As part of the ConTEXt lmtx project, we experiment with new ways to interface subsys-
tems and MetaPost makes a good candidate. Here I wrap up some experiments. It’s 
not a manual or a recipe, just food for thought. All you see here might change and/or 
evolve.
As we move to Con\TeX lmtx, some decisions need to be made. For instance, how shall we introduce new functionality while not affecting the MkIV code base too much with engine related tests, messy and performance sapping. Of course we remain downward compatible but the Lua\Meta\TeX engine sometimes permits multiple solutions. As this is something new, for a while we can use experimental features that may change or later disappear depending on experience. It is not a big deal to adapt to these new possibilities and users shouldn’t notice, unless of course they currently employ (yet undocumented) low-level features. Part of the solution is to have separate files for MkIV and lmtx, although that will mostly concern low level support modules; even then differences will be subtle and should not affect users. I must admit that some of those changes are mostly aesthetic and often don’t improve performance, although better performance sometimes may be the case, in particular concerning ‘extreme’ usage.

But, that said, there are areas where lmtx will offer entirely new functionality. One of the areas where we are making interesting developments is in MetaFun, which means MetaPost. Now, it is not to be said that some of the core functionality needed for this will migrate from Lua\Meta\TeX (used by lmtx) to Lua\TeX (used by MkIV). But, because (due to external pressure), the later engine is now considered stable and its functionality frozen, it is not likely to be quickly backported or at least only will happen when this has been thoroughly tested in Con\TeX/lmtx.

Over the years there have been (small) improvements in the interface but while Alan Braslau and I experiment with new graphic abilities, we also are coming up with (graphic) interfaces that suit Con\TeX much better. Below, I will show some examples.\footnote{These optional argument handlers themselves are not part of MetaPost but implemented in Con\TeX using some experimental parsing related extensions to a version of mplib used in Lua\Meta\TeX.}

\begin{verbatim}
\startMPcode{doublefun}
  draw lmt_grid [ 
    nx = 20, ny = 10, 
    dx = 5, dy = 2, 
    tx = 10, ty = 10, 
  ] sized 5cm withpen pencircle scaled 1mm withcolor "red" ;
\stopMPcode
\end{verbatim}
What we see here is a key/value driven interface that has been one of the unique core properties of Con\TeXt since it was created. Of course parameters can be optional as demonstrated in:

\startMPcode{doublefun}
  draw lmt_grid [
    nx = 10, ny = 10,
    dx = 1, dy = 1
  ] sized 3cm
  withpen pencircle scaled 1mm
  withcolor "darkblue" ;

  draw lmt_grid [
    nx = 10, ny = 10
  ] sized 3cm shifted (4cm,0)
  withpen pencircle scaled .3mm
  withcolor "darkgreen" ;
\stopMPcode

This syntax means that we can also use the already used interface documentation system.

We can actually go quite far in what we provide. It’s not so much that these things were not possible before, but the more concise user interface is something new indeed. Let’s start with drawing some simple axis:

\startMPcode{doublefun}
  draw lmt_axis [
    nx = 20, ny = 10,
    dx = 5, dy = 2,
  ] sized 4cm
  withpen pencircle scaled 1mm
  withcolor "darkcyan" ;
\stopMPcode

\footnote{Note the use of the color string "red" rather than the MetaPost color variable \texttt{red := (1,0,0)}. This is a small detail but here imports the color definition from Con\TeXt, which might be a bit special.}
This ‘article’ is not meant as tutorial but more as a teaser about what is possible and hopefully it triggers users in coming up with useful demands. As the previous axis example is not that interesting we move on to a more complex one:

\texttt{\begin{verbatim}
\startMPcode{doublefun}
  \draw lmt_axis [ \\
    sx = 5mm, sy = 5mm, \\
    nx = 20, ny = 10, \\
    dx = 5, dy = 2, \\
    tx = 10, ty = 10, \\
  ] withpen pencircle scaled 1mm withcolor "darkblue" ; \\
  \draw lmt_connected ( \\
    for i = 1 step 2 until 20 :  \\
      -- (i, 1 randomized 8) * 5mm \\
    endfor \\
  ) \\
  withpen pencircle scaled 2mm withcolor "darkyellow" ; \\
  \draw lmt_connected ( \\
    for i = 1 step 2 until 20 :  \\
      lmt_connection (i, 1 randomized 8) * 5mm \\
    endfor \\
  ) \\
  withpen pencircle scaled 2mm withcolor "darkgreen" ; \\
\stopMPcode
\end{verbatim}}
These examples are made as part of exploring how to best interface with MetaPost, to be used by the upcoming luagraph module, a flexible and high performance set of macros aimed at drawing graphs and to visualize data. On the one hand, we want to free the user from dealing with details such as scales to get the graphics right, but on the other hand it should be possible to fine-tune when needed.

