

The `lualatex-math` package*

Philipp Stephani
p.stephani2@gmail.com

2012/10/13

Contents

1	Introduction	1
2	Interface	2
3	Implementation of the $\text{\LaTeX} 2_{\epsilon}$ package	2
3.1	Requirements	2
3.2	Messages	3
3.3	Initialization	4
3.4	Patching	4
3.5	$\text{\LaTeX} 2_{\epsilon}$ kernel	5
3.6	<code>amsmath</code>	6
3.7	<code>amsopn</code>	9
3.8	<code>mathtools</code>	10
3.9	<code>icomma</code>	11
4	Implementation of the <code>Lua\text{\LaTeX}</code> module	11
5	Test files	12
5.1	Common definitions	12
5.2	$\text{\LaTeX} 2_{\epsilon}$ kernel, allocation of math families	17
5.3	$\text{\LaTeX} 2_{\epsilon}$ kernel, <code>\mathstyle</code> primitive	17
5.4	<code>amsmath</code> , <code>amsopn</code> , and <code>mathtools</code>	18
5.5	<code>unicode-math</code>	21
5.6	<code>icomma</code> without <code>unicode-math</code>	21
5.7	<code>icomma</code> with <code>unicode-math</code>	22

1 Introduction

`Lua\text{\TeX}` brings major improvements to all areas of `\text{\TeX}` typesetting and programming. They are made available through new primitives or the embedded Lua interpreter, and combining them with existing $\text{\LaTeX} 2_{\epsilon}$ packages is not a task the average \LaTeX user should have to care about. Therefore a multitude of $\text{\LaTeX} 2_{\epsilon}$ packages have been written to bridge the gap between documents and the new features. The `lualatex-math` package focuses on the additional possibilities for mathematical typesetting. The most eminent of the new features is the ability to use Unicode and OpenType fonts, as provided by Will Robertson’s `unicode-math` package. However,

*This document corresponds to `lualatex-math` v1.1, dated 2012/10/13.

there is a smaller group of changes unrelated to Unicode: these are to be dealt with in this package. While in principle most \TeX documents written for traditional engines should work just fine with Lua \TeX , there is a small number of breaking changes that require the attention of package authors. The `lualatex-math` package tries to fix some of the issues encountered while porting traditional macro packages to Lua \TeX .

The decision to write patches for existing macro packages should not be made lightly: monkey patching done by somebody different from the original package author ties the patching package to the implementation details of the patched functionality and breaks all rules of encapsulation. However, due to the lack of alternatives, it has become an accepted way of providing new functionality in \TeX . To keep the negative impact as small as possible, the `lualatex-math` package patches only the $\TeX_{2\epsilon}$ kernel and a small number of popular packages. In general, this package should be regarded as a temporary kludge that should be removed once the math-related packages are updated to be usable with Lua \TeX . By its very nature, the package is likely to cause problems; in such cases, please refer to the issue tracker¹.

2 Interface

The `lualatex-math` package can be loaded with `\usepackage` or `\RequirePackage`, as usual. It has no options and no public interface; the patching is always done when the package is loaded and cannot be controlled. As a matter of course, the `lualatex-math` package needs Lua \TeX to function; it will produce error messages and refuse to load under other engines and formats. The package depends on the `expl3` bundle, the `etoolbox` package, the `luatexbase` bundle and the `filehook` package. The `lualatex-math` package is independent of the `unicode-math` package; the fixes provided here are valid for both Unicode and legacy math typesetting.

Currently patches for the $\TeX_{2\epsilon}$ kernel and the `amsmath`, `amsopn`, `mathtools` and `icomma` packages are provided. It is not relevant whether you load these packages before or after `lualatex-math`. They should work as expected (and ideally you shouldn't notice anything), but if you load other packages that by themselves overwrite commands patched by this package, bad things may happen, as it is usual with \TeX .

One user-visible change is that the new `\mathstyle` primitive (usually called `\luatexmathstyle` in Lua \TeX) should work in all cases after the `lualatex-math` package has been loaded, provided you use the high-level macros `\frac`, `\binom`, and `\genfrac`. The fraction-like \TeX primitives like `\over` or `\atopwithdelims` and the plain \TeX leftovers like `\brack` or `\choose` cannot be patched, and you shouldn't use them.

`\mathstyle`, `\luatexmathstyle`

`\frac`, `\binom`, `\genfrac`

3 Implementation of the $\TeX_{2\epsilon}$ package

3.1 Requirements

```

1 <*package>
2 <@@=lltxmath>
3 \NeedsTeXFormat{LaTeX2e}[2009/09/24]
4 \RequirePackage{expl3}[2012/08/14]
5 \ProvidesExplPackage{lualatex-math}{2012/10/13}{1.1}%
6   {Patches for mathematics typesetting with LuaLaTeX}
7 \RequirePackage{etoolbox}[2007/10/08]
```

¹<https://github.com/phst/lualatex-math/issues>

```

8 \RequirePackage { luatexbase } [ 2010/05/27 ]
9 \RequirePackage { filehook } [ 2011/03/09 ]
10 \RequireLuaModule { lualatex-math } [ 2011/05/05 ]

\@@_restore_catcode:N Executing the exhaustive expansion of \@@_restore_catcode:N⟨character token⟩
restores the category code of the ⟨character token⟩ to its current value.

11 \cs_new_nopar:Npn \@@_restore_catcode:N #1 {
12   \char_set_catcode:nn { \int_eval:n { `#1 } }
13   { \char_value_catcode:n { `#1 } }
14 }

```

We use the macro defined above to restore the category code of the dollar sign. There are packages that make the dollar sign active; hopefully they get loaded after the packages we are trying to patch.

```

15 \exp_args:Nx \AtEndOfPackage {
16   \@@_restore_catcode:N \$
17 }
18 \char_set_catcode_math_toggle:N \$

```

3.2 Messages

luatex-required Issued when not running under LuaTeX.

```

19 \msg_new:nnn { lualatex-math } { luatex-required } {
20   The~ lualatex-math~ package~ requires~ LuaTeX. \\
21   I~ will~ stop~ loading~ now.
22 }

```

different-meanings Issued when two control sequences have different meanings, but should not.

```

23 \msg_new:nnnn { lualatex-math } { different-meanings } {
24   I've~ expected~ the~ control~ sequences \\
25   #1~ and~ #3 \\
26   to~ have~ the~ same~ meaning,~ but~ their~ meanings~ are~ different.
27 } {
28   The~ meaning~ of~ #1~ is: \\
29   #2 \\
30   The~ meaning~ of~ #3~ is: \\
31   #4
32 }

```

macro-expected Issued when trying to patch a non-macro. The first argument must be the detokenized macro name.

```

33 \msg_new:nnn { lualatex-math } { macro-expected } {
34   I've~ expected~ that~ #1~ is~ a~ macro,~ but~ it~ isn't.
35 }

```

wrong-meaning Issued when trying to patch a macro with an unexpected meaning. The first argument must be the detokenized macro name; the second argument must be the actual detokenized meaning; and the third argument must be the expected detokenized meaning.

```

36 \msg_new:nnn { lualatex-math } { wrong-meaning } {
37   I've~ expected~ #1~ to~ have~ the~ meaning \\
38   #3, \\
39   but~ it~ has~ the~ meaning \\
40   #2.
41 }

```

patch-macro Issued when a macro is patched. The first argument must be the detokenized macro name.

```

42 \msg_new:nnn { lualatex-math } { patch-macro } {
43   I'm~ going~ to~ patch~ macro~ #1.
44 }

```

3.3 Initialization

Unless we are running under LuaTeX, we issue an error and quit immediately. Loading the `luatexbase` module will already have produced an error, but we issue another one for clarity.

```

45 \luatex_if_engine:F {
46   \msg_error:nn { lualatex-math } { luatex-required }
47   \endinput
48 }

```

3.4 Patching

\@@_temp:w A scratch macro.

```

49 \cs_new_eq:NN \@@_temp:w \prg_do_nothing:

```

\luatexUmathcode We need the extended versions of `\mathcode` and `\mathchardef`. The command
\luatexUmathcodenum `\luatexbase@ensure@primitive{<name>}` makes sure that the LuaTeX primitive
\luatexUmathchardef `\<name>` is available under the qualified name `\luatex<name>`.

```

50 \luatexbase@ensure@primitive { Umathcode }
51 \luatexbase@ensure@primitive { Umathcodenum }
52 \luatexbase@ensure@primitive { Umathchardef }

```

\@@_assert_eq:NN The macro `\@@_assert_eq:NN<first command><second command>` tests whether the control sequences `<first command>` and `<second command>` have the same meaning, and prints an error message if they do not.

```

53 \cs_new_protected_nopar:Npn \@@_assert_eq:NN #1 #2 {
54   \cs_if_eq:NNF #1 #2 {
55     \msg_error:nnxxxx { lualatex-math } { different-meanings }
56     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
57     { \token_to_str:N #2 } { \token_to_meaning:N #2 }
58   }
59 }

```

\@@_patch:NNnnn The auxiliary macro `\@@_patch:NNnnn<command><factory command><parameter text><expected replacement text><new replacement text>` tries to patch `<command>`. If `<command>` is undefined, do nothing. Otherwise it must be a macro with the given `<parameter text>` and `<expected replacement text>`, created by the given `<factory command>` or equivalent. In this case it will be overwritten using the `<parameter text>` and the `<new replacement text>`. Otherwise issue a warning and don't overwrite.

