

Subject: Elastic Tabstops coming soon to TextEdit & Memo
From: Graeme Geldenhuys <graemeg@gmail.com>
Date: 2016-05-17 11:05
Newsgroups: fpGUI.support

Hi,

I've recently been working on implementing Elastic Tabstops (ET) for both TextEdit and Memo widgets. My prototype implementation is already rendering 100% in a TfpGUI.Memo and the results are amazing!

The only outstanding feature is that the edit cursor doesn't know anything about the TAB (#9) character. This applies to both TextEdit and Memo widgets. A rather large outstanding feature - I know. 😞 But I'm working on implementing that soon.

A pre-requisite of ET is that tabs are defined as pixels (highly preferred), and that every line can have its own tabstops defined. Just like any word processor has allowed for decades. With ET, a tabstop is not a set width any more - it grows or shrinks based on the content inside a column of data (aka a "cell" of data).

Some benefits of ET:

1. Highly configurable. If you like your code to look like 1 or 2 space indented, ET can do that for you. What more indentation space so you can more easily see code blocks in nested code - no problem too.
2. Manually aligning code is now a thing of the past. Think parameter lists, enum definitions, code comments at the end of a line of code. One simple TAB character, and ET does the rest.
3. File sizes reduce, because a single TAB uses less disk space than multiple Space characters.
4. Because of (3), in theory, it is also faster to parse such code,

- because there is less characters of text to parse.
5. Code optimised for ET is fairly backwards compatible with editors than only support simple TAB (eg: 1 tab = 8 spaces) indentation.
 6. My all time favourite! Much better looking Proportional fonts can now be used for coding. No need to limit yourself to mostly ugly mono-spaced fonts.
 6. It works fantastic with CSV or TAB delimited data. SQL is also very nice looking with a ET enabled editor.

ET is by no means the silver bullet, but it is a lot more flexible than anything that came before it, and makes most Tab versus Space indentation arguments moot.

Don't know what Elastic Tabstops are?

The best thing since sliced bread! 😊 Here are a couple of links to tell you more.

Elastic Tabstops homepage:

<http://nickgravgaard.com/elastic-tabstops/>

A nice overview of ET:

<http://tibleiz.net/code-browser/elastic-tabstops.html>

I also suggest you try an editor that supports it. eg: jEdit has had support for it for years, and that is where I really started liking ET. Nick's website (person that came up with the idea) has a small Java demo showing Elastic Tabstops too - very handy for playing around with and trying various code and text to see what it does.

I've also attached some screenshots showing my ET implementation rendering a tab indented/aligned file using a TfpGMemo. Unfortunately my Animated Gif setup wasn't working, otherwise I would have shown the

Elastic Tabstops coming soon to TextEdit & Memo

dynamic sizing and alignment in action. The ET homepage has a nice animated GIF showing dynamic sizing of tabs.

