The visualcounter module

Aditya Mahajan

November 4, 2012

```
\usemodule[visualcounter]
```

- Find TEX documents to be too boring?
- Voilà, the visualcounter module!
- Make your presentations stand out.
- Turn any counter into a picture.

The above effect was achieved by first defining a `visualitem` counter and a symbol `visual` that uses that counter:

```
\definevisualcounter
  [visualitem]
  [scratchmarks]
  [ counter=\getvalue{v_strc_itemgroups_counter},
    width=1.5bp,
    height=1.2ExHeight,
    distance=3bp]

\definesymbol[visual][\usevisualcounter{visualitem}]
```

and then using the symbol `visual` in an itemization:

```
\startitemize[visual, ...]
    \item ...
    \item ...
\stopitemize
```

Notice the counter used for page numbering? That was achieved by first defining a `visualpage` counter:

```
\definevisualcounter
  [visualpage]
  [mayanumbers]
  [
    counter=userpage,
    maxwidth=\textwidth,
  ]
```

and setting it as the footer text:

```
\setupfootertexts[\usevisualcounter{visualpage}]
```

The above examples show the basic usage of the modules. The module provides two commands: **\definevisualcounter** to define a visual counter

```
\definevisualcounter
    [...]     % name of the counter
    [...]     % optional name of the parent counter
    [
     ...=..., % key-value settings
    ]
```

and **\usevisualcounter** to use an already defined counter

```
\usevisualcounter
  [...=...] % key-value settings
  {...}     % name of the counter
```

So, how do I use this?

Visual counters are defined and used in two ways:

- Using a low level interface that explicitly sets the current values of the counter, last count, the METAPOST graphic that draws the counter, and the color palette.
- Using higher-level interfaces built on top of the low-level interface that allows you to specify a *structure counter* like those used for page numbering, itemizations, descriptions, etc.

The low-level interface

To begin with, lets not worry about how to define METAPOST graphics that draw the counter. The module provides a predefined set of `visualcounter`s, and, for now, we'll just use those: the `scratchmarks` counter. Details on defining new counters are explained later.

Suppose that I want to show that I am on page 6 out of 12 pages:

which uses a predefined counter `scratchmarks` and was typed as follows:

```
\usevisualcounter[n=6, last=12]{scratchmarks}
```

The counter may be made smaller

or may use a different color palette

These settings are changed using `\setupvisualcounter`. In particular, to get a small counter, use:

```
\setupvisualcounter[scratchmarks]
                   [width=1pt, height=8pt, distance=2pt]
```

and to change the color palette, use:

```
\setupvisualcounter[scratchmarks][palette=brightred]
```

where the `brightred` palette was defined as

```
\definecolor[bright-red] [h=DE1B1B]
\definecolor[dull-black] [h=2B2B2B]
\definecolor[dull-yellow][h=E9E581]

\definepalet
   [brightred]
   [past=dull-black, active=bright-red, future=dull-yellow]
```

The high-level interface

The high level interface is useful to display a ConTEXt counter. Rather than manually setting the value of `n`, `last`, `text`, and `maxtext` (the last two are used only by a few counters), simply set the value of a `counter` and the other values are automatically generated. As an example, recall the set up for displaying the page numbers in the footer:

```
\definevisualcounter
  [visualpage]
  [mayanumbers]
  [
    counter=userpage,
    maxwidth=\textwidth,
  ]
```

In the above example, `userpage` is the name of the counter that keeps track of the user page number.

A list of some commonly used ConTEXt counters is given below:

Page numbers	`userpage`
Item group numbers	`\v_strc_itemgroups_counter`[1]
Enumeration number	name of the enumeration

[1] Macros names with underscore are internal ConTEXt macros, and generally are not meant to be used in user code. The easiest way to set the value of `counter` to `\v_strc_itemgroups_counter` is to use:

```
counter=\getvalue{v_strc_itemgroups_counter},
```

Changing the color of counters

All counters use three colors: `past` to display past counters, `active` to display current counter, and `future` to display future counters. To change the color scheme, first define a new color palette (which is misspelled in ConTEXt is `colorpalet`):

```
\definecolorpalet
    [...], % Name of the color palette
    [
      past=..., % any previously defined \CONTEXT{} color
    active=..., % any previously defined \CONTEXT{} color
    future=..., % any previously defined \CONTEXT{} color
    ]
```

and then set the color palette of a particular counter either using

```
\definevisualcounter
    [...]     % name of the counter
    [...]     % optional name of the parent counter
    [
      ...,
      palette=..., % name of a previously defined colorpalet
      ...
    ]
```

or using

```
\setupvisualcounter
    [...]     % name of the counter
    [
      ...,
      palette=..., % name of a previously defined colorpalet
      ...
    ]
```

Changing the size of the counters

All predefined counters have tunable parameters (such as `width`, `height`, and `distance`) that change the size of the counter. The default sizes are given in terms of `EmWidth` or `ExHeight`; so they adapt to the size of the surrounding text.

