { luamml / converted } { 1 }

\cs_new_protected:Npn \__luamml_output_hook:n {
  \hook_use:nnw { luamml / converted } { 1 }
}
\__luamml_register_output_hook:N \__luamml_output_hook:n
⟨/luatex⟩
```

## B.4 Flags

The most important interface is for setting the flag which controls how the formulas should be converted.

\luamml_process: Consider the current formula to be a complete, free-standing mathematical expression which should be converted to MathML. Additionally, the formula is also saved in the start_math node as with \luamml_save:.

```
\cs_new_protected:Npn \luamml_process: {
  \tl_set:Nn \l__luamml_label_tl {}
  \int_set:Nn \l__luamml_flag_int { 3 }
}
```

Temporarily for compatibility

```
\cs_set_eq:NN \luamml_flag_process: \luamml_process:
```

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

__luamml_maybe_structelem: A internal helper which can be added to a tag to preserve the external state of the structelem flag.

```
\cs_new:Npn \__luamml_maybe_structelem: {
  (
    8 * \int_mod:nn {
      \int_div_truncate:nn { \l__luamml_flag_int } {8}
    } {2}
  ) +
}
```

(*End of definition for* __luamml_maybe_structelem:.)

__luamml_style_to_num:N

```
\cs_new:Npn \__luamml_style_to_num:N #1 {
⟨luatex⟩  32 * #1
⟨*pdftex⟩
  \token_case_meaning:NnF #1 {
    \displaystyle {0}
    \textstyle {32}
    \scriptstyle {64}
    \scriptscriptstyle {96}
  } {
    \Invalid_mathstyle
  }
⟨/pdftex⟩
}
```

(*End of definition for* __luamml_style_to_num:N.)

\luamml_save:n
\luamml_save:nN
\luamml_save:nn
\luamml_save:nNn

Convert the current formula but only save it's representation in the math node without emitting it as a complete formula. This is useful when the expression forms part of a bigger formula and will be integrated into it's MathML tables later by special code. It optionally accepts three parameters: A label, one math style command (\displaystyle, \textstyle, etc.) which is the implicit math style (so the style which the surrounding code expects this style to have) and a name for the root element (defaults to mrow). If the root element name is mrow, it will get suppressed in some cases.

```
\cs_new_protected:Npn \luamml_save:n #1 {
  \tl_set:Nn \l__luamml_label_tl {#1}
  \int_set:Nn \l__luamml_flag_int { \__luamml_maybe_structelem: 1 }
}
\cs_new_protected:Npn \luamml_save:nN #1#2 {
```

```
  \tl_set:Nn \l__luamml_label_tl {#1}
  \int_set:Nn \l__luamml_flag_int { \__luamml_maybe_structelem: 17 + \__luamml_style_to_num:N
}
\cs_new_protected:Npn \luamml_save:nn #1 {
  \tl_set:Nn \l__luamml_label_tl {#1}
  \int_set:Nn \l__luamml_flag_int { \__luamml_maybe_structelem: 5 }
  \tl_set:Nn \l__luamml_root_tl
}
\cs_new_protected:Npn \luamml_save:nNn #1#2 {
  \tl_set:Nn \l__luamml_label_tl {#1}
  \int_set:Nn \l__luamml_flag_int { \__luamml_maybe_structelem: 21 + \__luamml_style_to_num:N
  \tl_set:Nn \l__luamml_root_tl
}
```

Temporarily for compatibility

```
\cs_set_eq:NN \luamml_flag_save:n \luamml_save:n
\cs_set_eq:NN \luamml_flag_save:nN \luamml_save:nN
\cs_set_eq:NN \luamml_flag_save:nn \luamml_save:nn
\cs_set_eq:NN \luamml_flag_save:nNn \luamml_save:nNn
```

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

`\luamml_ignore:` Completely ignore the math mode material.

```
\cs_new_protected:Npn \luamml_ignore: {
  \int_set:Nn \l__luamml_flag_int { 0 }
}
```

Temporarily for compatibility

```
\cs_set_eq:NN \luamml_flag_ignore: \luamml_ignore:
```

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

`\luamml_structelem:` Like `\luamml_process:`, but additionally adds PDF structure elements. This only works in LuaTEX and requires that the tagpdf package has been loaded *before* `luamml`.

