The **Adafruit_mfGFX** library is a modified Adafruit_GFX library which supports multiple fonts of variable width and fixed height. These fonts are most easily generated from TTF fonts using TheDotFactory for conversion to a format suitable with this library.

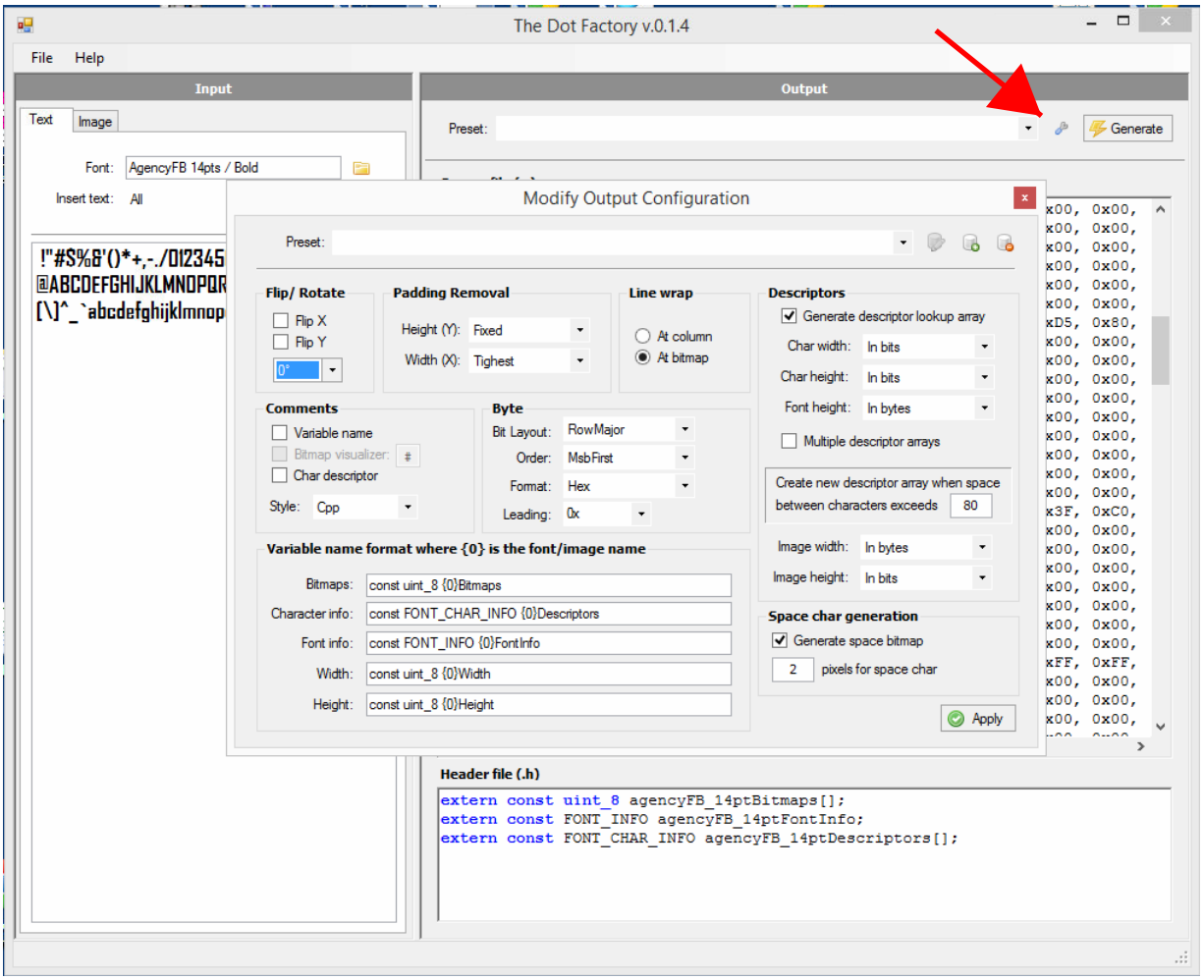
The library is made up of the following files:

Adafruit_mfGFX.cpp	- modified GFX library includes new setFont() function
Adafruit_mfGFX.h	
Adafruit_SharpMem.cpp	- Display driver for Sharp Memory Display, can be any Adafruit GFX compatible driver
Adafruit_SharpMem.h	
fonts.cpp	- Font file containing all font bitmap arrays and corresponding description arrays
fonts.h	
SharMemDisplay.ino	- Test program for Sharp Memory Display. Written for selected display driver

