

DATATYPES

additional datatypes in lmtx

context 2020 meeting

Native T_EX datatypes: simple registers

```
integer: \count 123 = 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
```

2 \advance \multiply \divide \numexpr \dimexpr \glueexpr \muexpr

Native T_EX datatypes: tokens

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

```
toks: 456
```

```
1 \global \the \toksdef
```

```
2 \toksapp \etoksapp \xtoksapp \gtoksapp
```

```
3 \tokspre \etokspre \xtokspre \gtokspre
```

(in retrospect: eetex)

Native T_EX datatypes: boxes

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

```
box: = 456 (0.0pt,0.0pt,0.0pt)
```

```
1 \global \box \copy \unhbox \unvbox
```

```
2 \hbox \vbox \vtop \hpack \vpack \tpack
```

```
3 \wd \ht \dp \boxtotal
```

```
4 \boxdirection \boxattr
```

```
5 \boxorientation \boxxoffset \boxyoffset \boxxmove \boxymove
```

Native T_EX datatypes: macros

```
1 \def\onetwothree{346} \onetwothree
```

```
346
```

```
1 \global \protected \frozen
```

```
2 \def \edef \edef \xdef
```

```
3 \meaning
```

Native Lua datatypes: numbers

```
1 \ctxlua{local n = 123           context(n)}\quad  
2 \ctxlua{local n = 123.456     context(n)}\quad  
3 \ctxlua{local n = 123.4E56    context(n)}\quad  
4 \ctxlua{local n = 0x123       context(n)}\quad  
5 \ctxlua{local n = 0x1.37fe4cd4b70b2p-1 context(n)}
```

123 123.456 1.234e+58 291 0.60936203095073

```
1 + - * / // % ^ | ~ & << >> == ~= < > <= >= ( )
```

Native Lua datatypes: strings

```
1 \ctxlua{local s = "abc"      context(s)}\quad  
2 \ctxlua{local s = 'abc'     context(s)}\quad  
3 \ctxlua{local s = [[abc]]   context(s)}\quad  
4 \ctxlua{local s = [=[abc]=] context(s)}\quad  
  
abc abc abc abc
```

```
1 .. # == ~= < > <= >=
```


Native Lua datatypes: booleans and nil

```
1 \ctxlua{local b = true  context(b)}\quad
```

```
2 \ctxlua{local b = false context(b)}\quad
```

```
3 \ctxlua{local n = nil   context(n)}\quad
```

```
1 == ~= and or not
```

Native Lua datatypes: some more

1 functions

2 userdata (lpeg is userdata)

3 coroutine

LuaMetaT_EX 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 T_EX end.
- Their values can go beyond what T_EX registers provide.

```
1 \luacardinal bar 123
```

```
2 \luainteger bar -456
```

```
3 \luafloat bar 123.456E-3
```

```
1 \the\luacardinal bar \quad
```

```
2 \the\luainteger bar \quad
```

```
3 \the\luafloat bar
```

```
123 -456 0.123455999999999999629718416827017790637910366058349609375
```

The usual Lua semantics apply:

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

So, now we get:

```
291 -1110 36.40887451171875
```

Equal signs are optional:

```
1 \luainteger gnu= 123456 \luafloat gnu= 123.456e12
2 \luainteger gnu = 123456 \luafloat gnu = 123.456e12
3 \luainteger gnu =123456 \luafloat gnu =123.456e12
```

These commands can be used for assignments as well as serialization. They use the LuaMetaTeX 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:

```
1 \luacardinal x = -123    \luafloat x = 123.123
2 \luacardinal y = 456    \luafloat y = -456.456
```

We can then use the macro `\luaexpression` that takes an optional keyword:

```
1 - : \luaexpression      {n.x + 2*n.y}
2 f : \luaexpression float {n.x + 2*n.y}
3 i : \luaexpression integer {n.x + 2*n.y}
4 c : \luaexpression cardinal {n.x + 2*n.y}
5 b : \luaexpression boolean {n.x + 2*n.y}
6 l : \luaexpression lua    {n.x + 2*n.y}
```

The serialization can be different for these cases:

```
- : -789.789
f : -789.7889999999999987267074175179004669189453125
i : -790
c : 790
b : 1
l : -0x1.8ae4fdf3b645ap+9
```

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

Special tricks:

```
\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 LuaMetaTeX 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:

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

And a special accessor. Here we set values:

```
1 \arrayvalue integers 1 2 4 \arrayvalue integers 2 1 8
2 \arrayvalue booleans 1 2 true \arrayvalue booleans 2 1 true
3 \arrayvalue floats 1 2 12.34 \arrayvalue floats 2 1 34.12
4 \arrayvalue dimensions 1 12.34pt \arrayvalue dimensions 3 34.12pt
```


Here we get values:

```
1 [\the\arrayvalue integers 1 2]
2 [\the\arrayvalue booleans 1 2]
3 [\the\arrayvalue floats 1 2]
4 [\the\arrayvalue dimensions 1 ]\crlf
5 [\the\arrayvalue integers 2 1]
6 [\the\arrayvalue booleans 2 1]
7 [\the\arrayvalue floats 2 1]
8 [\the\arrayvalue dimensions 3]
```

```
[4][1][12.33999999999999857891452847979962825775146484375][12.34pt]
```

```
[8][1][34.11999999999999744204615126363933086395263671875][34.12pt]
```

When a value is expected the integer is serialized:

```
1 \scratchcounter\arrayvalue integers 1 2\relax \the\scratchcounter
4
```

You can view an array on the console with:

```
1 \showarray integers
```

Another expression example:

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

[12.34pt: 6.17pt] [0.0pt: 0pt] [34.12pt: 17.06pt] [0.0pt: 0pt]

We can combine it all with if tests:

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

slot 1 is not zero slot 2 is zero

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

```
5 slot 1: \ifcmpdim\arrayvalue dimensions 1 3pt lt \or eq \else gt \fi zero\quad  
6 slot 2: \ifcmpdim\arrayvalue dimensions 2 3pt lt \or eq \else gt \fi zero\quad  
7 slot 3: \ifcmpdim\arrayvalue dimensions 3 3pt lt \or eq \else gt \fi zero\quad  
8 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

```
1 \startluacode
2 local c1 = xcomplex.new(1,3)
3 local c2 = xcomplex.new(2,4)
4 context(c1) context.quad() context(c2) context.quad(c1 + c2)
5 \stopluacode

1.0+3.0i  2.0+4.0i  3.0+7.0i
```

Decimal numbers

```
1 \startluacode  
2 local c1 = xdecimal.new("123456789012345678901234567890")  
3 local c2 = xdecimal.new(1234567890)  
4 context(c1) context.crlf() context(c2) context.crlf(c1 * c2)  
5 \stopluacode
```

123456789012345678901234567890

1234567890

152415787517146788751714678875019052100