Changing the style of text displayed in the counters

Some visual counters (currently, only the `countdown` counter) also display text. The style of this text may be set using:

```
\setupvisualcounter
    [...]     % name of the counter
    [
      ...,
      style=..., %any valid \CONTEXT{} style
      color=..., %any valid \CONTEXT{} color
      ...
    ]
```

Note that changing the font size in the `style` affects the value of `EmWidth` and `ExHeight` and therefore also scales the counter appropriately. If this is not desirable, then you also need to set the size of the counter in dimensions that are independent of body-fontsize.

Predefined counters

The `visualcounter` module provides the following predefined counters:

- The `scratchmarks` counter
- The `mayanumbers` counter
- The `countdown` counter
- The `markers` counter
- The `progressbar` counter
- The `pulseline` counter

The scratchmarks counter

The scratchmarks counter is inspired by the fuzzycount counter that is part of the ConTEXt's metapost library txt. The output looks as follows:

3 out of 12

4 out of 12

5 out of 12

6 out of 12

The scratchmarks counter has the following tunable parameters:

- width (default 3bp): the width of each stroke
- height (default 3ExHeight): the length of the marker (and strictly speaking, not the height; the real height is height × sin(angle)).
- distance (default 0.5EmWidth): the distance between two successive markers. The distance is measured from the middle of one marker to the middle of the other (that is, it does not take the width of the stroke into account).
- angle (default 75): the angle of the forward markers. The angle of the backward marker is -angle. Only angles between -90 and 90 give proper output.

For example, the output with `width=1.5bp, angle=45` is:

3 out of 12

4 out of 12

5 out of 12

6 out of 12

An angle less than `0` changes the direction of the stroke. For example the output with `width=1.5bp, angle=-45` is:

3 out of 12

4 out of 12

5 out of 12

6 out of 12

2 The mayanumbers counter

The mayanumbers counter is inspired by the Mayan numbering system that I saw in the documentary “Breaking the Maya code”. It counter does not strictly follow the Mayan numbering system. The Mayan numbering system is written vertically; the output of this counter is horizontal which makes it more useful for displaying page numbers in presentations.

3 out of 12

4 out of 12

5 out of 12

6 out of 12

The shape of the small and the large markers is as follows:

The shape of the small marker

The shape of the large marker

The `scratchmarks` counter has the following tunable parameters:

- `width` (default value `1EmWidth`): The width of the small marker.
- `height` (default value `1ExHeight`): The height of the markers.
- `distance` (default value `0.25EmWidth`): The distance between two small markers. The distance between each group of four small counters is `2*distance`.

For example, to get an output that is half the default size, use the options

`width=0.5EmWidth, height=0.5ExHeight, distance=0.125EmWidth`.

3 out of 12

4 out of 12

5 out of 12

6 out of 12

The countdown counter

This counter is inspired by the spinning wheel on the iPhone and on Google images.

3 out of 12

4 out of 12

5 out of 12

6 out of 12

The countdown counter may also be used with a text display of the counter.

3 out of 12

4 out of 12

5 out of 12

6 out of 12

The example below shows how the counter changes when the number of steps are increased.

13 out of 24

14 out of 36

25 out of 48

36 out of 60

The countdown counter has the following tunable parameters:

- width (default value 1EmWidth) and height (default value 1ExHeight): The maximum of these two determine the difference between the inner and outer diameter of the ring.
- text (not set by default): The text to be displayed in the middle of the counter.
- maxtext (not set by default): The diameter of the inner circle is equal to 1.5 times the maximum of the width and the height of maxtext.
- distance (default value 3EmWidth): The distance between the consecutive markers along the circumference of the outer circle is equal to distance/last.

For example, to get a continuous circle set distance=0pt:

13 out of 24

14 out of 36

25 out of 48

36 out of 60

When the high level interface is used, i.e., when the option counter=... is set, the value of text is the converted value of the counter, and the value of maxtext is the converted value of last counter.