```
⟨*luatex⟩
\cs_new_protected:Npn \luamml_structelem: {
  \tl_set:Nn \l__luamml_label_tl {}
  \int_set:Nn \l__luamml_flag_int { 11 }
}
```

Temporarily for compatibility

```
\cs_set_eq:NN \luamml_flag_structelem: \luamml_structelem:
⟨/luatex⟩
```

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

`\luamml_set_filename:n` Allows to set a filename to which the generated MathML gets written. Previous content from the file will get overwritten. This includes results written by a previous formula. Therefore this has to be called separately for every formula or it must expand to different values to be useful. The value is fully expanded when the file is written.

Only complete formulas get written into files (so formulas where `\luamml_process:` or `\luamml_structelem:` are in effect).

Only implemented in LuaTEX, in pdfTEX the arguments for `pdfmml` determine the output location.

```
⟨*luatex⟩
```

```
\cs_new_protected:Npn \luamml_set_filename:n {
  \tl_set:Nn \l__luamml_filename_tl
}
⟨/luatex⟩
```

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

`\luamml_begin_single_file:`
`\luamml_end_single_file:`

Everything between these two commands gets written into the same XML file. The filename is expanded when `\luamml_begin_single_file:` gets executed.

(Implemented in Lua)

(*End of definition for* `\luamml_begin_single_file:` *and* `\luamml_end_single_file:`. *These functions are documented on page* **??**.)

By default, the flag is set to assume complete formulas.

```
\luamml_process:
```

## B.5 Annotations

These are implemented very differently depending on the engine, but the interface should be the same.

### B.5.1 LuaTEX

```
⟨*luatex⟩
```

`\luamml_annotate:nen`
`\luamml_annotate:en`

A simple annotation scheme: The first argument is the number of top level noads to be annotated, the second parameter the annotation and the third parameter the actual list of math tokens. The first argument can be omitted to let LuaTEXdetermine the number itself.

Passing the first parameter explicitly is useful for any annotations which should be compatible with future pdfTEXversions of this functionality.

```
\cs_new_protected:Npn \luamml_annotate:nen #1#2#3 {
  \__luamml_annotate_begin:
    #3
  \__luamml_annotate_end:we \tex_numexpr:D #1 \scan_stop: {#2}
}

\cs_new_protected:Npn \luamml_annotate:en #1#2 {
  \__luamml_annotate_begin:
    #2
  \__luamml_annotate_end:e {#1}
}
```

(*End of definition for* `\luamml_annotate:nen` *and* `\luamml_annotate:en`. *These functions are documented on page* **??**.)

```
⟨/luatex⟩
```

### B.5.2 pdfTEX

⟨*pdftex⟩

__luamml_pdf_showlists: Here and in many other locations the pdfTEX implementation is based on \showlists, so we define a internal wrapper which sets all relevant parameters.

```
\cs_if_exist:NTF \showstream {
  \iow_new:N \l__luamml_pdf_stream
  \iow_open:Nn \l__luamml_pdf_stream { \jobname .tml }
  \cs_new_protected:Npn \__luamml_pdf_showlists: {
    \group_begin:
      \int_set:Nn \tex_showboxdepth:D { \c_max_int }
      \int_set:Nn \tex_showboxbreadth:D { \c_max_int }
      \showstream = \l__luamml_pdf_stream
      \tex_showlists:D
    \group_end:
  }
} {
  \cs_set_eq:NN \l__luamml_pdf_stream \c_log_iow
  \cs_set_eq:NN \__luamml_pdf_set_showstream: \scan_stop:
  \cs_new_protected:Npn \__luamml_pdf_showlists: {
    \group_begin:
      \int_set:Nn \l_tmpa_int { \tex_interactionmode:D }
      \int_set:Nn \tex_interactionmode:D { 0 }
      \int_set:Nn \tex_showboxdepth:D { \c_max_int }
      \int_set:Nn \tex_showboxbreadth:D { \c_max_int }
      \tex_showlists:D
      \int_set:Nn \tex_interactionmode:D { \l_tmpa_int }
    \group_end:
  }
}
```

(*End of definition for* __luamml_pdf_showlists:.)

\luamml_annotate:nen
\luamml_annotate:en

Now we can define the annotation commands for pdfTEX.