Again we draw an axis. Using strings as color names, mentioned in the footnote earlier, is not new, but already present for a while. It will use the colors defined at the \TeX end and is more convenient and natural than using the older \MPcolor. The \texttt{lmt\_connected} macro is sort of a gimmick: if you look closely you will see that it has to start with a point (coordinate) but that one is discarded silently. Again this is something we need to deal with.

The next variant uses a list of samples. Watch the use of braces, this is something normally not part of MetaPost syntax, and it signals a Lua-like list. This is a choice of interfacing. There are subtle and somewhat complicated aspects involved here as MetaPost parsing involves primaries, secondaries, tertiaries and expressions, symbols, strings, tags and other specialities.

\startMPcode{doublefun}
\begin{verbatim}
draw lmt_axis [
  sx  = 5mm, sy  = 5mm,
  nx  = 20, ny  = 10,
  dx  = 5, dy  = 2,
  tx  = 5, ty  = 2,
  startx = 1, starty = 0,
  connect = true,
  samples = {
    4, 2, 4, 9, 3, 0, 8, 9, 10, 1,
}\end{verbatim}
\end{MPcode}
We can have multiple sample lists. But of course dealing with that is really up to the underlying code, which itself is not that much related to MetaPost but has been delegated to Lua.

\definecolor[tdarkred] [r=.6,a=1,t=.5]
\definecolor[tdarkgreen] [g=.6,a=1,t=.5]
\definecolor[tdarkblue] [b=.6,a=1,t=.5]
\startMPcode{doublefun}
  save u ; let u = uniformdeviate ;
  draw lmt_axis [ 
    sx = 5mm, sy = 5mm,
    nx = 20, ny = 10,
    dx = 5, dy = 2,
    tx = 10, ty = 2,
    samples = { 
      
  u10, u10, u10, u10, u10,
  u10, u10, u10, u10, u10,
  ] withpen pencircle scaled 1mm ;
\stopMPcode
New interfacing in MetaFun, a preview

\begin{verbatim}
\begin{MPpicture}(10,10)
\draw [dashed] (0,0) grid (10,10);
\pause \end{MPpicture}
\end{verbatim}

This Way
September 11, 2019
#1104 LMTX
Yet another interesting variant might be this:

\startluacode
\documentdata["lmx-sample-1"] = {
    {4, 2, 4, 9, 3, 0, 8, 9, 10, 1,
     5, 3, 0, 8, 5, 6, 10, 2, 3, 5}
} \stopluacode
\startMPcode{doublefun}
draw lmt_axis [
    sx = 5\,\text{mm}, sy = 5\,\text{mm},
    nx = 20, ny = 10,
    dx = 5, dy = 2,
    tx = 5, ty = 2,
    startx = 1, starty = 0,
    samples = "lmx-sample-1",
    axiscolor = "darkgreen",
    textcolor = "darkred",
    samplecolors = { "darkblue" },
] withpen pencircle scaled 1\,\text{mm} ; \stopMPcode

Adding more tricks is not that complex, take, for example:

\startMPcode{doublefun}
draw lmt_axis [ 
    sx = 5mm, sy = 5mm,
    nx = 20, ny = 10,
    dx = 5, dy = 2,
    tx = 10, ty = 10,
    
    list = { 
        [ 
            connect = true,
            color = "darkred",
            close = true,
            points = { (1, 1), (15, 8), (2, 10) },
            texts = { "segment 1", "segment 2", "segment 3" } 
        ],
        [ 
            connect = true,
            color = "darkgreen",
            points = { (2, 2), (4, 1), (10, 3), (16, 8), (19, 2) },
            labels = { "a", "b", "c", "d", "e" } 
        ],
        [ 
            connect = true,
            color = "darkblue",
            close = true,
            points = { (5, 3), (8, 8), (16, 1) },
            labels = { "1", "2", "3" } 
        ]
    },
    ] withpen pencircle scaled 1mm ;
\stopMPcode

yielding
but this is experimentation, work in progress, not to be taken as settled.

A variant on this one is the following:

\startMPcode{doublefun}
\draw lmt_function [ \\
  xmin = 0, xmax = 10, xstep = .05, \\
  ymin = -2, ymax = 2, \\
  xticks = "bottom", xsmall = 80, xlarge = 20, \\
  yticks = "left", ysmall = 40, ylarge = 4, \\
  code = "1.5 * math.cosd(240 * math.sqrt(x))", \\
  xlabels = "yes", \\
  ylabels = "yes", \\
  ycaption = "\strut\tfd \rotate[rotation=90]{vertical}", \\
  xcaption = "\strut\tfd horizontal", \\
  pointsymbol = "dot", pointsize = 4, pointcolor = "orange", \\
  sx = 2mm, sy = 4mm, linewidth = .025mm, offset = .1mm, \\
]
  xsized 8cm
; \\
\stopMPcode