Regards,

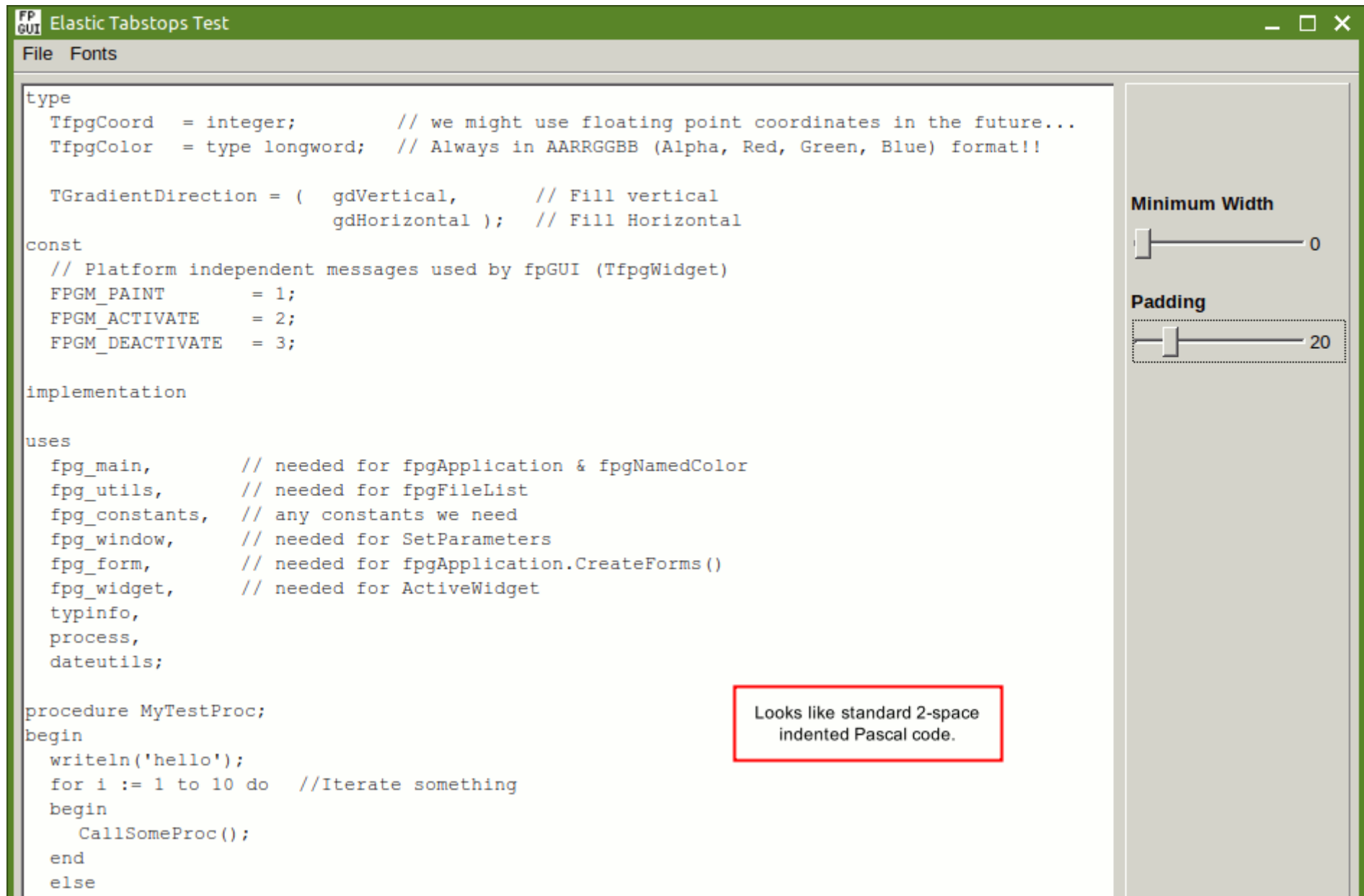
- Graeme -

--

fpGUI Toolkit - a cross-platform GUI toolkit using Free Pascal

<http://fpgui.sourceforge.net/>

~~—screenshot01.png~~



The screenshot shows a window titled "fpGUI Elastic Tabstops Test" with a menu bar containing "File" and "Fonts". The main area displays Pascal code with elastic tabstops. The code is as follows:

```
type
  TfpgCoord  = integer;      // we might use floating point coordinates in the future...
  TfpgColor  = type longword; // Always in AARRGGBB (Alpha, Red, Green, Blue) format!!

  TGradientDirection = (  gdVertical,      // Fill vertical
                          gdHorizontal );  // Fill Horizontal

const
  // Platform independent messages used by fpGUI (TfpgWidget)
  FPGM_PAINT      = 1;
  FPGM_ACTIVATE   = 2;
  FPGM_DEACTIVATE = 3;

implementation

uses
  fpg_main,      // needed for fpgApplication & fpgNamedColor
  fpg_utils,     // needed for fpgFileList
  fpg_constants, // any constants we need
  fpg_window,   // needed for SetParameters
  fpg_form,     // needed for fpgApplication.CreateForms()
  fpg_widget,   // needed for ActiveWidget
  typinfo,
  process,
  dateutils;

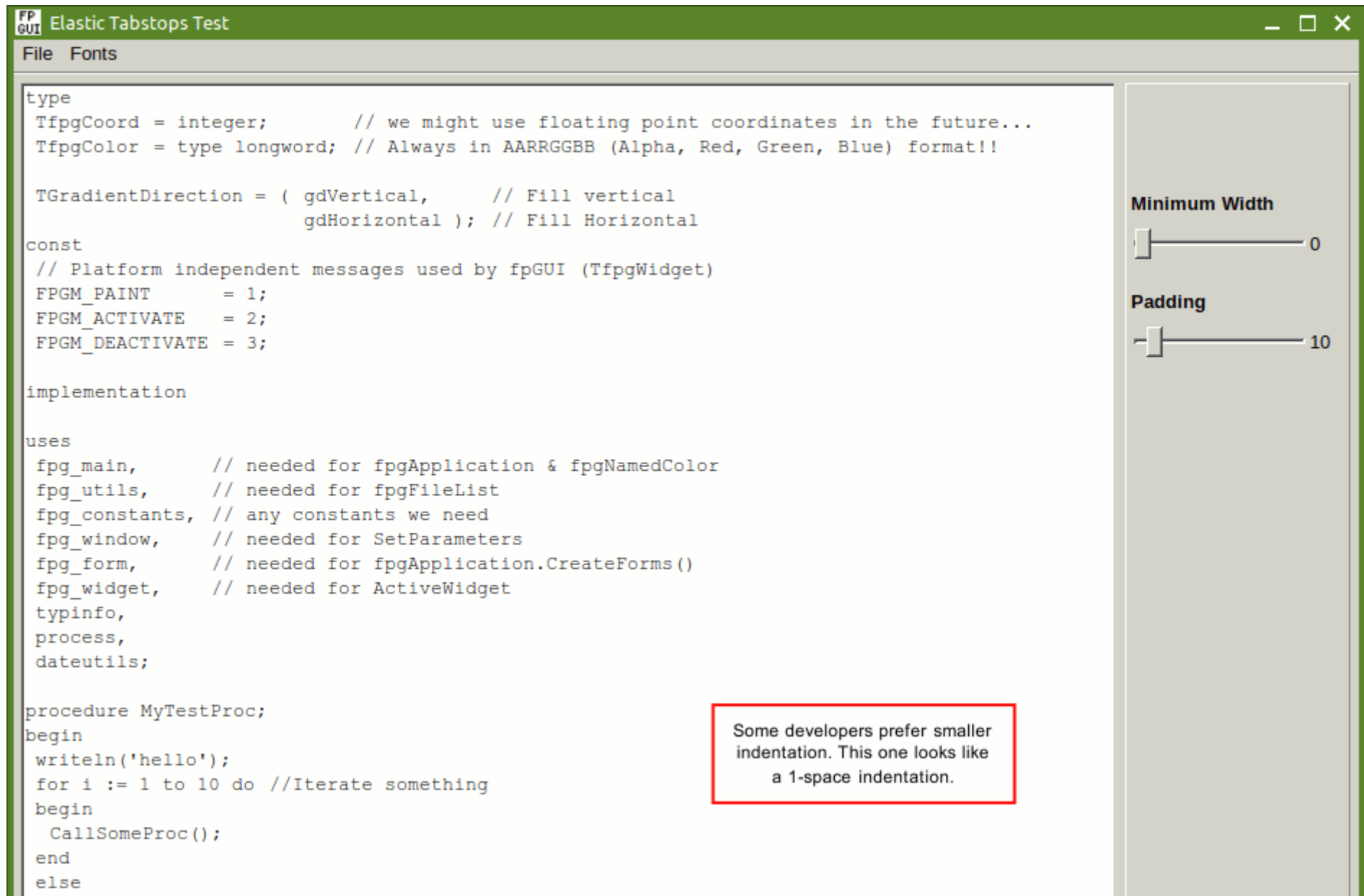
procedure MyTestProc;
begin
  writeln('hello');
  for i := 1 to 10 do //Iterate something
  begin
    CallSomeProc();
  end
end
else
```

On the right side of the window, there is a settings panel with two sliders:

- Minimum Width**: A slider with a value of 0.