\@@_patch:cNnnn

```

60 \cs_new_protected_nopar:Npn \@@_patch:NNnnn #1 #2 #3 #4 #5 {
61   \cs_if_exist:NT #1 {
62     \token_if_macro:NTF #1 {
63       \group_begin:
64       #2 \@@_temp:w #3 { #4 }
65       \cs_if_eq:NNTF #1 \@@_temp:w {
66         \msg_info:nnx { lualatex-math } { patch-macro }
67         { \token_to_str:N #1 }
68       \group_end:
69       #2 #1 #3 { #5 }

```

```

70     } {
71     \msg_warning:nnxxx { lualatex-math } { wrong-meaning }
72     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
73     { \token_to_meaning:N \@@_temp:w }
74     \group_end:
75     }
76   } {
77     \msg_warning:nnx { lualatex-math } { macro-expected }
78     { \token_to_str:N #1 }
79   }
80 }
81 }
82 \cs_generate_variant:Nn \@@_patch:NNnnn { c }

```

\@@_set_mathchar:NN The macro \@@_set_mathchar:NN<control sequence>(token) defines the <control sequence> as an extended mathematical character shorthand whose mathematical code is given by the mathematical code of the character ‘<token>’. We cannot use the \Umathcharnumdef primitive here since we would then rely on the \Umathcodenum primitive which is currently broken.²

```

83 \cs_new_protected_nopar:Npn \@@_set_mathchar:NN #1 #2 {
84   \luatexUmathchardef #1
85   \lua_now_x:n {
86     lualatex.math.print_class_fam_slot( \int_eval:n { `#2 } )
87   }
88   \scan_stop:
89 }

```

3.5 L^AT_EX 2_ε kernel

In LuaT_EX, we have 256 math families at our disposal. Therefore we modify the L^AT_EX allocation macros \newfam and \new@mathgroup accordingly.

First we test whether \newfam and \new@mathgroup are equal.

```

90 \@@_assert_eq:NN \newfam \new@mathgroup

```

\new@mathgroup It is enough to modify the maximum number of families known to the allocation system; the macro \alloc@ takes care of the rest. This would work even if the etex package weren’t loaded.

```

91 \@@_patch:NNnnn \new@mathgroup \cs_set_nopar:Npn { } {
92   \@@_=
93   \alloc@ 8 \mathgroup \chardef \sixt@@n
94   \@@_=lltxmath
95 } {
96   \alloc@ 8 \mathgroup \chardef \c_two_hundred_fifty_six
97 }

```

\newfam We have to reset \newfam to equal \new@mathgroup.

```

98 \cs_set_eq:NN \newfam \new@mathgroup

```

LuaT_EX enables access to the current mathematical style via the \mathstyle primitive. For this to work, fraction-like constructs (e.g., <numerator> \over <denominator>) have to be enclosed in a \Ustack group. \frac can be patched to do this, but the plain T_EX remnants \choose, \brack and \brace should be discouraged.

²<http://tug.org/pipermail/luatex/2012-October/003794.html>

`\luatexUstack` First we make sure that we can use the `\Ustack` primitive (under the name `\luatexUstack`).

```
99 \luatexbase@ensure@primitive { Ustack }
```

`\frac` Here we assume that nobody except `amsmath` redefines `\frac`. This is obviously not the case, but we ignore other packages (e.g., `nath`) for the moment. We only patch the \LaTeX 2_ϵ kernel definition if the `amsmath` package is not loaded; the corresponding patch for `amsmath` follows below.

```
100 \AtEndPreamble {
101   \ifpackageloaded { amsmath } { } {
102     \@@_patch:NNnnn \frac \cs_set_nopar:Npn { #1 #2 } {
103       {
104         \begingroup #1 \endgroup \over #2
105       }
106     } {
```

To do: do we need the additional set of braces around `\Ustack`?

```
107     {
108       \luatexUstack { \group_begin: #1 \group_end: \over #2 }
109     }
110   }
111 }
112 }
```

3.6 amsmath

The popular `amsmath` package is subject to three \LaTeX -related problems:

- The `\mathcode` primitive is used several times, which fails for Unicode math characters. `\Umathcode` should be used instead.
- Legacy font dimensions are used for constructing stacks in the `\substack` command and the `subarray` environment. This doesn't work if a Unicode math font is selected.
- The fraction commands `\frac` and `\genfrac` don't use the `\Ustack` primitive.

`\luatexalignmark` We use the primitives corresponding to the alignment mark (`#`) and to the inline math switches; this is more semantical and might lead to better error messages.

```
\luatexUstartmath
\luatexUstopmath
113 \luatexbase@ensure@primitive { alignmark }
114 \luatexbase@ensure@primitive { Ustartmath }
115 \luatexbase@ensure@primitive { Ustopmath }
```

`\luatexUmathstacknumup` Now we require the font parameters we will use.

```
\luatexUmathstackdenomdown
\luatexUmathstackvgap
116 \luatexbase@ensure@primitive { Umathstacknumup }
117 \luatexbase@ensure@primitive { Umathstackdenomdown }
118 \luatexbase@ensure@primitive { Umathstackvgap }
```

`\c_@@_std_minus_mathcode_int` These constants contain the standard \TeX mathematical codes for the minus and
`\c_@@_std_equal_mathcode_int` the equal signs. We temporarily set the math codes to these constants before loading the `amsmath` package so that it can request the legacy math code without error.

```
119 \int_const:Nn \c_@@_std_minus_mathcode_int { "2200 }
120 \int_const:Nn \c_@@_std_equal_mathcode_int { "303D }
```

`\@@_char_dim:NN` The macro `\@@_char_dim:NN<primitive><token>` expands to a $\langle dimen \rangle$ whose value is the metric of the mathematical character corresponding to the character ``<token>`

specified by $\langle primitive \rangle$, which must be one of `\fontcharwd`, `\fontcharht` or `\fontchardp`, in the currently selected text style font.

```
121 \cs_new_nopar:Npn \@@_char_dim:NN #1 #2 {
122   #1 \textfont
123   \lua_now_x:n {
124     luatex.math.print_fam_slot( \int_eval:n { `#2 } )
125   }
126 }
```

`\l_@@_minus_mathchar` These mathematical characters are saved before `amsmath` is loaded so that we
`\l_@@_equal_mathchar` can temporarily assign the \TeX values to the mathematical codes of the minus and equals signs. The `amsmath` package queries these codes, and if they represent Unicode characters, the package loading will fail. If `amsmath` has already been loaded, there is nothing we can do, therefore we use the non-starred version of `\AtBeginOfPackageFile`.

```
127 \chk_if_free_cs:N \l_@@_minus_mathchar
128 \chk_if_free_cs:N \l_@@_equal_mathchar
129 \AtBeginOfPackageFile { amsmath } {
130   \@@_set_mathchar:NN \l_@@_minus_mathchar \-
131   \@@_set_mathchar:NN \l_@@_equal_mathchar \=
}
```

Now we temporarily reset the mathematical codes.

```
132 \char_set_mathcode:nn { `\- } { \c_@@_std_minus_mathcode_int }
133 \char_set_mathcode:nn { `\= } { \c_@@_std_equal_mathcode_int }
134 \AtEndOfPackageFile { amsmath } {
```

`\std@minus` The `amsmath` package defines the control sequences `\std@minus` and `\std@equal`
`\std@equals` as mathematical character shorthands while loading, but uses our restored mathematical codes, which must be fixed.

```
135   \cs_set_eq:NN \std@minus \l_@@_minus_mathchar
136   \cs_set_eq:NN \std@equal \l_@@_equal_mathchar
```

Finally, we restore the original mathematical codes of the two signs.

```
137   \luatexUmathcodenum `\- \l_@@_minus_mathchar
138   \luatexUmathcodenum `\= \l_@@_equal_mathchar
139 }
140 }
```

All of the following fixes work even if `amsmath` is already loaded.