```
\cs_generate_variant:Nn \tl_to_str:n { e }
\int_new:N \g__luamml_annotation_id_int
\cs_new_protected:Npn \luamml_annotate:nen #1#2#3 {
  \int_gincr:N \g__luamml_annotation_id_int
  \iow_shipout_x:Nx \l__luamml_pdf_stream {
    LUAMML_MARK_REF:
    \int_use:N \g__luamml_annotation_id_int
    :
  }
  \iow_now:Nx \l__luamml_pdf_stream {
    LUAMML_MARK:
    \int_use:N \g__luamml_annotation_id_int
    :
    count = \int_eval:n {#1},
    #2
    \iow_newline:
    LUAMML_MARK_END
  }
  #3
}
```

```
\cs_new_protected:Npn \luamml_annotate:en #1#2 {
  \int_gincr:N \g__luamml_annotation_id_int
  \iow_shipout_x:Nx \l__luamml_pdf_stream {
    LUAMML_MARK_REF:
    \int_use:N \g__luamml_annotation_id_int
    :
  }
  \iow_now:Nx \l__luamml_pdf_stream {
    LUAMML_MARK:
    \int_use:N \g__luamml_annotation_id_int
    :
    count = data.count[\int_use:N \g__luamml_annotation_id_int],
    #1
    \iow_newline:
    LUAMML_MARK_END
  }
  \use:x {
    \iow_now:Nn \l__luamml_pdf_stream {
      LUAMML_COUNT:
      \int_use:N \g__luamml_annotation_id_int
    }
    \__luamml_pdf_showlists:
    \exp_not:n {#2}
    \iow_now:Nn \l__luamml_pdf_stream {
      LUAMML_COUNT_END:
      \int_use:N \g__luamml_annotation_id_int
    }
    \__luamml_pdf_showlists:
  }
}
```

(*End of definition for* `\luamml_annotate:nen` *and* `\luamml_annotate:en`*. These functions are documented on page* **??**.)

⟨/pdftex⟩

## B.6 Trigger for specific formula

This only applies for pdfTEX since in LuaTEX everything is controlled by the callback, but for compatibility the function is defined anyway.

`\luamml_pdf_write:` We could accept parameters for the flag and tag here, but for compatibility with LuaTEXthey are passed in macros instead.

⟨*pdftex⟩

```
\cs_new_protected:Npn \luamml_pdf_write: {
  \int_gincr:N \g__luamml_formula_id_int
  \iow_now:Nx \l__luamml_pdf_stream {
    LUAMML_FORMULA_BEGIN:
    \int_use:N \g__luamml_formula_id_int
    :
    \int_use:N \l__luamml_flag_int
    :
    \l__luamml_root_tl
    :
    \l__luamml_label_tl
```

```
  }
  \__luamml_pdf_showlists:
  \iow_now:Nx \l__luamml_pdf_stream {
    LUAMML_FORMULA_END
  }
}
⟨/pdftex⟩
⟨luatex⟩\cs_new_eq:NN \luamml_pdf_write: \scan_stop:
```

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

## B.7 Further helpers

`\RegisterFamilyMapping` The Lua version of this is defined in the Lua module.

```
⟨*pdftex⟩
\NewDocumentCommand \RegisterFamilyMapping {m m} {
  \iow_now:Nx \l__luamml_pdf_stream {
    LUAMML_INSTRUCTION:REGISTER_MAPPING: \int_use:N #1 : #2
  }
}
⟨/pdftex⟩
```

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

## B.8 Sockets

In various places luamml has to add code to kernel commands. This is done through sockets which are predeclared in lttagging.

### B.8.1 Save sockets

These sockets are wrappers around the `\luamml_save:...` commands They should be provided until 2025-06-01

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/save/nn_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/save/nn}{1}
   \AssignSocketPlug{tagsupport/math/luamml/save/nn}{noop}
   \NewSocket{tagsupport/math/luamml/save/nNn}{1}
   \AssignSocketPlug{tagsupport/math/luamml/save/nNn}{noop}
 }
\NewSocketPlug{tagsupport/math/luamml/save/nNn}{luamml}
 {
   \luamml_save:nNn #1
 }
\AssignSocketPlug{tagsupport/math/luamml/save/nNn}{luamml}
\NewSocketPlug{tagsupport/math/luamml/save/nn}{luamml}
 {
   \luamml_save:nn #1
 }
\AssignSocketPlug{tagsupport/math/luamml/save/nn}{luamml}
```