A valid question is if this should be combined with the previous but having distinctive variants is less likely to complicate the settings, as these can depend on the application:
A more complex usage (still under exploration) is demonstrated next:

\definecolor[MyColorR]{r}=.5,t=.5,a=1
\definecolor[MyColorG]{g}=.5,t=.5,a=1
\definecolor[MyColorB]{b}=.5,t=.5,a=1
\definecolor[MyColorY]{r}=.5,g=.5,t=.5,a=1

\startMPcode{doublefun}
\startMP_environment{doublefun}[
  sx = 1mm, sy = 4mm, linewidth = .025mm, offset = .1mm,
  xmin = 0, xmax = 20, xstep = .1,
  ymin = -2, ymax = 2,
  xticks = "bottom", xsmall = 80, xlarge = 20, xlabels = "nolimits",
  yticks = "left", ysmall = 40, ylarge = 4, ylabels = "yes",
  code = "1.5 * math.sind (50 * x - 150)",
  frame = "sticks", % "ticks"
]
\endMP_environment{doublefun}
\functions{doublefun}{
  [xmin = 1.0, xmax = 7.0, close = true, fillcolor = "MyColorR"],
  [xmin = 7.0, xmax = 12.0, close = true, fillcolor = "MyColorG"]
}
Of course calling the same command multiple times with different code snippets will also work.
The previous examples used new code but how about improving existing features? Let's look at outlines.

\begin{verbatim}
\startMPcode{doublefun}
  save h ; h = 12mm ;

  draw lmt_outline [text = "hello", kind = "draw", drawcolor = "darkblue", ] sized h ;

  draw lmt_outline [text = "hello", kind = "fill", fillcolor = "darkred", ] sized h shifted (3.75h,0) ;

  draw lmt_outline [text = "hello", kind = "both", fillcolor = "darkred", ] sized h shifted (7.5h,0mm) ;

  draw lmt_outline [text = "hello", kind = "both", fillcolor = "darkred", drawcolor = "darkblue", rulethickness = 1/5, ] sized h shifted (0,-1.25h) ;

  draw lmt_outline [a mixture of $\sin(x)$, $\cos(x)$, $\sqrt{x}$ etc.]
\end{verbatim}
This interface is much nicer than the one where each variant (the parameter kind above) had its own macro due to the need to group properties of the outline and fill. Let’s show some more:

```latex
\startMPcode{doublefun}
  draw lmt_outline [
    text = "\obeydiscretionaries\samplefile{tufte}",
    align = "normal",
    kind = "draw",
    drawcolor = "darkblue",
  ] xsized TextWidth ;
\stopMPcode

We thrive in information-thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

\startMPcode{doublefun}
  draw lmt_outline [
    text = "\obeydiscretionaries\samplefile{ward}",
    align = "normal,tolerant",
    width = 10cm,
  ] xsized TextWidth ;
\stopMPcode
The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

Agriculture is a fairly recent human invention, and in many ways it was one of the great stupid moves of all time. Hunter-gatherers have thousands of wild sources of food to subsist on. Agriculture changed that all, generating an overwhelming reliance on a few dozen domesticated food sources, making you extremely vulnerable to the next famine, the next locust infestation, the next potato blight. Agriculture allowed for stockpiling of surplus resources and thus, inevitably, the unequal stockpiling of them—stratification of society and the invention of classes. Thus, it allowed for the invention of poverty. I think that the punch line of the primate-human difference is that when humans invented poverty, they came up with a way of subjugating the low-ranking like nothing ever seen before in the primate world.
Another MetaFun text related mechanism deals with following a specific path. That interface has evolved over time, also due to more advanced possibilities in MkIV. But, it’s again time for an upgrade, for example:

```latex
\startMPcode{doublefun}
  draw lmt_followtext [ 
    text  = "How well does it work \{bf 1\}! ",
    path  = (fullcircle scaled 4cm),
    trace = true,
    spread = true,
  ] sized 5cm ;

draw lmt_followtext [ 
    text  = "How well does it work \{bf 2\}! ",
    path  = fullcircle scaled 4cm,
    trace = true,
    spread = false, 
    reverse = true,
  ] sized 5cm shifted (0,-6cm) ;

draw lmt_followtext [ 
    text  = "How well does it work \{bf 3\}! ",
    trace = true,
    autoscaleup = "yes"
  ] sized 5cm shifted (6cm,0) ;