`\@begindocumenthook` `amsmath` repeats the definition of `\std@minus` and `\std@equal` at the beginning of the document, so we also have to patch the internal kernel macro `\@begindocumenthook` which contains the hook code.

```
141 \AtEndOfPackageFile * { amsmath } {
142   \tl_replace_once:Nnn \@begindocumenthook {
143     \mathchardef \std@minus \mathcode `\- \relax
144     \mathchardef \std@equal \mathcode `\= \relax
145   } {
146     \@@_set_mathchar:NN \std@minus \-
147     \@@_set_mathchar:NN \std@equal \=
148   }
```

`\resetMathstrut@` `amsmath` uses the box `\Mathstrutbox@` for struts in mathematical mode. This box is defined to have the height and depth of the opening parenthesis taken from the current text font. The command `\resetMathstrut@` is executed whenever the mathematical fonts are changed and has to restore the correct dimensions. The original definition uses a temporary mathematical character shorthand definition

whose meaning is queried to extract the family and slot. We can do this in Lua; furthermore we can avoid a temporary box because ε -TeX allows us to query glyph metrics directly.

```

149 \@@_patch:NNnnn \resetMathstrut@ \cs_set_nopar:Npn { } {
150   \setbox \z@ \hbox {
151     \mathchardef \@tempa \mathcode `\< \relax % \)
152     \def \@tempb ##1 "##2 ##3 { \the \textfont "##3 \char" }
153     \expandafter \@tempb \meaning \@tempa \relax
154   }
155   \ht \Mathstrutbox@ \ht \z@
156   \dp \Mathstrutbox@ \dp \z@
157 } {
158   \box_set_ht:Nn \Mathstrutbox@ {
159     \@@_char_dim:NN \fontcharht \ ( % \)
160   }
161   \box_set_dp:Nn \Mathstrutbox@ {
162     \@@_char_dim:NN \fontchardp \
163   }
164 }

```

subarray The `subarray` environment uses legacy font dimensions. We simply patch it to use LuaTeX font parameters (and L^AT_EX3 expressions instead of T_EX arithmetic). Since subscript arrays are conceptually vertical stacks, we use the sum of top and bottom shift for the default vertical baseline distance (`\baselineskip`) and the minimum vertical gap for stack for the minimum baseline distance (`\lineskip`).

```

165 \@@_patch:NNnnn \subarray \cs_set:Npn { #1 } {
166   \vcenter
167   \bgroup
168   \Let@
169   \restore@math@cr
170   \default@tag
171   \baselineskip \fontdimen 10~ \scriptfont \tw@
172   \advance \baselineskip \fontdimen 12~ \scriptfont \tw@
173 \@@=>
174   \lineskip \thr@@ \fontdimen 8~ \scriptfont \thr@@
175 \@@=lltxmath>
176   \lineskiplimit \lineskip
177   \ialign
178   \bgroup
179   \ifx c #1 \hfil \fi
180   $ \m@th \scriptstyle ## $
181   \hfil
182   \crrc
183 } {
184   \vcenter
185   \c_group_begin_token
186   \Let@
187   \restore@math@cr
188   \default@tag
189   \skip_set:Nn \baselineskip {
190     \luatexUmathstacknumup \scriptstyle
191     + \luatexUmathstackdenomdown \scriptstyle
192   }
193   \lineskip \luatexUmathstackvgap \scriptstyle
194   \lineskiplimit \lineskip
195   \ialign
196   \c_group_begin_token
197   \token_if_eq_meaning:NNT c #1 { \hfil }

```



```

198 \luatexUstartmath
199 \m@th
200 \scriptstyle
201 \luatexalignmark \luatexalignmark
202 \luatexUstopmath
203 \hfil
204 \crrc
205 }

```

\frac Since \frac is declared by \DeclareRobustCommand, we must patch the macro \frac.

```

206 \@@_patch:cNnnn { frac~ } \cs_set:Npn { #1 #2 } {
207   {
208   \@@=
209   \begingroup #1 \endgroup \@@over #2
210   }
211   } {
212   {
213   \luatexUstack { \group_begin: #1 \group_end: \@@over #2 }
214   \@@=lltxmath)
215   }
216   }

```

\@genfrac Generalized fractions are typeset by the internal \@genfrac command.

```

217 \@@_patch:NNnnn \@genfrac \cs_set_nopar:Npn {
218   #1 #2 #3 #4 #5
219   } {
220   {
221   #1 { \begingroup #4 \endgroup #2 #3 \relax #5 }
222   }
223   } {
224   {
225   #1 {
226   \luatexUstack {
227   \group_begin: #4 \group_end: #2 #3 \scan_stop: #5
228   }
229   }
230   }
231   }
232 }

```

3.7 amsopn

The amsopn package can be used standalone, but is also loaded by amsmath. It provides the \DeclareMathOperator command which breaks when the minus character is a Unicode math character; this issue was brought to my attention by Jean-François Burnol.

\newmcodes@ We only need to patch one usage of \mathcode in the internal macro \newmcodes@, which is called by all user-defined operators.

```

233 \group_begin:
234 \char_set_catcode_other:N \"
235 \AtEndOfPackageFile * { amsopn } {
236   \@@_patch:NNnnn \newmcodes@ \cs_gset_nopar:Npn { } {
237   \mathcode ~\' 39
238   \mathcode ~\'* 42
239   \mathcode ~\' "613A

```

```

240 \ifnum \mathcode `\- = 45 ~ \else
241 \mathchardef \std@minus \mathcode `\- \relax
242 \fi
243 \mathcode `\- 45
244 \mathcode `\/ 47
245 \mathcode `\: "603A \relax
246 } {
247 \char_set_mathcode:nn { `\' } { 39 }
248 \char_set_mathcode:nn { `\'* } { 42 }
249 \char_set_mathcode:nn { `\. } { "613A }
250 \int_compare:nNnF { \luatexUmathcodenum `\- } = { 45 } {
251 \@@_set_mathchar:NN \std@minus \-
252 }
253 \char_set_mathcode:nn { `\- } { 45 }
254 \char_set_mathcode:nn { `\/ } { 47 }
255 \char_set_mathcode:nn { `\: } { "603A }
256 }
257 }
258 \group_end:

```

3.8 mathtools

`mathtools`' `\cramped` command and others that make use of its internal version use a hack involving a null radical. LuaTeX has primitives for setting material in cramped mode, so we make use of them.

```

\luatexcrampeddisplaystyle First we make sure that the needed primitives for cramped styles are available.
\luatexcrampedtextstyle 259 \luatexbase@ensure@primitive { crampeddisplaystyle }
\luatexcrampedscriptstyle 260 \luatexbase@ensure@primitive { crampedtextstyle }
\luatexcrampedscriptscriptstyle 261 \luatexbase@ensure@primitive { crampedscriptstyle }
262 \luatexbase@ensure@primitive { crampedscriptscriptstyle }

\MT_cramped_internal:Nn The macro \MT_cramped_internal:Nn<style>{<expression>} typesets the <expression>
in the cramped style corresponding to the given <style> (\displaystyle etc.);
all we have to do in LuaTeX is to select the correct primitive. Rewriting the user-
level \cramped command and employing \mathstyle would be possible as well, but
we avoid this way since we want to patch only a single command.

263 \AtEndOfPackageFile * { mathtools } {
264 \@@_patch:NNnnn \MT_cramped_internal:Nn
265 \cs_set_nopar:Npn { #1 #2 } {
266 \sbox \z@ {
267 $
268 \m@th
269 #1
270 \nulldelimiterspace = \z@
271 \radical \z@ { #2 }
272 $
273 }
274 \ifx #1 \displaystyle
275 \dimen@ = \fontdimen 8 \textfont 3
276 \advance \dimen@ .25 \fontdimen 5 \textfont 2
277 \else
278 \dimen@ = 1.25 \fontdimen 8
279 \ifx #1 \textstyle
280 \textfont
281 \else
282 \ifx #1 \scriptstyle

```

```

283         \scriptfont
284     \else
285         \scriptscriptfont
286     \fi
287 \fi
288 3
289 \fi
290 \advance \dimen@ -\ht\z@
291 \ht\z@ = -\dimen@
292 \box\z@
293 } {

```

Here the additional set of braces is absolutely necessary, otherwise the changed mathematical style would be applied to the material after the `\mathchoice` construct.

```

294 {
295     \use:c { luatexcramped \cs_to_str:N #1 } #2
296 }
297 }
298 }

```

3.9 icomma

The `icomma` package uses `\mathchardef` to save the mathematical code of the comma character. This breaks for Unicode fonts. The incompatibility was noticed by Peter Breithfeld.³

`\mathcomma` `icomma` defines the mathematical character shorthand `\icomma` at the beginning of the document, therefore we again patch `\@begindocumenthook`.

```

299 \AtEndOfPackageFile * { icomma } {
300     \tl_replace_once:Nnn \@begindocumenthook {
301         \mathchardef \mathcomma \mathcode `\",
302     } {
303         \@@_set_mathchar:NN \mathcomma \,
304     }
305 }
306 \</package>

```

4 Implementation of the Lua_{La}T_EX module

For the Lua module, we use the standard `luatexbase-modutils` template and the `module` function.

```

307 <lua>
308 require("luatexbase.modutils")
309 require("luatexbase.cctb")
310 local err, warn, info, log = luatexbase.provides_module({
311     name = "lualatex-math",
312     date = "2011/05/05",
313     version = 0.1,
314     description = "Patches for mathematics typesetting with LuaLaTeX",
315     author = "Philipp Stephani",
316     licence = "LPPL v1.3+"
317 })
318 local unpack = unpack
319 local string = string

```

³<https://groups.google.com/forum/#!topic/de.comp.text.tex/Cputk-AJS5I/discussion>

```

320 local tex = tex
321 local cctb = luatexbase.catcodetables
322 module("lua $\text{lua}$ tex.math")

print_fam_slot The function print_fam_slot takes one argument which must be a number.
It interprets the argument as a Unicode code point whose mathematical code
is printed in the form  $\langle family \rangle_{\langle slot \rangle}$ , suitable for the right-hand side of e.g.
 $\backslash fontcharht \backslash textfont$ .

323 function print_fam_slot(char)
324   local code = tex.getmathcode(char)
325   local class, family, slot = unpack(code)
326   local result = string.format("%i %i ", family, slot)
327   tex.sprint(cctb.string, result)
328 end

print_class_fam_slot The function print_class_fam_slot takes one argument which must be a number.
It interprets the argument as a Unicode code point whose mathematical code
is printed in the form  $\langle class \rangle_{\langle family \rangle_{\langle slot \rangle}}$ , suitable for the right-hand side of
 $\backslash Umathchardef$ .

329 function print_class_fam_slot(char)
330   local code = tex.getmathcode(char)
331   local class, family, slot = unpack(code)
332   local result = string.format("%i %i %i ", class, family, slot)
333   tex.sprint(cctb.string, result)
334 end
335  $\backslash lua$ 

```

5 Test files

Finally six small test files—but not a real test suite.