### B.8.2 sockets to annotate content

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/annotate/false_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/annotate/false}{2}
   \NewSocketPlug{tagsupport/math/luamml/annotate/false}{default}{#2}
   \AssignSocketPlug{tagsupport/math/luamml/annotate/false}{default}
 }
⟨*luatex⟩
\NewSocketPlug{tagsupport/math/luamml/annotate/false}{luamml}
 {
   \luamml_annotate:en { core = false }
    {
      #2
    }
 }
\AssignSocketPlug{tagsupport/math/luamml/annotate/false}{luamml}
⟨/luatex⟩
```

### B.8.3 socket plugs for the array package

The socket declaration can go with the 2025-06-01 release

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/array/finalize_plug_str }
  {
     \NewSocket{tagsupport/math/luamml/array/save}{0}
     \NewSocket{tagsupport/math/luamml/array/finalize}{0}
     \NewSocket{tagsupport/math/luamml/array/initcol}{0}
     \NewSocket{tagsupport/math/luamml/array/finalizecol}{1}
     \AssignSocketPlug{tagsupport/math/luamml/array/finalizecol}{noop}
  }
```

The luamml support makes only sense with luatex.

```
⟨*luatex⟩
\AddToHook{package/array/after}{\lua_now:n { require'luamml-array' }}
```

t/math/luamml/array/save (*plug*) The socket of this plug is used in `\endarray`.

```
\NewSocketPlug{tagsupport/math/luamml/array/save}{luamml}
 {
   \__luamml_array_save_array:
 }
```

th/luamml/array/finalize (*plug*) This socket of this plug is used in `\endarray`.

```
\NewSocketPlug{tagsupport/math/luamml/array/finalize}{luamml}
 {
   \mode_if_math:T { \__luamml_array_finalize_array: }
 }
```

ath/luamml/array/initcol (*plug*) The socket of this plug is used in `\@classz`.

```
\NewSocketPlug{tagsupport/math/luamml/array/initcol}{luamml}
 {
   \__luamml_array_init_col:
 }
```

luamml/array/finalizecol (*plug*) The socket of this plug is used used in `\@classz`.

```
\NewSocketPlug{tagsupport/math/luamml/array/finalizecol}{luamml}
 {
   \__luamml_array_finalize_col:w #1~
 }

\AssignSocketPlug{tagsupport/math/luamml/array/save}{luamml}
\AssignSocketPlug{tagsupport/math/luamml/array/finalize}{luamml}
\AssignSocketPlug{tagsupport/math/luamml/array/initcol}{luamml}
\AssignSocketPlug{tagsupport/math/luamml/array/finalizecol}{luamml}
⟨/luatex⟩
```

### B.8.4 amsmath alignments

This socket is used at the end of alignment cells and adds the content to the current row.

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/mtable/finalizecol_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/mtable/finalizecol}{1}
 }
⟨*luatex⟩
\NewSocketPlug{tagsupport/math/luamml/mtable/finalizecol}{luamml}
 {
   \use:c{__luamml_amsmath_add_#1_to_row:}
 }
\AssignSocketPlug{tagsupport/math/luamml/mtable/finalizecol}{luamml}

⟨/luatex⟩
```

These sockets save an inner table

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/mtable/innertable/save_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/mtable/innertable/save}{0}
   \NewSocket{tagsupport/math/luamml/mtable/smallmatrix/save}{0}
   \NewSocket{tagsupport/math/luamml/mtable/innertable/finalize}{0}
 }
⟨*luatex⟩
\NewSocketPlug{tagsupport/math/luamml/mtable/innertable/save}{luamml}
 {
   \__luamml_amsmath_save_inner_table:n \@currenvir
 }
\AssignSocketPlug{tagsupport/math/luamml/mtable/innertable/save}{luamml}
\NewSocketPlug{tagsupport/math/luamml/mtable/smallmatrix/save}{luamml}
 {
   \__luamml_amsmath_save_smallmatrix:
 }
\AssignSocketPlug{tagsupport/math/luamml/mtable/smallmatrix/save}{luamml}
\NewSocketPlug{tagsupport/math/luamml/mtable/innertable/finalize}{luamml}
 {
   \__luamml_amsmath_finalize_inner_table:
 }
\AssignSocketPlug{tagsupport/math/luamml/mtable/innertable/finalize}{luamml}
⟨/luatex⟩
```

This socket finalize the `mtable` in alignments like align or gather. It takes an argument, the environment. It should be used normally with `\UseExpandableTaggingSocket`.