draw lmt_followtext [ 
    text  = "How well does it work \{bf 4\}! ",
    path  = fullcircle scaled 2cm,
    trace = true,
    autoscaleup = "max"
  ] sized 5cm shifted (6cm,-6cm) ;
\stopMPcode
```
Although MetaPost can draw arrows a bit more code is involved than one would have expected. As a consequence, influencing the way an arrowhead is drawn as currently using variables is somewhat cumbersome. Here is an alternative:

\startMPcode{doublefun}

draw lmt_arrow [  
    path = fullcircle  
        scaled 3cm  
        shifted (0,0cm),  
    ] withcolor "darkred" ;

draw lmt_arrow [  
    location = "both"  
        path = fullcircle  
        scaled 3cm  
        rotated 90  
        shifted (0,-3.5cm),  
    ] withcolor "darkgreen" ;

draw lmt_arrow [  
    kind = "draw",  
    location = "middle",  
    alternative = "curved",  
] ;
\stopMPcode
These new interfaces for outlines, following text and arrows certainly will make it in some form to be permanent \MetaFun extensions. Some will also be used in macros at the \LaTeX end, for instance, when you load the dummy library:

\useMPlibrary{dum}

You can do this:

\startcombination
{\externalfigure{crap-001}} \{one\}
{\externalfigure{crap-004}[alternative=triangle]} \{two\}
\stopcombination

and get:
This placeholder now also can be accessed directly:

\useMPmacro
[minifun]
[placeholder]
[width=10cm, height=3cm, reduction=.2, alternative=triangle, color=orange]

The \useMPmacro is an entirely new interface to MetaFun code. Although it is still experimental, we show an example. Of course, as needed, you can throw in some Lua magic.

\startMPcalculation
presetparameters "demo" [n=5, s=1cm, ] ;
def lmt_demo = applyparameters "demo" "lmt_do_demo" enddef ;
vardef lmt_do_demo =
for i=1 upto getparameter "demo" "n" :
  draw fullcircle scaled (i*getparameter "demo" "s");
endfor;
enddef ;
\stopMPcalculation

The calculation wrapper makes this code be processed immediately, which is needed because we don't use a module.

\texttt{A the \LaTeX\ end we do this:}
\texttt{\defineMPparameterset[\text{demo}][n=\text{integer},s=\text{dimension}]}

We can now let the macro do its work:

\texttt{\startcombination[nx=4,location=\text{middle}]}

\begin{itemize}
\item \texttt{\useMPmacro[\text{demo}][n=8,s=2\text{mm}]} \hspace{1cm} \{\text{Demo 1}\}
\item \texttt{\useMPmacro[\text{demo}][n=4,s=4\text{mm}]} \hspace{1cm} \{\text{Demo 2}\}
\item \texttt{\useMPmacro[\text{demo}][n=4,s=5\text{exheight}]} \hspace{1cm} \{\text{Demo 3}\}
\item \texttt{\useMPmacro[\text{demo}]} \hspace{1cm} \{\text{Demo 4}\}
\end{itemize}

\texttt{\stopcombination}

You can also set the parameters separately. There is one catch: there is no grouping so in this case you need to reset the parameters afterwards.

\texttt{\texttt{\texttt{\texttt{\resetMPparameters[\text{demo}]}}}}

\texttt{\texttt{\texttt{\texttt{\useMPmacro[\text{demo}]}}}} \hspace{1cm} \{\text{Demo 1}\}
\texttt{\texttt{\texttt{\texttt{\resetMPparameters[\text{demo}]}}}}

These examples demonstrate that we can make \texttt{MetaPost} a bit friendlier for users who are not that fluent with the language. It remains to be seen to what extent additional modules will be written, but the possibilities are there.
As we move to \CONTEXT\ |/| \LMTX\, some decisions need to be made. For instance, how shall we introduce new functionality while not affecting the \MKIV\ code base too much with engine related tests, messy and performance sapping. Of course we remain downward compatible but the \LUAMETATEX\ engine sometimes permits multiple solutions. As this is something new, for a while we can use experimental features that may change or later disappear depending on experience. It is not a big deal to adapt to these new possibilites and users shouldn’t notice, unless of course they currently employ (yet undocumented) low-|/|level features. Part of the solution is to have separate files for \MKIV\ and \LMTX\, although that will mostly concern low level support modules; even then differences will be subtle and should not affect users. I must admit that some of those changes are mostly aesthetic and often don’t improve performance, although better performance sometimes may be the case, in particular concerning \quote {extreme} usage.

But, that said, there are areas where \LMTX\ will offer entirely new functionality. One of the areas where we are making interesting developments is in \METAFUN\, which means \METAPOST\. Now, it is not to be said that some of the core functionality needed for this will migrate from \LUAMETATEX\ (used by \LMTX\) to \LUATEX\ (used by \MKIV\). But, because (due to external pressure), the later engine is now considered stable and its functionality frozen, it is not likely to be quickly backported or at least only will happen when this has been thoroughly tested in \CONTEXT\ |/| \LMTX\.

