DATATYPES

additional datatypes in lmtx

context 2020 meeting
Native \TeX\ datatypes: simple registers

integer: \count123 = 456 \the\count123

integer: 456

dimension: \dimen123 = 456pt \the\dimen123

dimension: 456.0pt

glue: \skip123 = 6pt plus 5pt minus 4pt\relax \the\skip123

glue: 6.0pt plus 5.0pt minus 4.0pt

muglue: \muskip123 = 6mu plus 5mu minus 4mu\relax \the\muskip123

muglue: 6.0mu plus 5.0mu minus 4.0mu

attribute: \attribute123 = 456 \the\attribute123

attribute: 456

\global \the \countdef \dimendef \skipdef \muskipdef \attributedef
\advance \multiply \divide \numexpr \dimexpr \glueexpr \muexpr
Native \TeX\ datatypes: tokens

toks: \toks123 = \{456\} \the\toks123

toks: 456

\global \the \toksdef
\toksapp \etoksapp \xtoksapp \gtoksapp
\tokspre \etokspre \xtokspre \gtokspre

(in retrospect: eetex)
Native \TeX datatypes: boxes

\texttt{box}: \texttt{\box123 = \hbox \{456\} (\the\wd123,\the\ht123,\the\dp123)} \texttt{\box123}

\texttt{box}: = 456 (0.0pt,0.0pt,0.0pt)

\texttt{\global \box \copy \unhbox \unvbox \hbox \vbox \vtop \hpack \vpack \tpack \wd \ht \dp \boxtotal \boxdirection \boxattr \boxorientation \boxxoffset \boxyoffset \boxxmove \boxymove}
Native \TeX datatypes: macros

\def\onetwothree{346} \onetwothree
346

\global \protected \frozen
\def \edef \edef \xdef
\meaning
Native Lua datatypes: numbers

```lua
local n = 123 context(n)}\quad
local n = 123.456 context(n)}\quad
local n = 123.4E56 context(n)}\quad
local n = 0x123 context(n)}\quad
local n = 0x1.37fe4cd4b70b2p-1 context(n)}
```

```
123   123.456   1.234e+58   291   0.60936203095073
```

```
+   -   *   /   //   %   ^   |   ~   &   <<   >>   ==   ~==   <   >   <=   >=   (   )
```
Native Lua datatypes: strings

\ctxlua{local s = "abc" context(s)}
\ctxlua{local s = 'abc' context(s)}
\ctxlua{local s = [[abc]] context(s)}
\ctxlua{local s = [[==[abc]==]] context(s)}

abc abc abc abc abc

.. # == ~= < > <= >=
Native Lua datatypes: booleans and nil

```lua
local b = true  context(b)
local b = false context(b)
local n = nil   context(n)

== ~= and or not
```
Native Lua datatypes: some more

functions
userdata (lpeg is userdata)
coroutine

LuaMetaTEX provides tokens and nodes as userdata and some libraries also use them (complex, decimal, pdf, etc).
Both worlds combined

- There are only 64K registers (although we can extend that if needed).
- Accessing registers at the Lua end is not that efficient.
- So we have now datatypes at the Lua end with access at the \text{T\LaTeX} end.
- Their values can go beyond what \text{T\LaTeX} registers provide.

\texttt{\textbackslash luacardinal bar 123}
\texttt{\textbackslash luainteger bar -456}
\texttt{\textbackslash luafloat bar 123.456E-3}
\texttt{\textbackslash the\textbackslash luacardinal bar \textbackslash quad}
\texttt{\textbackslash the\textbackslash luainteger bar \textbackslash quad}
\texttt{\textbackslash the\textbackslash luafloat bar}

123 -456 0.1234559999999999629718416827017790637910366058349609375
The usual Lua semantics apply:

```
\luacardinal  bar  0x123
\luainteger  bar  -0x456
\luafloat     bar  0x123.456p-3
```

So, now we get:

```
291  -1110  36.40887451171875
```

Equal signs are optional:

```
\luainteger  gnu=  123456  \luafloat  gnu=  123.456e12
\luainteger  gnu = 123456  \luafloat  gnu =  123.456e12
\luainteger  gnu =123456  \luafloat  gnu  =123.456e12
```

These commands can be used for assignments as well as serialization. They use the LuaMeta\TeX\ value function feature.
Dimensions are serialized differently so that they can be used like this:

\luadimen test 100pt \scratchdimen = .25 \luadimen test: \the\scratchdimen
0.0pt
Assume that we have this:

\texttt{\textcolor{blue}{\textbf{\textbackslash luacardinal}}} \texttt{x = -123} \quad \texttt{\textcolor{blue}{\textbf{\textbackslash luafloat}}} \texttt{x = 123.123}
\texttt{\textcolor{blue}{\textbf{\textbackslash luacardinal}}} \texttt{y = 456} \quad \texttt{\textcolor{blue}{\textbf{\textbackslash luafloat}}} \texttt{y = -456.456}

We can then use the macro \texttt{\textcolor{blue}{\textbf{\textbackslash luaexpression}}} that takes an optional keyword:

- \texttt{\textcolor{blue}{\textbf{\textbackslash luaexpression}}} \texttt{- \{n.x + 2*n.y\}}
- \texttt{\textcolor{blue}{\textbf{\textbackslash luaexpression}}} \texttt{float \{n.x + 2*n.y\}}
- \texttt{\textcolor{blue}{\textbf{\textbackslash luaexpression}}} \texttt{integer \{n.x + 2*n.y\}}
- \texttt{\textcolor{blue}{\textbf{\textbackslash luaexpression}}} \texttt{cardinal \{n.x + 2*n.y\}}
- \texttt{\textcolor{blue}{\textbf{\textbackslash luaexpression}}} \texttt{boolean \{n.x + 2*n.y\}}
- \texttt{\textcolor{blue}{\textbf{\textbackslash luaexpression}}} \texttt{lua \{n.x + 2*n.y\}}

The serialization can be different for these cases:

- \texttt{- : -789.789}
- \texttt{f : -789.788999999999987267074175179004669189453125}
- \texttt{i : -790}
- \texttt{c : 790}
- \texttt{b : 1}
- \texttt{l : -0x1.8ae4fdf3b645ap+9}

Variables have their own namespace but get resolved across namespaces (f, i, c).
Special tricks:

```latex
\scratchdimen 123.456pt [\the\scratchdimen] [\the\nodimen\scratchdimen]

[123.456pt][123.456pt]

Does nothing, nor does:

\nodimen\scratchdimen = 654.321pt

But:

\the \nodimen bp \scratchdimen 651.876462bp
\the \nodimen cc \scratchdimen 50.959168cc
\the \nodimen cm \scratchdimen 22.996753cm
\the \nodimen dd \scratchdimen 611.510013dd
\the \nodimen in \scratchdimen 9.05384in
\the \nodimen mm \scratchdimen 229.96753mm
\the \nodimen pt \scratchdimen 654.320999pt
\the \nodimen sp \scratchdimen 42881581sp
```

gives different units! In the coffee break it was decided to drop the nc and nd units in LuaMeta\TeX when Arthur indicated that they never became a standard. Dropping the true variants also makes sense but we postponed dropping the in (inch).
Arrays

Two dimensional arrays have names and a type:

\newarray name integers type integer nx 2 ny 2
\newarray name booleans type boolean nx 2 ny 2
\newarray name floats type float nx 2 ny 2
\newarray name dimensions type dimension nx 4

And a special accessor. Here we set values:

\arrayvalue integers 1 2 4 \arrayvalue integers 2 1 8
\arrayvalue booleans 1 2 true \arrayvalue booleans 2 1 true
\arrayvalue floats 1 2 12.34 \arrayvalue floats 2 1 34.12
\arrayvalue dimensions 1 12.34pt \arrayvalue dimensions 3 34.12pt
Here we get values:

- integers: [1, 2]
- booleans: [1, 2]
- floats: [1, 2]
- dimensions: [1]

- integers: [2, 1]
- booleans: [2, 1]
- floats: [2, 1]
- dimensions: [3]

When a value is expected the integer is serialized:

- integers: [1, 2]
- floats: [12.34, 34.12]

You can view an array on the console with:

- integers: showarray integers
Another expression example:

```
\dostepwiserecurse {1} {4} {1} {
    \the\arrayvalue dimensions #1 :
    \luaexpression dimen {math.sind(30) * a.dimensions[#1]}
}
```

[12.34pt: 6.17pt] [0.0pt: 0pt] [34.12pt: 17.06pt] [0.0pt: 0pt]
We can combine it all with if tests:

slot 1 is \ifboolean\arrayequals dimensions 1 0pt zero \else not zero \fi\quad slot 2 is \ifboolean\arrayequals dimensions 2 0pt zero \else not zero \fi

slot 1 is not zero slot 2 is zero

slot 1: \ifcase\arraycompare dimensions 1 3pt lt \or eq \else gt \fi zero\quad slot 2: \ifcase\arraycompare dimensions 2 3pt lt \or eq \else gt \fi zero\quad slot 3: \ifcase\arraycompare dimensions 3 3pt lt \or eq \else gt \fi zero\quad slot 4: \ifcase\arraycompare dimensions 4 3pt lt \or eq \else gt \fi zero

slot 1: \ifcmpdim\arrayvalue dimensions 1 3pt lt \or eq \else gt \fi zero\quad slot 2: \ifcmpdim\arrayvalue dimensions 2 3pt lt \or eq \else gt \fi zero\quad slot 3: \ifcmpdim\arrayvalue dimensions 3 3pt lt \or eq \else gt \fi zero\quad slot 4: \ifcmpdim\arrayvalue dimensions 4 3pt lt \or eq \else gt \fi zero

slot 1: gt zero slot 2: lt zero slot 3: gt zero slot 4: lt zero

slot 1: gt zero slot 2: lt zero slot 3: gt zero slot 4: lt zero
Complex numbers

```lua
local c1 = xcomplex.new(1,3)
local c2 = xcomplex.new(2,4)
context(c1) context.quad() context(c2) context.quad(c1 + c2)
```

1.0+3.0i  2.0+4.0i  3.0+7.0i
Decimal numbers

\startlua
local c1 = xdecimal.new("123456789012345678901234567890")
local c2 = xdecimal.new(1234567890)
context(c1) context.crlf() context(c2) context.crlf(c1 * c2)
\stoplua

123456789012345678901234567890
1234567890
152415787517146788751714678875019052100