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/mtable/finalize_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/mtable/finalize}{1}
   \AssignSocketPlug{tagsupport/math/luamml/mtable/finalize}{noop}
 }
⟨*luatex⟩
\NewSocketPlug{tagsupport/math/luamml/mtable/finalize}{luamml}
 {
   \__luamml_amsmath_finalize_table:n {#1}
 }
\AssignSocketPlug{tagsupport/math/luamml/mtable/finalize}{luamml}
⟨/luatex⟩
```

This socket adds attributes for the alignment in `multline`. It takes an argument, the alignment.

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/mtable/aligncol_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/mtable/aligncol}{1}
   \AssignSocketPlug{tagsupport/math/luamml/mtable/aligncol}{noop}
 }
⟨*luatex⟩
\NewSocketPlug{tagsupport/math/luamml/mtable/aligncol}{luamml}
 {
   \__luamml_amsmath_set_row_columnalign:n {#1}
 }
\AssignSocketPlug{tagsupport/math/luamml/mtable/aligncol}{luamml}
⟨/luatex⟩
```

### B.8.5 Tags and labels

These sockets save and set tags and labels in alignments.

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/mtable/tag/save_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/mtable/tag/save}{0}
   \NewSocket{tagsupport/math/luamml/mtable/tag/set}{0}
 }
⟨*luatex⟩
\NewSocketPlug{tagsupport/math/luamml/mtable/tag/save}{luamml}
 {
   \__luamml_amsmath_save_tag:
 }
\AssignSocketPlug{tagsupport/math/luamml/mtable/tag/save}{luamml}
\NewSocketPlug{tagsupport/math/luamml/mtable/tag/set}{luamml}
 {
   \__luamml_amsmath_set_tag:
 }
\AssignSocketPlug{tagsupport/math/luamml/mtable/tag/set}{luamml}

⟨/luatex⟩
```

If math structure elements are created the Lbl-structure of a tag must be moved inside the math structure, typically as an additional column in an `mtable` with an intent `:equation-label` or `:no-equation-label`.

The luamml-code handles this by stashing the Lbl-structure, storing the structure number in an array and reusing it once it creates the math structure elements.

This should only be done for specific environments, we define a constant to test:

```
\str_if_exist:cF { l__socket_tagsupport/math/display/tag/begin_plug_str }
 {
   \NewSocket{tagsupport/math/display/tag/begin}{0}
   \NewSocket{tagsupport/math/display/tag/end}{0}
 }
⟨*luatex⟩
\clist_map_inline:nn
  {
    align,
    align*,
    alignat,
    alignat*,
    xalignat,
    xalignat*,
```

there is never a tag/label in xxalignat, so does it make sense to add a label column? Left out for now.

```
    %xxalignat,
    flalign,
    flalign*,
    gather,
    gather*,
```

equation and multline have at most one tag, so we do not use a label column but rely on the external Lbl for now.

```
    %multline, % NO
    %multline*, % NO
    %equation, % NO
    %equation*, % NO
```

split has never a numbering so is ignored

```
    %split, % NO
  }
  {\tl_const:cn { c__luamml_label_#1_tl}{}}
\NewSocketPlug{tagsupport/math/display/tag/begin}{luamml}
 {
   \tag_mc_end:
   \bool_lazy_and:nnTF
    { \tl_if_exist_p:c { c__luamml_label_ \@currenvir _tl } }
    { \int_if_odd_p:n { \int_div_truncate:nn { \l__luamml_flag_int } { 8 } } }
    {
      %\typeout{Stash~and~move~\@currenvir\c_space_tl Lbl}
      \tag_struct_begin:n {tag=Lbl,stash}
      \directlua{table.insert(ltx.__tag.struct.luamml.labels,\tag_get:n{struct_num})}
    }
    {
      \tag_struct_begin:n {tag=Lbl}
```

```
    }
   \tag_mc_begin:n {}
 }
\AssignSocketPlug{tagsupport/math/display/tag/begin}{luamml}
⟨/luatex⟩
```

### B.8.6 Horizontal boxes

This socket annotates an \hbox inside box commands used in math. We test for the socket until the release 2025-06-01.

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/hbox_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/hbox}{2}
   \NewSocketPlug{tagsupport/math/luamml/hbox}{default}{#2}
   \AssignSocketPlug{tagsupport/math/luamml/hbox}{default}
 }
⟨*luatex⟩
\NewSocketPlug{tagsupport/math/luamml/hbox}{luamml}
 {
   \bool_lazy_and:nnTF
    { \mode_if_math_p: }
    { \int_if_odd_p:n { \int_div_truncate:nn { \l__luamml_flag_int } { 8 } } }
    {
      \tag_struct_begin:n
       {
         tag=mtext,
         stash,
       }
      \tag_mc_begin:n {}
      \luamml_annotate:en
       {
         nucleus = true,
         structnum=\tag_get:n{struct_num}
       }
       { #2 }
      \tag_mc_end:
      \tag_struct_end:
    }
    { #2 }
  }
\AssignSocketPlug{tagsupport/math/luamml/hbox}{luamml}
⟨/luatex⟩
```

### B.8.7 Artifact characters

Unicode characters like a root sign should be marked as artifacts to avoid duplication e.g. in derivation if mathml structure elements are used that imply the meaning. We test for the socket until the release 2025-06-01.