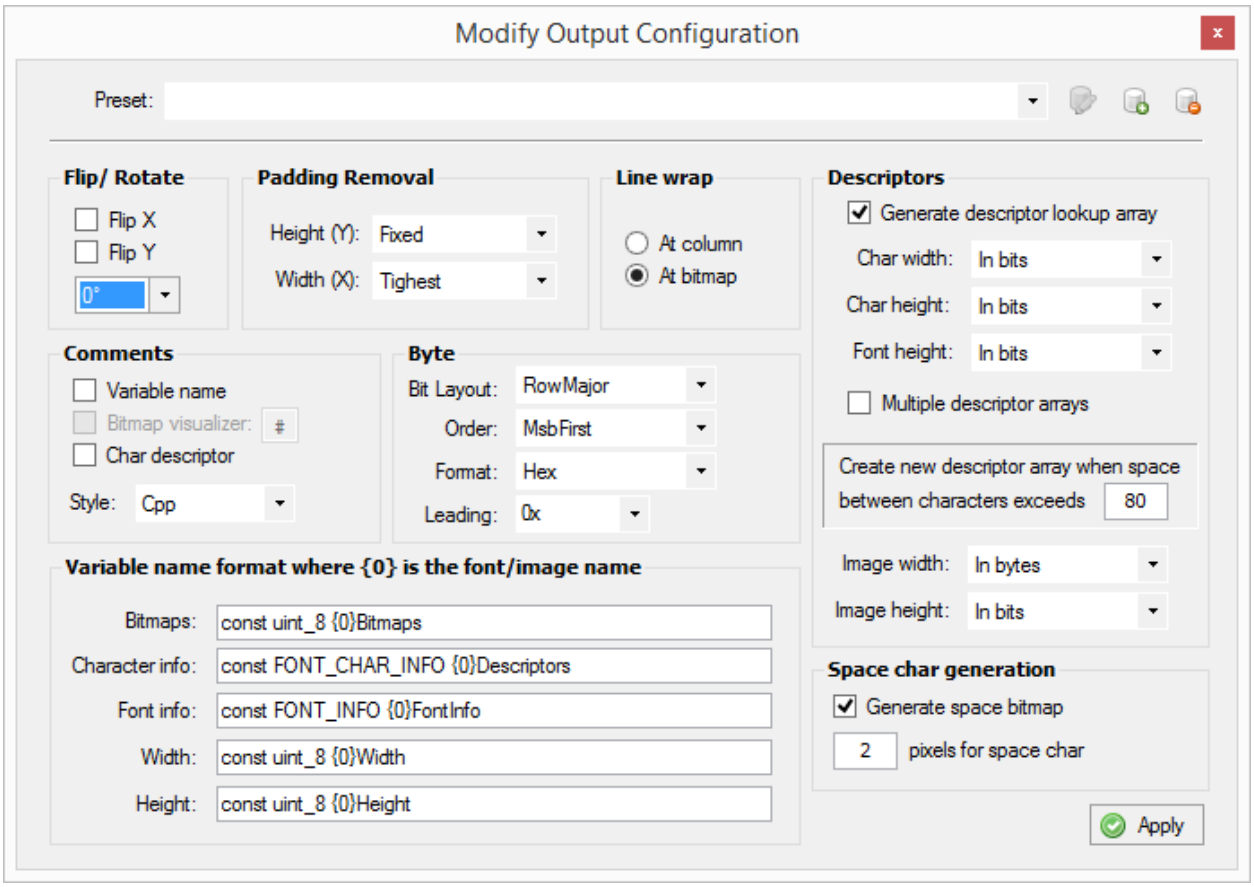
These files, when in a single directory, were compiled successfully with the Spark CLI cloud compile option.

Creating Fonts using TheDotFactory:
TheDotFactory is a free Windows program for converting existing TTF (truetype) fonts for use with the Adafruit_mfGFX library. Any font in any size and format on the user's system may be converted but will vary in appearance on the target display depending on its resolution. As such, it is recommend that smaller fonts be used and scaled using the mfGFX library's setSize() command. Experimentation is to be expected ☺

Key to creating the correct font data structures is the "Modify Output Configuration" button (highlighted)



The following screen shows the settings which **MUST** be applied in order to obtain the correct data:



Once set, click **Apply** and then **Generate** to obtain the data in the Source file (.c) window. Right clicking on the window will allow the contents to be copied to the clipboard. It is highly recommend that this be copied to an editor as sections will be cut&paste into the fonts.cpp file.