Over the years there have been (small) improvements in the interface but while Alan Braslau and I experiment with new graphic abilities, we also are coming up
with (graphic) interfaces that suit \CONTEXT\ much better. Below, I will show some examples. \footnote{These optional argument handlers themselves are not part of \METAPOST\ but implemented in \CONTEXT\ using some experimental parsing related extensions to a version of \MPLIB\ used in \LUAMETATEX{}}

\startbuffer
\startMPcode{doublefun}
  draw lmt_grid [nx = 20, ny = 10,  
  dx = 5, dy = 2,  
  tx = 10, ty = 10,  
] sized 5cm withpen pencircle scaled 1mm withcolor "red" ;
\stopMPcode
\stopbuffer
\typebuffer[option=TEX]
\startlinecorrection
\getbuffer
\stoplinecorrection

What we see here is a key/value driven interface that has been one of the unique core properties of \CONTEXT\ since it was created. \footnote{Note the use of the color string \texttt{type} {"red"} rather than the \METAPOST\ color variable \texttt{type} \texttt{red := (1,0,0)}. This is a small detail but here imports the color definition from \CONTEXT, which might be a bit special.} Of course parameters can be optional as demonstrated in:

\startbuffer
\startMPcode{doublefun}
  draw lmt_grid [nx = 10, ny = 10,  
  dx = 1, dy = 1,  
] sized 3cm  
  withpen pencircle scaled 1mm  
  withcolor "darkblue" ;
  draw lmt_grid [nx = 10, ny = 10,  
] sized 3cm shifted (4cm,0)  
  withpen pencircle scaled .3mm  
  withcolor "darkgreen" ;
\stopMPcode
\stopbuffer
\typebuffer[option=TEX]
\startlinecorrection
\getbuffer
\stoplinecorrection

This syntax means that we can also use the already used interface documentation system.

We can actually go quite far in what we provide. It’s not so much that these things were not possible before, but the more concise user interface is something new indeed. Let’s start with drawing some simple axis:

\startbuffer
\startMPcode{doublefun}
New interfacing in MetaFun, a preview

This quote (article) is not meant as tutorial but more as a teaser about what is possible and hopefully it triggers users in coming up with useful demands. As the previous axis example is not that interesting we move on to a more complex one:

\startbuffer
\startMPcode
{doublenfunction}
\startlinecorrection
\getbuffer
\stoplinecorrection

These examples are made as part of exploring how to best interface with \metaPost, to be used by the upcoming luagraph module, a flexible and high performance set of macros aimed at drawing graphs and to visualize data. On the one hand, we want to free the user from dealing with details such as scales to get the graphics right, but on the other hand it should be possible to fine-tune when needed.

Again we draw an axis. Using strings as color names, mentioned in the footnote earlier, is not new, but already present for a while. It will use the colors defined at the \TEX\ end and is more convenient and natural than using the older \type{\MPcolor}. The \type{\lmt_connected} macro is sort of a gimmick: if you
look closely you will see that it has to start with a point (coordinate) but that one is discarded silently. Again this is something we need to deal with.

% This is obscure, and silly. I'm sure that we can do better. Sure, but so is the % HIDE related alternative trickery. We probably need something getparameterpath.

The next variant uses a list of samples. Watch the use of braces, this is something normally not part of \METAPOST\ syntax, and it signals a \LUA\-like list. This is a choice of interfacing. There are subtle and somewhat complicated aspects involved here as \METAPOST\ parsing involves primaries, secondaries, tertiaries and expressions, symbols, strings, tags and other specialities.

% Why does samples = {} require a second pair of brackets? Wouldn't a % comma-separated list be enough in Lua? Because it's \METAPOST\ which has its own % ideas about where expressions start and end.

\startbuffer\startMPcode
doublefun draw lmt_axis [ sx = 5mm, sy = 5mm, nx = 20, ny = 10, dx = 5, dy = 2, tx = 5, ty = 2, startx = 1, starty = 0, connect = true, samples = { 4, 2, 4, 9, 3, 0, 8, 9, 10, 1, 5, 3, 0, 8, 5, 6, 10, 2, 3, 5 } ], axiscolor = "darkgreen", textcolor = "darkred", samplecolors = { "darkblue" }, ] withpen pencircle scaled 1mm ; \stopMPcode\stopbuffer

\typebuffer[option=TEX]
\startlinecorrection
\getbuffer
\stoplinecorrection

We can have multiple sample lists. But of course dealing with that is really up to the underlying code, which itself is not that much related to \METAPOST\ but has been delegated to \LUA\.