5.1 Common definitions

```

336  $\langle *test \rangle$ 
337  $\langle @@=test \rangle$ 
338  $\backslash documentclass [pagesize=auto] \{ scrartcl \}$ 

Only xparse starting with 2008/08/03 has  $\backslash NewDocumentCommand$ .
339  $\backslash usepackage \{ xparse \} [2008/08/03]$ 
340  $\backslash usepackage \{ luacode \}$ 
341  $\backslash ExplSyntaxOn$ 
342  $\backslash AtBeginDocument \{ \backslash errorcontextlines = \backslash c\_fifteen \}$ 

pass This message is issued when a test passed.
343  $\backslash msg\_new : nnn \{ test \} \{ pass \} \{ \#1 \}$ 

 $\backslash @@\_pass : x$  The macro  $\backslash @@\_pass : x \{ \langle text \rangle \}$  issues the pass message with description  $\langle text \rangle$ .
344  $\backslash cs\_new\_protected\_nopar : Npn \backslash @@\_pass : x \#1 \{$ 
345    $\backslash msg\_info : nnx \{ test \} \{ pass \} \{ \#1 \}$ 
346  $\}$ 

fail This message is issued when a test failed.
347  $\backslash msg\_new : nnn \{ test \} \{ fail \} \{ \#1 \}$ 

 $\backslash @@\_fail : x$  The macro  $\backslash @@\_fail : x \{ \langle text \rangle \}$  issues the fail message with description  $\langle text \rangle$ .
348  $\backslash cs\_new\_protected\_nopar : Npn \backslash @@\_fail : x \#1 \{$ 
349    $\backslash msg\_error : nnx \{ test \} \{ fail \} \{ \#1 \}$ 
350  $\}$ 

```

`\tl_const:Nx` We need expanding constants.

```

351 \cs_generate_variant:Nn \tl_const:Nn { Nx }

```

`\c_@@_equal_tl` Two shorthands for pretty-printing test results.

```

\c_@@_not_equal_tl 352 \tl_const:Nx \c_@@_equal_tl { \c_space_tl == \c_space_tl }
353 \tl_const:Nx \c_@@_not_equal_tl { \c_space_tl != \c_space_tl }

```

`\@@_equal_pass:nxn` The macro `\@@_equal_pass:nxn{<first expression>}{<first value>}{<second expression>}{<second value>}` is called when the two values arising from the two expressions are equal.

```

354 \cs_new_protected_nopar:Npn \@@_equal_pass:nxn #1 #2 #3 #4 {
355   \@@_pass:x {
356     \exp_not:n { #1 }
357     \c_@@_equal_tl
358     #2
359     \c_@@_equal_tl
360     #4
361     \c_@@_equal_tl
362     \exp_not:n { #3 }
363   }
364 }

```

`\@@_equal_fail:nxn` The macro `\@@_equal_pass:nxn{<first expression>}{<first value>}{<second expression>}{<second value>}` is called when the two values arising from the two expressions are not equal.

```

365 \cs_new_protected_nopar:Npn \@@_equal_fail:nxn #1 #2 #3 #4 {
366   \@@_fail:x {
367     \exp_not:n { #1 }
368     \c_@@_equal_tl
369     #2
370     \c_@@_not_equal_tl
371     #4
372     \c_@@_equal_tl
373     \exp_not:n { #3 }
374   }
375 }

```

`\@@_assert_equal:NNNNNnn` The macro `\@@_assert_equal:NNNNNnn<set command><use command><compare command><first temporary command><second temporary command>{<first expression>}{<second expression>}` asserts that the two expressions are equal. The `<set command>` must have the argument specification `Nn`, the `<use command>` `N`, and the `<compare command>` `nNnTF`.

`\@@_assert_equal:ccccnn`

```

376 \cs_new_protected_nopar:Npn
377 \@@_assert_equal:NNNNNnn #1 #2 #3 #4 #5 #6 #7 {
378   #1 #4 { #6 }
379   #1 #5 { #7 }
380   #3 { #4 } = { #5 } {
381     \@@_equal_pass:nxn { #6 } { #2 #4 } { #7 } { #2 #5 }
382   } {
383     \@@_equal_fail:nxn { #6 } { #2 #4 } { #7 } { #2 #5 }
384   }
385 }
386 \cs_generate_variant:Nn \@@_assert_equal:NNNNNnn { ccccc }

```

`\@@_assert_equal:nnn` The macro `\@@_assert_equal:nnn{<data type>}{<first expression>}{<second expression>}` is a simplified version of `\@@_assert_equal:NNNNNnn` for data types following the L^AT_EX3 naming conventions; `<data type>` must be `int`, `dim`, etc.

```

387 \cs_new_protected_nopar:Npn \@@_assert_equal:nnn #1 #2 #3 {
388   \@@_assert_equal:cccccnn
389   { #1 _set:Nn } { #1 _use:N } { #1 _compare:nNnTF }
390   { l_@@_tmpa_ #1 } { l_@@_tmpb_ #1 } { #2 } { #3 }
391 }

\l_@@_tmpa_int Scratch registers for numbers.
\l_@@_tmpb_int
392 \int_new:N \l_@@_tmpa_int
393 \int_new:N \l_@@_tmpb_int

\AssertIntEqual The command \AssertIntEqual{<first expression>}{<second expression>} asserts
that the two integral expressions are equal.
394 \NewDocumentCommand \AssertIntEqual { m m } {
395   \@@_assert_equal:nnn { int } { #1 } { #2 }
396 }

\l_@@_tmpa_int Scratch registers for dimensions.
\l_@@_tmpb_int
397 \dim_new:N \l_@@_tmpa_dim
398 \dim_new:N \l_@@_tmpb_dim

\AssertDimEqual The command \AssertDimEqual{<first expression>}{<second expression>} asserts
that the two dimension expressions are equal.
399 \NewDocumentCommand \AssertDimEqual { m m } {
400   \@@_assert_equal:nnn { dim } { #1 } { #2 }
401 }

\AssertMathStyle The command \AssertMathStyle{<expression>} asserts that the current mathe-
matical style is equal to the value of the integral <expression>.
402 \NewDocumentCommand \AssertMathStyle { m } {
403   \AssertIntEqual { \luatexmathstyle } { #1 }
404 }

\@@_assert_cramped:Nx The macro \@@_assert_cramped:Nn<predicate>{<name>} asserts that we are in
math mode and that the current style fulfills the <predicate> (identified by the
<name>) which must have the argument specification n.
405 \cs_new_protected_nopar:Npn \@@_assert_cramped:Nx #1 #2 {
406   \int_set:Nn \l_@@_tmpa_int { \luatexmathstyle }
407   \bool_if:nTF {
408     \int_compare_p:nNn { \l_@@_tmpa_int } > { \c_minus_one }
409     &&
410     #1 { \l_@@_tmpa_int }
411   } {
412     \@@_pass:x {
413       \exp_not:N \luatexmathstyle
414       \c_@@_equal_tl
415       \int_use:N \l_@@_tmpa_int
416       \c_space_tl
417       is~ a~ #2~ style
418     }
419   } {
420     \@@_fail:x {
421       \exp_not:N \luatexmathstyle
422       \c_@@_equal_tl
423       \int_use:N \l_@@_tmpa_int
424       \c_space_tl
425       is~ not~ a~ #2~ style
426     }
427   }
428 }

```

`\AssertNoncrampedStyle` The command `\AssertNoncrampedStyle` asserts that the current mathematical style is one of the non-cramped styles.

```

429 \NewDocumentCommand \AssertNoncrampedStyle { } {
430   \@@_assert_cramped:Nx \int_if_even_p:n { non-cramped }
431 }

```

`\AssertCrampedStyle` The command `\AssertCrampedStyle` asserts that the current mathematical style is one of the cramped styles.

```

432 \NewDocumentCommand \AssertCrampedStyle { } {
433   \@@_assert_cramped:Nx \int_if_odd_p:n { cramped }
434 }

```

`\l_@@_tmpa_box` Scratch registers for box constructions.