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/artifact_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/artifact}{0}
 }
⟨*luatex⟩
```

```
\NewSocketPlug{tagsupport/math/luamml/artifact}{luamml}
 {
   \int_if_odd:nT { \int_div_truncate:nn { \l__luamml_flag_int } { 8 } }
    {
      \tag_mc_begin:n{artifact}
    }
 }
\AssignSocketPlug{tagsupport/math/luamml/artifact}{luamml}
⟨/luatex⟩
```

### B.8.8 Math phantom socket

This socket is used around `\finph@nt`. It should provided until 2025-06-01

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/finph@nt_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/finph@nt}{2}
   \NewSocketPlug{tagsupport/math/luamml/finph@nt}{default}{#2}
   \AssignSocketPlug{tagsupport/math/luamml/finph@nt}{default}
 }
⟨*luatex⟩
\NewSocketPlug{tagsupport/math/luamml/finph@nt}{luamml}
 {
   \luamml_annotate:nen {1}
    {
     nucleus = true,
     core =
       {
        [0] = 'mpadded',
        \ifh@\else
         width = 0,
        \fi
        \ifv@\else
         height = 0, depth = 0,
        \fi
        consume_label'mathphant',
      }
    }
    { #2 }
 }
\AssignSocketPlug{tagsupport/math/luamml/finph@nt}{luamml}
⟨/luatex⟩
```

### B.8.9 Math smash socket

This socket is used around `\finsm@sh`. It should provided until 2025-06-01

```
\str_if_exist:cF { l__socket_tagsupport/math/luamml/finsm@sh_plug_str }
 {
   \NewSocket{tagsupport/math/luamml/finsm@sh}{2}
   \NewSocketPlug{tagsupport/math/luamml/finsm@sh}{default}{#2}
   \AssignSocketPlug{tagsupport/math/luamml/finsm@sh}{default}
 }
```

```
⟨*luatex⟩
\NewSocketPlug{tagsupport/math/luamml/finsm@sh}{luamml}
 {
   \luamml_annotate:nen {2}
    {
     nucleus = true,
     core =
       consume_label('mathsmash',
           function(padded)
             padded.height, padded.depth = 0, 0~
           end),
    }
    { #2 }
 }
\AssignSocketPlug{tagsupport/math/luamml/finsm@sh}{luamml}
⟨/luatex⟩
```

## B.9 Patching

For some packages, we ship with patches to make them more compatible and to demonstrate how other code can be patched to work with `luamml`.

These are either loaded directly if the packages are loaded or delayed using LaTeX's hook system otherwise.

`\__luamml_patch_package:nn`
`\__luamml_patch_package:n`

For this, we use two helpers: First a wrapper which runs arbitrary code either now (if the package is already loaded) or as soon as the package loads, second an application of the first one to load packages following `luamml`'s naming scheme for these patch packages.

```
\cs_new_protected:Npn \__luamml_patch_package:nn #1 #2 {
  \@ifpackageloaded {#1} {#2} {
    \hook_gput_code:nnn {package/#1/after} {luamml} {#2}
  }
}
\cs_new_protected:Npn \__luamml_patch_package:n #1 {
  \__luamml_patch_package:nn {#1} {
    \RequirePackage { luamml-patches-#1 }
  }
}
```

(*End of definition for* `\__luamml_patch_package:nn` *and* `\__luamml_patch_package:n`.)

We currently provide minimal patching for the kernel, amsmath. Currently only the kernel code supports pdfTeX, but it's planned to extend this.

```
\RequirePackage { luamml-patches-kernel }
⟨*luatex⟩
\__luamml_patch_package:n {amstext}
\__luamml_patch_package:n {amsmath}
\__luamml_patch_package:n {mathtools}
⟨/luatex⟩
```

# 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.

## M



## N



## P



## R



## S



## T

## U