\startbuffer\definecolor[tdarkred] [r=.6,a=1,t=.5] \definecolor[tdarkgreen] [g=.6,a=1,t=.5] \definecolor[tdarkblue] [b=.6,a=1,t=.5] \startMPcode doublefun save u ; let u = uniformdeviate ; draw lmt_axis [ sx = 5mm, sy = 5mm, nx = 20, ny = 10, 

\stopMPcode\stopbuffer
$dx = 5, \ dy = 2,$
$tx = 10, \ ty = 2,$
$samples = \{
  \{
    u10, u10, u10, u10, u10,
    u10, u10, u10, u10, u10,
    u10, u10, u10, u10, u10,
    u10, u10, u10, u10, u10
  \},
  \{
    u10, u10, u10, u10, u10,
    u10, u10, u10, u10, u10,
    u10, u10, u10, u10, u10,
    u10, u10, u10, u10, u10
  \},
  \{
    u10, u10, u10, u10, u10,
    u10, u10, u10, u10, u10,
    u10, u10, u10, u10, u10,
    u10, u10, u10, u10, u10
  \}\},
$\text{startx} = 1,$
$\text{starty} = 0,$
$\text{connect} = \text{true},$
$\text{axiscolor} = \text{"darkgray"},$
$\text{textcolor} = \text{"darkyellow"},$
$samplecolors = \{
  \text{"tdarkred"},
  \text{"tdarkgreen"},
  \text{"tdarkblue"}
\},$
$\textrm{withpen pencircle scaled } 1 \text{mm} ;$
$\stopMPcode$
$\stopbuffer$
$\typebuffer[option=TEX]$n
$\startlinecorrection$
$\getbuffer$
$\stoplinecorrection$

Yet another interesting variant might be this:

$\startbuffer$
$\startincode$\startluacode$\documentdata[lmx-sample-1] = \{
\{
  4, 2, 4, 9, 3, 0, 8, 9, 10, 1,
  5, 3, 0, 8, 5, 6, 10, 2, 3, 5
\}\}$\stopluacode$
$\stopbuffer$
$\typebuffer[option=TEX] \getbuffer$
$\startbuffer$
$\startMPcode$\stopMPcode$\stopbuffer$
Adding more tricks is not that complex, take, for example:

draw lmt_axis [  
  sx = 5mm, sy = 5mm,  
  nx = 20, ny = 10,  
  dx = 5, dy = 2,  
  tx = 5, ty = 2,  
  startx = 1, starty = 0,  
  samples = "lmx-sample-1",  
  axiscolor = "darkgreen",  
  textcolor = "darkred",  
  samplecolors = { "darkblue" },  
  ] withpen pencircle scaled 1mm ;
\stopMPcode
\stopbuffer

\typebuffer[option=TEX]
\startlinecorrection
\getbuffer
\stoplinecorrection
% OOPS, doesn't work!

Adding more tricks is not that complex, take, for example:

draw lmt_axis [  
  sx = 5mm, sy = 5mm,  
  nx = 20, ny = 10,  
  dx = 5, dy = 2,  
  tx = 10, ty = 10,  
  list = {  
    [  
      connect = true,  
      color = "darkred",  
      close = true,  
      points = { (1, 1), (15, 8), (2, 10) },  
      texts = { "segment 1", "segment 2", "segment 3" }  
    ],  
    [  
      connect = true,  
      color = "darkgreen",  
      close = true,  
      points = { (2, 2), (4, 1), (10, 3), (16, 8), (19, 2) },  
      labels = { "a", "b", "c", "d", "e" }  
    ],  
    [  
      connect = true,  
      color = "darkblue",  
      close = true,  
      points = { (5, 3), (8, 8), (16, 1) },  
      labels = { "1", "2", "3" }  
    ]  
  },  
  ] withpen pencircle scaled 1mm ;
\stopMPcode
\stopbuffer
source code of this document

\typebuffer[option=TEX]
yielding
\startlinecorrection
\getbuffer
\stoplinecorrection

but this is experimentation, work in progress, not to be taken as settled.

\blank
A variant on this one is the following:

\startbuffer
\startMPcode
{doublefun}
draw lmt_function [
xmin = 0, xmax = 10, xstep = .05,
  ymin = -2, ymax = 2,
  xticks = "bottom", xsmall = 80, xlarge = 20,
  yticks = "left", ysmall = 40, ylarge = 4,
  code = "1.5 * math.cosd(240 * math.sqrt(x))",
  xlabels = "yes",
  ylabels = "yes",
  ycaption = "\strut\tfd \rotate[rotation=90]{vertical}",
  xcaption = "\strut\tfd horizontal",
  pointsymbol = "dot", pointsize = 4, pointcolor = "orange",
  sx = 2mm, sy = 4mm, linewidth = .025mm, offset = .1mm,
]
xsized 8cm;
\stopMPcode
\stopbuffer
\typebuffer[option=TEX]

A valid question is if this should be combined with the previous but having
distinctive variants is less likely to complicate the settings, as these can
depend on the application:

\startlinecorrection
\getbuffer
\stoplinecorrection

A more complex usage (still under exploration) is demonstrated next:

\startbuffer
\definecolor[MyColorR][r=.5,t=.5,a=1]
\definecolor[MyColorG][g=.5,t=.5,a=1]
\definecolor[MyColorB][b=.5,t=.5,a=1]
\definecolor[MyColorY][r=.5,g=.5,t=.5,a=1]
\startMPcode(doublefun)
draw lmt_function [
  sx = 1mm, sy = 4mm, linewidth = .025mm, offset = .1mm,
source code of this document

xmin = 0, xmax = 20, xstep = .1,
ymin = -2, ymax = 2,
xticks = "bottom", xsmall = 80, xlarge = 20, xlabels = "nolimits",
yticks = "left", ysmall = 40, ylarge = 4, ylabels = "yes",
code = "1.5 * math.sin (50 * x - 150)",
frame = "sticks", % "ticks"
functions = {
  [  
    xmin = 1.0,  
xmax = 7.0, 
close = true,  
    fillcolor = "MyColorR"
  ],
  [  
    xmin = 7.0,  
xmax = 12.0, 
close = true, 
    fillcolor = "MyColorG"
  ],
  [  
    xmin = 12.0,  
xmax = 19.0, 
close = true, 
    fillcolor = "MyColorB"
  ],
  [  
    drawcolor = "darkyellow", 
    drawsize = 2,
  ],
  [  
    xmin = 4,  
xmax = 17,  
xstep = .25, 
    drawcolor = "darkmagenta", 