`\l_@@_tmpb_box`

```

435 \box_new:N \l_@@_tmpa_box
436 \box_new:N \l_@@_tmpb_box

```

`contains_space` The function `contains_space(head, width)` returns `true` if the node list starting at `head` or any of its sublists contain a glue or kern node of width `width`. If `width` is `nil`, returns `true` if there is any glue or kern node. If `width` is the string `"nonzero"`, returns `true` if there is any glue node or kern node of nonzero width.

```

437 \begin{luacode*}
438 function contains_space(head, width)
439   for n in node.traverse(head) do
440     local id = n.id
441     if id == 10 then -- glue node
442       if width then
443         if width == "nonzero" or n.spec.width == width then
444           return true
445         end
446       end
447     elseif id == 11 then -- kern node
448       if width then
449         if width == "nonzero" then
450           if n.kern ~= 0 then
451             return true
452           end
453         elseif n.kern == width then
454           return true
455         end
456       end
457     elseif id == 0 or id == 1 then -- sublist
458       if contains_space(n.head, width) then
459         return true
460       end
461     end
462   end
463   return false
464 end
465 \end{luacode*}

```

`\AssertNoSpace` The command `\AssertNoSpace{<text>}` asserts that the node list that is the result of typesetting `<text>` contains no glue or kern nodes. When called with a star, the command ignores zero-width kerns.

```

466 \NewDocumentCommand \AssertNoSpace { s m } {
467   \hbox_set:Nn \l_@@_tmpa_box { #2 }
468   \int_if_odd:nTF {
469     \lua_now_x:n {

```

```

470     local~ b = tex.getbox(\int_use:N \l_@@_tmpa_box)
471     if~ contains_space(b.head,
472       \IfBooleanTF { #1 } { "nonzero" } { nil }) then~
473       tex.sprint("0")
474     else~
475       tex.sprint("1")
476     end
477   }
478 } {
479   \@@_pass:x {
480     \tl_to_str:n { #2 } ~
481     contains~ no~ skip~ or~ kern~ node
482   }
483 } {
484   \@@_fail:x {
485     \tl_to_str:n { #2 } ~
486     contains~ a~ skip~ or~ kern~ node
487   }
488 }
489 }

```

`\AssertMuSpace` The command `\AssertMuSpace{<text>}{<muskip>}` asserts that the node list that is the result of typesetting `<text>` contains at least one glue or kern node of with `<muskip>`.

```

490 \makeatletter
491 \NewDocumentCommand \AssertMuSpace { m m } {
492   \hbox_set:Nn \l_@@_tmpa_box { #1 }
493   \hbox_set:Nn \l_@@_tmpb_box { $ \mskip #2 \m@th $ }
494   \int_if_odd:nTF {
495     \lua_now_x:n {
496       local~ b = tex.getbox(\int_use:N \l_@@_tmpa_box)
497       local~ s = tex.getbox(\int_use:N \l_@@_tmpb_box)
498       if~ contains_space(b.head, s.width) then~
499         tex.sprint("1")
500       else~
501         tex.sprint("0")
502       end
503     }
504   } {
505     \@@_pass:x {
506       \tl_to_str:n { #1 } ~
507       contains~ a~ skip~ or~ kern~ node~ of~ width~
508       \tl_to_str:n { #2 }
509     }
510   } {
511     \@@_fail:x {
512       \tl_to_str:n { #1 } ~
513       contains~ no~ skip~ or~ kern~ node~ of~ width~
514       \tl_to_str:n { #2 }
515     }
516   }
517 }
518 \makeatother
519 \ExplSyntaxOff
520 </test>

```


5.2 L^AT_EX 2_ε kernel, allocation of math families

The L^AT_EX 2_ε kernel itself allocates four families (also known as “math groups” in L^AT_EX parlance). Therefore we should still be able to allocate 252 families. We do this alternately with `\newfam`, `\new@mathgroup` and `\DeclareSymbolFont`.

```

521 (*test-kernel-alloc)
522 \usepackage{lualatex-math}
523 \makeatletter
524 \ExplSyntaxOn
525 \prg_stepwise_inline:nnnn { \c_four } { \c_one } {
526   \c_two_hundred_fifty_five - \c_one
527 } {
528   \prg_case_int:nnn { \int_mod:nn { #1 } { \c_three } } {
529     { \c_zero } {
530       \chk_if_free_cs:N \g_@@_family_int
531       \newfam \g_@@_family_int
532       \AssertIntEqual { \g_@@_family_int } { #1 }
533       \cs_undefine:N \g_@@_family_int
534     }
535     { \c_one } {
536       \chk_if_free_cs:N \g_@@_mathgroup_int
537       \new@mathgroup \g_@@_mathgroup_int
538       \AssertIntEqual { \g_@@_mathgroup_int } { #1 }
539       \cs_undefine:N \g_@@_mathgroup_int
540     }
541     { \c_two } {
542       \DeclareSymbolFont { Test #1 } { OT1 } { cmr } { m } { n }
543       \exp_args:Nc \AssertIntEqual { sym Test #1 } { #1 }
544     }
545   } {
546     \@@_fail:x { This~ cannot~ happen }
547   }
548 }
549 \DeclareSymbolFont { Test 255 } { OT1 } { cmr } { bx } { it }
550 \DeclareSymbolFontAlphabet { \TestAlphabet } { Test 255 }
551 \exp_args:Nc \AssertIntEqual { sym Test 255 }
552 { \c_two_hundred_fifty_five }
553 \ExplSyntaxOff
554 \makeatother
555 \begin{document}
556 \[
557 \TestAlphabet{
558   abc
559   \AssertIntEqual{\fam}{255}
560   \AssertIntEqual{\mathgroup}{255}
561 }
562 \]
563 \end{document}
564 /test-kernel-alloc)

```

5.3 L^AT_EX 2_ε kernel, `\mathstyle` primitive

Here we only check whether different fractions and other style-changing commands result in the correct mathematical style.

```

565 (*test-kernel-style)
566 \usepackage{lualatex-math}
567 \begin{document}
568 \begin{displaymath}

```

```

569 \AssertMathStyle{0} \sqrt{\AssertMathStyle{1}}
570 \frac{\AssertMathStyle{2}}{\AssertMathStyle{3}}
571 a^{\frac{\AssertMathStyle{6}}{\AssertMathStyle{7}}}
572 \sqrt{\frac{\AssertMathStyle{3}}{\AssertMathStyle{3}}}
573 \displaystyle
574 \frac{\AssertMathStyle{2}}{\AssertMathStyle{3}}
575 \luatexcrampeddisplaystyle
576 \frac{\AssertMathStyle{3}}{\AssertMathStyle{3}}
577 \textstyle
578 \frac{\AssertMathStyle{4}}{\AssertMathStyle{5}}
579 \luatexcrampedtextstyle
580 \frac{\AssertMathStyle{5}}{\AssertMathStyle{5}}
581 \scriptstyle
582 \frac{\AssertMathStyle{6}}{\AssertMathStyle{7}}
583 \luatexcrampedscriptstyle
584 \frac{\AssertMathStyle{7}}{\AssertMathStyle{7}}
585 \end{displaymath}
586 \begin{math}
587 \AssertMathStyle{2} \sqrt{\AssertMathStyle{3}}
588 \frac{\AssertMathStyle{4}}{\AssertMathStyle{5}}
589 a^{\frac{\AssertMathStyle{6}}{\AssertMathStyle{7}}}
590 \sqrt{\frac{\AssertMathStyle{5}}{\AssertMathStyle{5}}}
591 \displaystyle
592 \frac{\AssertMathStyle{2}}{\AssertMathStyle{3}}
593 \luatexcrampeddisplaystyle
594 \frac{\AssertMathStyle{3}}{\AssertMathStyle{3}}
595 \textstyle
596 \frac{\AssertMathStyle{4}}{\AssertMathStyle{5}}
597 \luatexcrampedtextstyle
598 \frac{\AssertMathStyle{5}}{\AssertMathStyle{5}}
599 \scriptstyle
600 \frac{\AssertMathStyle{6}}{\AssertMathStyle{7}}
601 \luatexcrampedscriptstyle
602 \frac{\AssertMathStyle{7}}{\AssertMathStyle{7}}
603 \end{math}
604 \end{document}
605 \end{test-kernel-style}

```

5.4 amsmath, amsopn, and mathtools

Since mathtools loads amsmath and amsopn anyway, we test all three in one file.