- Padding**: A slider with a value of 20, which is highlighted with a dashed border.

A red box highlights a portion of the code with the text: "Looks like standard 2-space indented Pascal code."

—screenshot02.png



The screenshot shows a window titled "Elastic Tabstops Test" with a menu bar containing "File" and "Fonts". The main area displays Pascal code with elastic tabstops. The code is as follows:

```
type
  TfpgCoord = integer;      // we might use floating point coordinates in the future...
  TfpgColor = type longword; // Always in AARRGGBB (Alpha, Red, Green, Blue) format!!

  TGradientDirection = ( gdVertical,      // Fill vertical
                          gdHorizontal ); // Fill Horizontal

const
  // Platform independent messages used by fpGUI (TfpgWidget)
  FPGM_PAINT      = 1;
  FPGM_ACTIVATE  = 2;
  FPGM_DEACTIVATE = 3;

implementation

uses
  fpg_main,      // needed for fpgApplication & fpgNamedColor
  fpg_utils,     // needed for fpgFileList
  fpg_constants, // any constants we need
  fpg_window,   // needed for SetParameters
  fpg_form,     // needed for fpgApplication.CreateForms()
  fpg_widget,   // needed for ActiveWidget
  typinfo,
  process,
  dateutils;

procedure MyTestProc;
begin
  writeln('hello');
  for i := 1 to 10 do //Iterate something
  begin
    CallSomeProc();
  end
end
else
```

On the right side of the window, there is a settings panel with two sliders:

- Minimum Width**: A slider set to 0.
- Padding**: A slider set to 10.