    shape = "steps", 
    code = "0.5 * math.random(-2,2)"
  ],
  [  
    xmin = 5.0,  
xmax = 15.0, 
close = true, 
    fillcolor = "MyColorY" 
    code = "1.75 * math.cosd (50 * x - 150)"
  ]
},
ycaption = "\rotate[rotation=90]\strut a mix of drawings",
xcaption = "\framed[foregroundstyle=\tfd]{a mixture of \sin(x), \cos(x), \sqrt{x} etc.}"
}
xsized 8cm
\stopMPcode
\stopbuffer
\typebuffer[option=TEX]

Of course calling the same command multiple times with different code snippets
will also work.

\startlinecorrection
\getbuffer
\stoplinecorrection

The previous examples used new code but how about improving existing features? Let’s look at outlines.

\startbuffer
\startMPcode{doublefun}
  save h ; h = 12mm ;
  draw lmt_outline [text = "hello", drawcolor = "darkblue", ] ysize h ;
  draw lmt_outline [text = "hello", kind = "draw", drawcolor = "darkred", ] ysize h shifted (3.75h,0) ;
  draw lmt_outline [text = "hello", kind = "fill", fillcolor = "darkred", ] ysize h shifted (7.5h,0mm) ;
  draw lmt_outline [text = "hello", kind = "both", fillcolor = "darkred", drawcolor = "darkblue", rulethickness = 1/5, ] ysize h shifted (0,-1.25h) ;
  draw lmt_outline [text = "hello", kind = "reverse", fillcolor = "darkred", drawcolor = "darkblue", rulethickness = 1/2, ] ysize h shifted (3.75h,-1.25h) ;
  draw lmt_outline [text = "hello", kind = "fillup", fillcolor = "darkgreen", rulethickness = 0, ] ysize h shifted (7.5h,-1.25h) ;
\stopMPcode
\stopbuffer
\typebuffer[option=TEX]

\startlinecorrection
\getbuffer
\stoplinecorrection
This interface is much nicer than the one where each variant (the parameter `\type` (kind) above) had its own macro due to the need to group properties of the outline and fill. Let's show some more:

\startbuffer
\startMPcode{doublefun}
  draw lmt_outline [
    text = "\obeydiscretionaries\samplefile{tufte}",
    align = "normal",
    kind = "draw",
    drawcolor = "darkblue",
  ] xsized TextWidth ;
\stopMPcode
\stopbuffer
\typebuffer{option=TEX}
\startlinecorrection
\getbuffer
\stoplinecorrection
\startbuffer
\startMPcode{doublefun}
  draw lmt_outline [
    text = "\obeydiscretionaries\samplefile{ward}",
    align = "normal,tolerant",
    width = 10cm,
    kind = "draw",
    drawcolor = "darkblue",
  ] xsized TextWidth ;
\stopMPcode
\stopbuffer
\typebuffer{option=TEX}
\startlinecorrection
\getbuffer
\stoplinecorrection
\startbuffer
\startMPcode{doublefun}
  draw lmt_outline [
    text = "\obeydiscretionaries\samplefile{sapolsky}",
    align = "normal,tolerant",
    style = "bold",
    width = 10cm,
    kind = "draw",
    rulethickness = 1/5,
    drawcolor = "darkred",
  ] xsized TextWidth ;
\stopMPcode
\stopbuffer
\typebuffer{option=TEX}
\startlinecorrection
\getbuffer
\stoplinecorrection

Another `\METAFUN` text related mechanism deals with following a specific path. That interface has evolved over time, also due to more advanced possibilities in
source code of this document

\MKIV. But, it's again time for an upgrade, for example:

\startbuffer
\startMPcode{doublefun}
  draw lmt_followtext [text = "How well does it work (\bf 1)! ",
                      path = (fullcircle scaled 4cm),
                      trace = true,
                      spread = true,]
  ysized 5cm ;

  draw lmt_followtext [text = "How well does it work (\bf 2)! ",
                      path = fullcircle scaled 4cm, 
                      trace = true, 
                      spread = false, 
                      reverse = true,]
  ysized 5cm shifted (0,-6cm) ;

  draw lmt_followtext [text = "How well does it work (\bf 3)! ",
                      trace = true,autoscaleup = "yes"]
  ysized 5cm shifted (6cm,0) ;

  draw lmt_followtext [text = "How well does it work (\bf 4)! ",
                      path = fullcircle scaled 2cm, 
                      trace = true,autoscaleup = "max"]
  ysized 5cm shifted (6cm,-6cm) ;
\stopMPcode
\stopbuffer
\typebuffer[option=TEX]
\startlinecorrection
\getbuffer
\stoplinecorrection

Although \ METAPOST \ can draw arrows a bit more code is involved than one would have expected. As a consequence, influencing the way an arrowhead is drawn as currently using variables is somewhat cumbersome. Here is an alternative:

\startbuffer
\startMPcode{doublefun}
  draw lmt_arrow [
    path = fullcircle
    scaled 3cm,
  ] withcolor "darkred" ;

  draw lmt_arrow [
    location = "both"
    path = fullcircle
    scaled 3cm
    rotated 90
    shifted (0,-3.5cm),
  ] withcolor "darkgreen" ;
  draw lmt_arrow [
These new interfaces for outlines, following text and arrows certainly will make it in some form to be permanent `\METAFUN\` extensions. Some will also be used in macros at the `\TEX\` end, for instance, when you load the dummy library:

```latex
\startbuffer
\useMPlibrary[dum]
\stopbuffer
\typebuffer[option=TEX] \getbuffer

You can do this:

```latex
\startbuffer
\useMPmacro[\minifun]
\stopbuffer
```

and get:

```latex
\typebuffer[option=TEX] \getbuffer

This placeholder now also can be accessed directly:

```latex
\startbuffer
\useMPmacro[\minifun]
```
source code of this document

\begin{verbatim}
[placeholder]
[width=10cm,
 height=3cm,
 reduction=.2,
 alternative=triangle,
 color=orange]\end{verbatim}

\typebuffer[option=TEX]
\startlinecorrection
\getbuffer
\stoplinecorrection

The \texttt{\useMPmacro} is an entirely new interface to \texttt{\METAFUN\ code}. Although it is still experimental, we show an example. Of course, as needed, you can throw in some \texttt{\LUA} magic.

\begin{verbatim}
\startbuffer
\startMPcalculation
  \presetparameters "demo" [
    \n = 5,
    \s = 1cm,
  ];
  \def lmt_demo = \applyparameters "demo" "lmt_do_demo" enddef ;
  \vardef lmt_do_demo =
    \for i=1 \upto \getparameter "demo" \n :
      \draw fullcircle scaled (i*\getparameter "demo" \s);
    \endfor ;
\enddef 
\stopMPcalculation
\stopbuffer
\typebuffer[option=TEX]
\getbuffer
\end{verbatim}

We can now let the macro do its work:

\begin{verbatim}
\startbuffer
\startcombination [nx=4,location=middle]
  \{\useMPmacro[\demo][n=8,s=2mm]\} \{Demo 1\}
  \{\useMPmacro[\demo][n=4,s=4mm]\} \{Demo 2\}
  \{\useMPmacro[\demo][n=4,s=5\exheight]\} \{Demo 3\}
  \{\useMPmacro[\demo]\} \{Demo 4\}
\stopcombination
\stopbuffer
\end{verbatim}
source code of this document

\typebuffer[option=TEX]
\startlinecorrection
\getbuffer
\stoplinecorrection

You can also set the parameters separately. There is one catch: there is no grouping so in this case you need to reset the parameters afterwards.

\starttyping
\presetMPparameters[demo][n=8] \useMPmacro[demo]\blank
\presetMPparameters[demo][n=4] \useMPmacro[demo]\blank
\resetMPparameters [demo] \useMPmacro[demo]\blank
\stoptyping

These examples demonstrate that we can make METAPOST a bit friendlier for users who are not that fluent with the language. It remains to be seen to what extent additional modules will be written, but the possibilities are there.

\stopdocument