\testbox First a scratch box register.

```

606 \newtestbox{test-amsmath}
607 \usepackage{luatex-math}
608 \newsavebox{\testbox}

```

We set the mathematical code for the minus sign to some arbitrary Unicode value to test whether the load-time patch works.

```

609 \luatexUmathcode`\-="2 "33 "44444 \relax
610 \usepackage{amsmath}
611 \AssertIntEqual{\luatexUmathcode`\-}{33444444}
612 \makeatletter
613 \AssertIntEqual{\std@minus}{33444444}
614 \makeatother

```

Check that we can still declare operators.

```

615 \DeclareMathOperator{\Operator}{*-/ 'a-b}
616 \DeclareMathOperator*{\OperatorWithLimits}{01'*/-/}

```

```

617 \DeclareMathOperator{\OperatorWithPunctuation}{a:b*/'-.}
618 \usepackage{mathtools}

```

The same for the document begin hook.

```

619 \luatexUmathcode`\="5 "66 "77777 \relax
620 \begin{document}
621 \AssertIntEqual{\luatexUmathcode`\="}{66A77777}
622 \makeatletter
623 \AssertIntEqual{\std@equal}{66A77777}
624 \makeatother

```

Here we test whether the strut box has the correct height and depth.

```

625 \sbox{\testbox}{$(\$ \% )}
626 \makeatletter
627 \AssertDimEqual{\ht\Mathstrutbox@}{\ht\testbox}
628 \AssertDimEqual{\dp\Mathstrutbox@}{\dp\testbox}
629 \makeatother

```

Here we test for the various amsmath features that have to be patched: sub-arrays and various kind of fraction-like objects. The `\substack` command and `subarray` environment aren't really tested since it is hard to check whether the outcome looks right in an automated way. All tests are done in both inline and display mode.

```

630 \begin{equation*}
631   \AssertMathStyle{0} \sqrt{\AssertMathStyle{1}}
632   \sum_{
633     \substack{\frac{1}{2} \ \ \frac{3}{4} \ \ \frac{5}{6}}
634   }
635   \sum_{
636     \begin{subarray}{l} \frac{1}{2} \ \ \frac{3}{4} \ \ \frac{5}{6} \end{subarray}
637   }
638   \frac{\AssertMathStyle{2}}{\AssertMathStyle{3}}
639   a^{\frac{\AssertMathStyle{6}}{\AssertMathStyle{7}}}
640   \dfrac{\AssertMathStyle{2}}{\AssertMathStyle{3}}
641   \tfrac{\AssertMathStyle{4}}{\AssertMathStyle{5}}
642   \binom{\AssertMathStyle{2}}{\AssertMathStyle{3}}
643   a^{\binom{\AssertMathStyle{6}}{\AssertMathStyle{7}}}
644   \dbinom{\AssertMathStyle{2}}{\AssertMathStyle{3}}
645   \tbinom{\AssertMathStyle{4}}{\AssertMathStyle{5}}
646   \genfrac{}{}{}{\AssertMathStyle{2}}{\AssertMathStyle{3}}
647   \genfrac{<}{/}{0pt}{0}{\AssertMathStyle{2}}{\AssertMathStyle{3}}
648   \genfrac{}{}{}{1}{\AssertMathStyle{4}}{\AssertMathStyle{5}}
649   \genfrac{|}{|}{4pt}{2}{\AssertMathStyle{6}}{\AssertMathStyle{7}}
650   \genfrac{}{}{}{3}{\AssertMathStyle{6}}{\AssertMathStyle{7}}
651 \end{equation*}
652 \begin{math}
653   \AssertMathStyle{2} \sqrt{\AssertMathStyle{3}}
654   \sum_{
655     \substack{\frac{1}{2} \ \ \frac{3}{4} \ \ \frac{5}{6}}
656   }
657   \sum_{
658     \begin{subarray}{l} \frac{1}{2} \ \ \frac{3}{4} \ \ \frac{5}{6} \end{subarray}
659   }
660   \frac{\AssertMathStyle{4}}{\AssertMathStyle{5}}
661   a^{\frac{\AssertMathStyle{6}}{\AssertMathStyle{7}}}
662   \dfrac{\AssertMathStyle{2}}{\AssertMathStyle{3}}
663   \tfrac{\AssertMathStyle{4}}{\AssertMathStyle{5}}
664   \binom{\AssertMathStyle{4}}{\AssertMathStyle{5}}
665   a^{\binom{\AssertMathStyle{6}}{\AssertMathStyle{7}}}
666   \dbinom{\AssertMathStyle{2}}{\AssertMathStyle{3}}

```

```

667 \tbinom{\AssertMathStyle{4}}{\AssertMathStyle{5}}
668 \genfrac{}{}{}{\AssertMathStyle{4}}{\AssertMathStyle{5}}
669 \genfrac{<}{/}{0pt}{0}{\AssertMathStyle{2}}{\AssertMathStyle{3}}
670 \genfrac{}{}{}{1}{\AssertMathStyle{4}}{\AssertMathStyle{5}}
671 \genfrac{|}{|}{4pt}{2}{\AssertMathStyle{6}}{\AssertMathStyle{7}}
672 \genfrac{}{}{}{3}{\AssertMathStyle{6}}{\AssertMathStyle{7}}
673 \end{math}

```

Since `mathtools' \cramped` command uses `\mathchoice`, we cannot test for a single mathematical style since all of them are executed; instead, we just verify that all styles encountered are cramped.

```

674 \begin{equation*}
675 \AssertMathStyle{0}
676 a^{\AssertMathStyle{4} a}
677 \cramped{\AssertCrampedStyle a^{\AssertCrampedStyle a}}
678 a^{
679 \AssertMathStyle{4}
680 a^a
681 \cramped{\AssertCrampedStyle a^{\AssertCrampedStyle a}}
682 a^a
683 \AssertMathStyle{4}
684 }
685 a^{
686 a^{
687 \AssertMathStyle{6}
688 a^a
689 \cramped{\AssertCrampedStyle a^{\AssertCrampedStyle a}}
690 a^a
691 \AssertMathStyle{6}
692 }
693 }
694 a^{\AssertMathStyle{4} a}
695 \AssertMathStyle{0}
696 \end{equation*}
697 \begin{math}
698 \AssertMathStyle{2}
699 a^{\AssertMathStyle{4} a}
700 \cramped{\AssertCrampedStyle a^{\AssertCrampedStyle a}}
701 a^{
702 \AssertMathStyle{4}
703 a^a
704 \cramped{\AssertCrampedStyle a^{\AssertCrampedStyle a}}
705 a^a
706 \AssertMathStyle{4}
707 }
708 a^{
709 a^{
710 \AssertMathStyle{6}
711 a^a
712 \cramped{\AssertCrampedStyle a^{\AssertCrampedStyle a}}
713 a^a
714 \AssertMathStyle{6}
715 }
716 }
717 a^{\AssertMathStyle{4} a}
718 \AssertMathStyle{2}
719 \end{math}

```

The `amsopn` package uses `\mathcode` when executing a user-defined operator command. Test that this was patched out.

```

720 \AssertNoSpace*{$\Operator$}
721 \AssertNoSpace*{$\OperatorWithLimits$}
722 \AssertMuSpace*{$\OperatorWithPunctuation$}{\thinmuskip}
723 \mathcode`\-=45 \relax
724 \AssertNoSpace*{$\Operator$}
725 \AssertNoSpace*{$\OperatorWithLimits$}
726 \AssertMuSpace*{$\OperatorWithPunctuation$}{\thinmuskip}
727 \end{document}
728 </test-amsmath>

```

5.5 unicode-math

This test file loads both `amsmath` and `unicode-math`. The latter package contains fixes that somewhat overlap with ours. We have to take care in all packages that no attempt is made to patch a single macro twice. Therefore we treat warnings (that occur when trying to patch a macro with an unknown meaning) as errors here. However, the auxiliary package `fontspec-patches` uses `\RenewDocumentCommand` from the `xparse` package, which generates a warning that we don't want to turn into an error. Therefore we treat the offending message `redefine-command` specially.

```

729 <*test-unicode>
730 \ExplSyntaxOn
731 \msg_redirect_class:nn { warning } { error }
732 \msg_redirect_name:nnn { LaTeX } { xparse / redefine-command } { info }
733 \ExplSyntaxOff
734 \usepackage{amsmath}
735 \usepackage{unicode-math}[2011/05/05]
736 \setmathfont{XITS Math}
737 \usepackage{lualatex-math}
738 \begin{document}
739 \begin{equation*}
740 \AssertMathStyle{0} \sqrt{\AssertMathStyle{1}}
741 \frac{\AssertMathStyle{2}}{\AssertMathStyle{3}}
742 a^{\frac{\AssertMathStyle{6}}{\AssertMathStyle{7}}}
743 \dfrac{\AssertMathStyle{2}}{\AssertMathStyle{3}}
744 \tfrac{\AssertMathStyle{4}}{\AssertMathStyle{5}}
745 \end{equation*}
746 \end{document}
747 </test-unicode>

```

5.6 icomma without unicode-math

This test file loads only `icomma` to test whether our patch works for Computer Modern.

```

748 <*test-icomma>
749 \usepackage{lualatex-math}
750 \usepackage{icomma}
751 \begin{document}
752 $1,234 \; (x, y)$
753 \AssertNoSpace{$1,234$}
754 \AssertMuSpace*{$(x, y)$}{\thinmuskip}
755 \AssertIntEqual{\mathcomma}{"1C0003B}
756 \end{document}
757 </test-icomma>