The markers counter

The markers counter is inspired by the section markers displayed by the LaTeX beamer package. This module comes in two shapes: the default shape is circle

3 out of 12

4 out of 12

5 out of 12

6 out of 12

The alternative shape is a square, which is selected using

```
\setupvisualcounter
    [...]     % name of the counter
    [
      ...
      mpsetups=visualcounter:markers:square,
      ...
    ]
```

3 out of 12

4 out of 12

5 out of 12

6 out of 12

The `visualcounter:markers:square` is a predefined alternative shape. Before discussing how to define a new shape, lets look at the tunable parameters of the `markers` counter:

- `width` (default value `1EmWidth`): The width of the each marker.
- `distance` (default value `1EmWidth`): The edge to edge distance between successive markers.
- `mpsetups` (default value `visualcounter:markers:circle`): A `useMPgraphic` that determines the shape and the display of the markers. The module comes with two predefined shapes, `visualcounter:markers:circle` and `visualcounter:markers:square`.

To define a new shape, you have to create a `useMPgraphic` that defines three METAPOST macros:

- `show_past_marker(expr shift)`
- `show_active_marker(expr shift)`
- `show_future_marker(expr shift)`

The argument to these macros is the shift calculated based on their position. The macros are responsible for using the shift amount either for horizontal shift or for vertical shift, and coloring the markers with appropriate colors. The three colors from the color palette are available as `past_color`, `active_color`, `future_color`.

As an example, lets consider a new visual counter that displays a “star rating”:

1 out of 5 3 out of 5 5 out of 5

For better visual effect, I should also change the color palette.

To create such a counter, first create a useMPgraphic as follows (I could have used any name instead of visualcounter:markers:star):

```
\startuseMPgraphic{visualcounter:markers:star}
% Copied from http://tug.org/pracjourn/2012-1/hefferon.html
def fullstar =
  hide (
    z0  = origin;
    z1  = (x0, 0.5);
    z2  = ((z1 - z0) rotated (  360/5)) + z0 ;
    z3  = ((z1 - z0) rotated (2*360/5)) + z0 ;
    z4  = ((z1 - z0) rotated (3*360/5)) + z0 ;
    z5  = ((z1 - z0) rotated (4*360/5)) + z0 ;
    z6  = whatever[z1, z3] = whatever[z2, z5];
    z7  = whatever[z1, z3] = whatever[z2, z4];
    z8  = whatever[z2, z4] = whatever[z3, z5];
    z9  = whatever[z1, z4] = whatever[z3, z5];
    z10 = whatever[z1, z4] = whatever[z2, z5];
  )
  (z1 -- z6 -- z2 -- z7 -- z3 -- z8 -- z4
      -- z9 -- z5 -- z10 -- cycle)
enddef;
```

```
def show_past_marker(expr shift) =
  newpath p;
  p := fullstar xyscaled(width, width) shifted (shift, 0);

  fill p withcolor active_color;
  draw p withcolor past_color;
enddef;

def show_active_marker(expr shift) =
    show_past_marker(shift);
enddef;

def show_future_marker(expr shift) =
  newpath p;
  p := fullstar xyscaled(width, width) shifted (shift, 0);

  fill p withcolor future_color;
  draw p withcolor 0.5*future_color;
enddef;
\stopuseMPgraphic
```

Next, define a new `visaulcounter` that inherits from `markers` but uses a different `mpsetups`:

```
\definevisualcounter
  [stars]
  [markers]
  [mpsetups=visualcounter:markers:star, % use "star" marker
   width=1.5EmWidth,
   distance=0.5EmWidth,
   last=5, % rating out of 5
  ]
```

To use this counter, type:

```
\usevisualcounter[n=4]{stars}
```

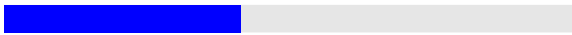
which gives: 

The progressbar counter

The progressbar counter is based on a question on TeX.SE. This counter is not yet finalized. I am still working on the interface to change the shape of the progress bar, and perhaps add some shading.

3 out of 12

4 out of 12



5 out of 12

6 out of 12

6 The pulseline counter

The pulseline counter is based on the heart pulses shown in a heart rate measurement device. This counter is not yet finalized.

3 out of 12

4 out of 12

5 out of 12

6 out of 12