A red-bordered box contains the following text:

Some developers prefer smaller indentation. This one looks like a 1-space indentation.

—screenshot03.png

The screenshot shows a window titled "fpGUI Elastic Tabstops Test" with a menu bar containing "File" and "Fonts". The main area is a code editor displaying the following code:

```

type
  TfpgCoord = integer;           // we might use floating point coordinates in the future...
  TfpgColor = type longword;     // Always in AARRGGBB (Alpha, Red, Green, Blue) format!!

  TGradientDirection = ( gdVertical,      // Fill vertical
                          gdHorizontal ); // Fill Horizontal

const
  // Platform independent messages used by fpGUI (TfpgWidget)
  FPGM_PAINT      = 1;
  FPGM_ACTIVATE  = 2;
  FPGM_DEACTIVATE = 3;

implementation

uses
  fpg_main,      // needed for fpgApplication & fpgNamedColor
  fpg_utils,     // needed for fpgFileList
  fpg_constants, // any constants we need
  fpg_window,   // needed for SetParameters
  fpg_form,     // needed for fpgApplication.CreateForms()
  fpg_widget,   // needed for ActiveWidget
  typinfo,
  process,
  dateutils;

procedure MyTestProc;
begin
  writeln('hello');
  for i := 1 to 10 do //Iterate something
  begin
    CallSomeProc();
  end
end
else

```

On the right side of the window, there is a settings panel with two sliders:

- Minimum Width:** A slider with a value of 30.
- Padding:** A slider with a value of 16.

A red box highlights a section of code with larger indentation, and a text box next to it says: "Here is larger indentation. Useful while coding complex multi-level nested code blocks."

—screenshot04.png

The screenshot shows a window titled "FP GUI Elastic Tabstops Test" with a menu bar containing "File" and "Fonts". The main area displays Pascal code with elastic tabstops. The code is organized into sections: "type", "const", "implementation", "uses", and "procedure MyTestProc". The "type" section defines "TfpgCoord" as an integer and "TfpgColor" as a longword. The "const" section defines platform-independent messages. The "uses" section lists various modules. The "procedure MyTestProc" contains a loop that writes "hello" and iterates something.

On the right side, there is a settings panel with two sliders: "Minimum Width" set to 30 and "Padding" set to 16. The "Padding" slider is highlighted with a dashed border.

A red-bordered box contains the following text:

```
Why not try a more beautiful proportional
font like Ubuntu for your coding.
ET will make sure indentation and
alignment is always correct.
```

—screenshot05.png

fpGUI Elastic Tabstops Test

File Fonts

```

type
  TfpgCoord = integer;      // we might use floating point coordinates in the future...
  TfpgColor = type longword; // Always in AARRGGBB (Alpha, Red, Green, Blue) format!!

  TGradientDirection = ( gdVertical,    // Fill vertical
                          gdHorizontal); // Fill Horizontal

const
  // Platform independent messages used by fpGUI (TfpgWidget)
  FPGM_PAINT      = 1;
  FPGM_ACTIVATE   = 2;
  FPGM_DEACTIVATE = 3;

implementation

uses
  fpg_main,      // needed for fpgApplication & fpgNamedColor
  fpg_utils,     // needed for fpgFileList
  fpg_constants, // any constants we need
  fpg_window,   // needed for SetParameters
  fpg_form,     // needed for fpgApplication.CreateForms()
  fpg_widget,   // needed for ActiveWidget
  typinfo,
  process,
  dateutils;

procedure MyTestProc;
begin
  writeln('hello');
  for i := 1 to 10 do //Iterate something
  begin
    CallSomeProc();
  end
end
else

```

Minimum Width 30

Padding 16

Lets be all wacky and try the Comic Sans proportional font. ET still makes sure your code looks beautiful and perfectly aligned.

—Attachments:—

screenshot01.png	19.6 KB
screenshot02.png	20.2 KB
screenshot03.png	20.5 KB
screenshot04.png	23.7 KB
screenshot05.png	30.4 KB