```

5.7 icomma with unicode-math

This test file loads both icomma and unicode-math to test whether they interact well.

```

758 \test-icomma-unicode
759 \usepackage{unicode-math}[2011/05/05]
760 \setmathfont{XITS Math}
761 \usepackage{lualatex-math}
762 \usepackage{icomma}
763 \begin{document}
764 $1,234 \; (x, y)$
765 \AssertNoSpace{$1,234$}
766 \AssertMuSpace{(x, y)}{\thinmuskip}
767 \AssertIntEqual{\mathcomma}{"0C0002C}
768 \end{document}
769 \test-icomma-unicode

```

Change History

v0.1	
General: Initial version	1
v0.2	
General: Added patch for the icomma package	11
Added test file for icomma with unicode-math	22
Added test file for icomma without unicode-math	21
v0.3	
General: Added test file for modified family allocation scheme	17
Patched math group allocation to gain access to all families	5
v0.3a	
General: Updated for changes in l3kernel	1
v0.3b	
\@begindocumenthook: Another update for a change in l3kernel	7
v0.3c	
\@@_char_dim:NN: l3kernel renamed \lua_now:x to \lua_now_x:n	7
\@@_set_mathchar:NN: l3kernel renamed \lua_now:x to \lua_now_x:n	5
General: Added special treatment for <code>redefine-command</code> warning	21
\AssertMuSpace: l3kernel renamed \lua_now:x to \lua_now_x:n	16
\AssertNoSpace: l3kernel renamed \lua_now:x to \lua_now_x:n	15
v1.0	
General: Switched to l3docstrip	1
v1.1	
\@@_set_mathchar:NN: Update reasoning why \Umathcharnumdef is not used here	5
General: Add fix and unit test for <code>amsopn</code>	9, 18
\AssertNoSpace: Allow testing for nonzero kern nodes	15
contains_space: Allow testing for nonzero kern nodes	15

Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

Symbols

\"	234
\\$	16, 18

\'	237, 247
\(151, 159
\)	151, 159, 162
*	238, 248
\,	301, 303
\-	130, 132, 137, 143, 146, 240, 241, 243, 250, 251, 253, 609, 611, 723
\.	239, 249
\/	244, 254
\:	245, 255
\;	752, 764
\=	131, 133, 138, 144, 147, 619, 621
\@@_assert_cramped:Nx	405, 430, 433
\@@_assert_eq:NN	53, 90
\@@_assert_equal:NNNNnn	376
\@@_assert_equal:cccccm	376, 388
\@@_assert_equal:nmn	387, 395, 400
\@@_char_dim:NN	121, 159, 162
\@@_equal_fail:nxnx	365, 383
\@@_equal_pass:nxnx	354, 381
\@@_fail:x	348, 366, 420, 484, 511, 546
\@@_pass:x	344, 355, 412, 479, 505
\@@_patch:NNnn	60, 91, 102, 149, 165, 217, 236, 264
\@@_patch:cNnn	60, 206
\@@_restore_catcode:N	11, 16
\@@_set_mathchar:NN	83, 130, 131, 146, 147, 251, 303
\@@_temp:w	49, 64, 65, 73
\@@over	209, 213
\@begindocumenthook	141, 300
\@genfrac	217
\@ifpackageloaded	101
\@tempa	151, 153
\@tempb	152, 153
\[556
\]	20, 24, 25, 28, 29, 30, 37, 38, 39, 633, 636, 655, 658
\]	562

A

\advance	172, 276, 290
\alloc@	93, 96
amsmath (package)	1, 2, 6, 7, 9, 18, 19, 21
amsopn (package)	1, 2, 9, 18, 21, 22
\AssertCrampedStyle	432, 677, 681, 689, 700, 704, 712
\AssertDimEqual	399, 627, 628
\AssertIntEqual	394, 403, 532, 538, 543, 551, 559, 560, 611, 613, 621, 623, 755, 767
\AssertMathStyle	402, 569, 570, 571, 572, 574, 576, 578, 580, 582, 584, 587, 588, 589, 590, 592, 594, 596, 598, 600, 602, 631, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 653, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 675, 676, 679, 683, 687, 691, 694, 695, 698, 699, 702, 706, 710, 714, 717, 718, 740, 741, 742, 743, 744
\AssertMuSpace	490, 722, 726, 754, 766
\AssertNoncrampedStyle	429
\AssertNoSpace	466, 720, 721, 724, 725, 753, 765
\AtBeginDocument	342
\AtBeginOfPackageFile	129
\AtEndOfPackage	15
\AtEndOfPackageFile	134, 141, 235, 263, 299
\AtEndPreamble	100

B

<code>\baselineskip</code>	171, 172, 189
<code>\begin</code>	437, 555, 567, 568, 586, 620, 630, 636, 652, 658, 674, 697, 738, 739, 751, 763
<code>\begingroup</code>	104, 209, 221
<code>\bgroup</code>	167, 178
<code>\binom</code>	2, 642, 643, 664, 665
<code>\bool_if:nTF</code>	407
<code>\box</code>	292
<code>\box_new:N</code>	435, 436
<code>\box_set_dp:Nn</code>	161
<code>\box_set_ht:Nn</code>	158
Breitfeld, Peter	11
Burnol, Jean-François	9

C

<code>\c_@@_equal_tl</code>	352, 357, 359, 361, 368, 372, 414, 422
<code>\c_@@_not_equal_tl</code>	352, 370
<code>\c_@@_std_equal_mathcode_int</code>	119, 133
<code>\c_@@_std_minus_mathcode_int</code>	119, 132
<code>\c_fifteen</code>	342
<code>\c_four</code>	525
<code>\c_group_begin_token</code>	185, 196
<code>\c_minus_one</code>	408
<code>\c_one</code>	525, 526, 535
<code>\c_space_tl</code>	352, 353, 416, 424
<code>\c_three</code>	528
<code>\c_two</code>	541
<code>\c_two_hundred_fifty_five</code>	526, 552
<code>\c_two_hundred_fifty_six</code>	96
<code>\c_zero</code>	529
<code>\char</code>	152
<code>\char_set_catcode:n</code>	12
<code>\char_set_catcode_math_toggle:N</code>	18
<code>\char_set_catcode_other:N</code>	234
<code>\char_set_mathcode:n</code>	132, 133, 247, 248, 249, 253, 254, 255
<code>\char_value_catcode:n</code>	13
<code>\chardef</code>	93, 96
<code>\chk_if_free_cs:N</code>	127, 128, 530, 536
<code>contains_space (function)</code>	15, 437
<code>\cramped</code>	677, 681, 689, 700, 704, 712
<code>\crr</code>	182, 204
<code>\cs_generate_variant:Nn</code>	82, 351, 386
<code>\cs_gset_nopar:Npn</code>	236
<code>\cs_if_eq:NMF</code>	54
<code>\cs_if_eq:NNTF</code>	65
<code>\cs_if_exist:NT</code>	61
<code>\cs_new_eq:NN</code>	49
<code>\cs_new_nopar:Npn</code>	11, 121
<code>\cs_new_protected_nopar:Npn</code>	53, 60, 83, 344, 348, 354, 365, 376, 387, 405
<code>\cs_set:Npn</code>	165, 206
<code>\cs_set_eq:NN</code>	98, 135, 136
<code>\cs_set_nopar:Npn</code>	91, 102, 149, 217, 265
<code>\cs_to_str:N</code>	295
<code>\cs_undefine:N</code>	533, 539

D

<code>\dbinom</code>	644, 666
<code>\DeclareMathOperator</code>	615, 616, 617
<code>\DeclareSymbolFont</code>	542, 549

<code>\DeclareSymbolFontAlphabet</code>	550
<code>\def</code>	152
<code>\default@tag</code>	170, 188
<code>\dfrac</code>	640, 662, 743
different-meanings (message)	23
<code>\dim_new:N</code>	397, 398
<code>\dimen@</code>	275, 276, 278, 290, 291
<code>\displaystyle</code>	274, 573, 591
<code>\documentclass</code>	338
<code>\dp</code>	156, 628

E

<code>\else</code>	240, 277, 281, 284
<code>\end</code>	465, 563, 585, 603, 604, 636, 651, 658, 673, 696, 719, 727, 745, 746, 756, 768
<code>\endgroup</code>	104, 209, 221
<code>\endinput</code>	47
environments:	
<code>subarray</code>	165
<code>\errorcontextlines</code>	342
<code>etex</code> (package)	5
<code>etoolbox</code> (package)	2
<code>\exp_args:Nc</code>	543, 551
<code>\exp_args:Nx</code>	15
<code>\exp_not:N</code>	413, 421
<code>\exp_not:n</code>	356, 362, 367, 373
<code>\expandafter</code>	153
<code>expl3</code> (package)	2
<code>\ExplSyntaxOff</code>	519, 553, 733
<code>\ExplSyntaxOn</code>	341, 524, 730

F

fail (message)	12, 347
<code>\fam</code>	559
<code>\fi</code>	179, 242, 286, 287, 289
<code>filehook</code> (package)	2
<code>\fontchardp</code>	162
<code>\fontcharht</code>	159
<code>\fontdimen</code>	171, 172, 174, 275, 276, 278
<code>fontspec-patches</code> (package)	21
<code>\frac</code>	2, 100, 206, 570, 571, 572, 574, 576, 578, 580, 582, 584, 588, 589, 590, 592, 594, 596, 598, 600, 602, 633, 636, 638, 639, 655, 658, 660, 661, 741, 742
functions:	
<code>contains_space</code>	15, 437
<code>module</code>	11
<code>print_class_fam_slot</code>	12, 329
<code>print_fam_slot</code>	12, 323

G

<code>\g_@_family_int</code>	530, 531, 532, 533
<code>\g_@_mathgroup_int</code>	536, 537, 538, 539
<code>\genfrac</code>	2, 646, 647, 648, 649, 650, 668, 669, 670, 671, 672
<code>\group_begin:</code>	63, 108, 213, 227, 233
<code>\group_end:</code>	68, 74, 108, 213, 227, 258

H

<code>\hbox</code>	150
<code>\hbox_set:Nn</code>	467, 492, 493
<code>\hfil</code>	179, 181, 197, 203
<code>\ht</code>	155, 290, 291, 627

I

<code>\ialign</code>	177, 195
icomma (package)	1, 2, 11, 21, 22
<code>\IfBooleanTF</code>	472
<code>\ifnum</code>	240
<code>\ifx</code>	179, 274, 279, 282
<code>\int_compare:nNf</code>	250
<code>\int_compare_p:nNn</code>	408
<code>\int_const:Nn</code>	119, 120
<code>\int_eval:n</code>	12, 86, 124
<code>\int_if_even_p:n</code>	430
<code>\int_if_odd:nTF</code>	468, 494
<code>\int_if_odd_p:n</code>	433
<code>\int_mod:nn</code>	528
<code>\int_new:N</code>	392, 393
<code>\int_set:Nn</code>	406
<code>\int_use:N</code>	415, 423, 470, 496, 497

L

l3docstrip (package)	22
l3kernel (package)	22
<code>\l_@_equal_mathchar</code>	127, 136, 138
<code>\l_@_minus_mathchar</code>	127, 135, 137
<code>\l_@_tmpa_box</code>	435, 467, 470, 492, 496
<code>\l_@_tmpa_dim</code>	397
<code>\l_@_tmpa_int</code>	392, 397, 406, 408, 410, 415, 423
<code>\l_@_tmpb_box</code>	435, 493, 497
<code>\l_@_tmpb_dim</code>	398
<code>\l_@_tmpb_int</code>	392, 397
<code>\Let@</code>	168, 186
<code>\lineskip</code>	174, 176, 193, 194
<code>\lineskiplimit</code>	176, 194
<code>\lua_now_x:n</code>	85, 123, 469, 495
luatex-required (message)	19
<code>\luatex_if_engine:F</code>	45
<code>\luatexalignmark</code>	113, 201
luatexbase (package)	2, 4
luatexbase-modutils (package)	11
<code>\luatexbase@ensure@primitive</code> ..	50, 51, 52, 99, 113, 114, 115, 116, 117, 118, 259, 260, 261, 262
<code>\luatexcrampeddisplaystyle</code>	259, 575, 593
<code>\luatexcrampedscriptscriptstyle</code>	259
<code>\luatexcrampedscriptstyle</code>	259, 583, 601
<code>\luatexcrampedtextstyle</code>	259, 579, 597
<code>\luatexmathstyle</code>	2, 403, 406, 413, 421
<code>\luatexUmathchardef</code>	50, 84
<code>\luatexUmathcode</code>	50, 609, 611, 619, 621
<code>\luatexUmathcodenum</code>	50, 137, 138, 250
<code>\luatexUmathstackdenomdown</code>	116, 191
<code>\luatexUmathstacknumup</code>	116, 190
<code>\luatexUmathstackvgap</code>	116, 193
<code>\luatexUstack</code>	99, 108, 213, 226
<code>\luatexUstartmath</code>	113, 198
<code>\luatexUstopmath</code>	113, 202

M

<code>\m@th</code>	180, 199, 268, 493
macro-expected (message)	33
<code>\makeatletter</code>	490, 523, 612, 622, 626
<code>\makeatother</code>	518, 554, 614, 624, 629

<code>\mathchardef</code>	143, 144, 151, 241, 301
<code>\mathcode</code>	143, 144, 151, 237, 238, 239, 240, 241, 243, 244, 245, 301, 723
<code>\mathcomma</code>	299, 755, 767
<code>\mathgroup</code>	93, 96, 560
<code>\Mathstrutbox@</code>	155, 156, 158, 161, 627, 628
<code>\mathstyle</code>	2
mathtools (package)	1, 2, 10, 18, 20
<code>\meaning</code>	153
messages:	
different-meanings	23
fail	12, 347
luatex-required	19
macro-expected	33
pass	12, 343
patch-macro	42
redefine-command	21, 22
wrong-meaning	36
module (function)	11
<code>\msg_error:nn</code>	46
<code>\msg_error:nnx</code>	349
<code>\msg_error:nnxxx</code>	55
<code>\msg_info:nnx</code>	66, 345
<code>\msg_new:nnn</code>	19, 33, 36, 42, 343, 347
<code>\msg_new:nnnn</code>	23
<code>\msg_redirect_class:nn</code>	731
<code>\msg_redirect_name:nnn</code>	732
<code>\msg_warning:nnx</code>	77
<code>\msg_warning:nnxxx</code>	71
<code>\mskip</code>	493
<code>\MT_cramped_internal:Nn</code>	263

N

nath (package)	6
<code>\NeedsTeXFormat</code>	3
<code>\new@mathgroup</code>	90, 91, 98, 537
<code>\NewDocumentCommand</code>	394, 399, 402, 429, 432, 466, 491
<code>\newfam</code>	90, 98, 531
<code>\newmcodes@</code>	233
<code>\newsavebox</code>	608
<code>\nulldelimiterspace</code>	270

O

<code>\Operator</code>	615, 720, 724
<code>\OperatorWithLimits</code>	616, 721, 725
<code>\OperatorWithPunctuation</code>	617, 722, 726
<code>\over</code>	104, 108

P

packages:	
amsmath	1, 2, 6, 7, 9, 18, 19, 21
amsopn	1, 2, 9, 18, 21, 22
etex	5
etoolbox	2
expl3	2
filehook	2
fontspec-patches	21
icomma	1, 2, 11, 21, 22
l3docstrip	22
l3kernel	22

luatexbase	2, 4
luatexbase-modutils	11
mathtools	1, 2, 10, 18, 20
nath	6
unicode-math	1, 2, 21, 22
xparse	12, 21
pass (message)	12, 343
patch-macro (message)	42
\prg_case_int:nnn	528
\prg_do_nothing:	49
\prg_stepwise_inline:nnnn	525
print_class_fam_slot (function)	12, 329
print_fam_slot (function)	12, 323
\ProvidesExplPackage	5

R

\radical	271
redefine-command (message)	21, 22
\relax	143, 144, 151, 153, 221, 241, 245, 609, 619, 723
\RequireLuaModule	10
\RequirePackage	4, 7, 8, 9
\resetMathstrut@	149
\restore@math@cr	169, 187
Robertson, Will	1

S

\sbox	266, 625
\scan_stop:	88, 227
\scriptfont	171, 172, 174, 283
\scriptscriptfont	285
\scriptstyle	180, 190, 191, 193, 200, 282, 581, 599
\setbox	150
\setmathfont	736, 760
\sixt@@n	93
\skip_set:Nn	189
\sqrt	569, 572, 587, 590, 631, 653, 740
\std@equal	136, 144, 147, 623
\std@equals	135
\std@minus	135, 143, 146, 241, 251, 613
\subarray	165
subarray (environment)	165
\substack	633, 655
\sum_	632, 635, 654, 657

T

\tbinom	645, 667
\TestAlphabet	550, 557
\testbox	606, 625, 627, 628
\textfont	122, 152, 275, 276, 280
\textstyle	279, 577, 595
\tfrac	641, 663, 744
\the	152
\thinmuskip	722, 726, 754, 766
\thr@@	174
\tl_const:Nn	351
\tl_const:Nx	351, 352, 353
\tl_replace_once:Nnn	142, 300
\tl_to_str:n	480, 485, 506, 508, 512, 514
\token_if_eq_meaning:NNT	197

<code>\token_if_macro:NTF</code>	62
<code>\token_to_meaning:N</code>	56, 57, 72, 73
<code>\token_to_str:N</code>	56, 57, 67, 72, 78
<code>\tw@</code>	171, 172

U

<code>unicode-math</code> (package)	1, 2, 21, 22
<code>\use:c</code>	295
<code>\usepackage</code>	339, 340, 522, 566, 607, 610, 618, 734, 735, 737, 749, 750, 759, 761, 762

V

<code>\vcenter</code>	166, 184
-----------------------------	----------

W

<code>wrong-meaning</code> (message)	36
--	--------------------

X

<code>xparse</code> (package)	12, 21
-------------------------------------	--------

Z

<code>\z@</code>	150, 155, 156, 266, 270, 271, 290, 291, 292
------------------------	---