There are two key arrays generated – the Bitmaps[] array (sample below) which contains the character bitmap data in HEX format,

```
const uint_8 agencyFB_14ptBitmaps[] =
{
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0x00, 0x00, 0xC0, 0xC0, 0x00, 0x00, 0x00, 0x00,
    0x00, 0xD8, 0xD8, 0xD8, 0xD8, 0x90, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x0D, 0x80, 0x19, 0x80, 0x19, 0x80, 0x19, 0x00, 0x7F, 0xC0, 0x7F, 0xC0, 0x1B, 0x00, 0x33, 0x00, 0x33, 0x00,
```

and the Descriptors[] array (sample below) listing the character width, height and offset (into the Bitmaps array) . Both arrays must be copied to the fonts.c file.

```
const FONT_CHAR_INFO agencyFB_14ptDescriptors[] =
{
    {2, 20, 0},         //
    {2, 20, 20},        // !
    {5, 20, 40},        // "
    {10, 20, 60},       // #
    {7, 20, 100},       // $
```

Note that Adafruit_mfGFX ***only*** works with fonts of **fixed** height and variable width.

Each table can be copied as-is (some modifications will be made in the next steps) into the fonts.c file. Once copied, the Bitmaps[] array must have an extra row added (see below) to the top which defines the starting and ending ASCII charaters in the entire table. TheDotFactory only generates “standard” ASCII characters between 0x20 and 0x7F. **This is crucial as fonts will not display correctly without this added line.**

```
//
//  Font data for AgencyFB 11pt
//
#include "fonts.h"

// Character bitmaps for timesNewRoman 8pt
const uint8_t timesNewRoman_8ptBitmaps[] =
{
    0x20, 0x7F,     // Start Character, End Character
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x00, 0x80, 0x00, 0x00, 0x00,
    0x00, 0xA0, 0xA0, 0xA0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x24, 0x24, 0x48, 0xFC, 0x48, 0xFC, 0x90, 0x90, 0x00, 0x00, 0x00,
```

Then, the Descriptors array definition must be changed from “const FONT_CHAR_INFO “ to “const FontDescriptor” as shown below.

```
// Character bitmaps for timesNewRoman 8pt
const FontDescriptor timesNewRoman_8ptDescriptors[] =
{
    {2, 12, 0},
    {1, 12, 12},
    {3, 12, 24},
```

Next, in the comments added to each Descriptor line (eg. // a), some characters will cause compile problems so you **MUST** remove all text after the coma on each line.

Now, fonts.h must be modified to define the new font information. A font selection #define and references to the font arrays must be added (see below for example).

```
// Font selection descriptors - Add an entry for each new font and number sequentially
#define TIMESNR_8   0
#define CENTURY_8   1
#define ARIAL_8     2
#define COMICS_8    3
#define TEST        4

#define FONT_START 0
#define FONT_END 1

struct FontDescriptor
{
    uint8_t width;      // width in bits
    uint8_t height;     // char height in bits
    uint16_t offset;    // offset of char into char array
};

// Font references - add pair of references for each new font
extern const uint8_t timesNewRoman_8ptBitmaps[];
extern const FontDescriptor timesNewRoman_8ptDescriptors[];

extern const uint8_t centuryGothic_8ptBitmaps[];
extern const FontDescriptor centuryGothic_8ptDescriptors[];

extern const uint8_t arial_8ptBitmaps[];
extern const FontDescriptor arial_8ptDescriptors[];

extern const uint8_t comicSansMS_8ptBitmaps[];
extern const FontDescriptor comicSansMS_8ptDescriptors[];

extern const uint8_t testBitmaps[];
extern const FontDescriptor testDescriptors[];
```

Finally, the Adafruit_mGFX.cpp file must be modified to add the new font to the setFont() function. A new “case” must added to setFont() using the font selector defined above in font.h (see below).

```
Adafruit_GFX::Adafruit_GFX(int16_t w, int16_t h):
  WIDTH(w), HEIGHT(h)
{
  _width    = WIDTH;
  _height   = HEIGHT;
  rotation  = 0;
  cursor_y  = cursor_x    = 0;
  textsize  = 1;
  textcolor = textbgcolor = 0xFFFF;
  wrap      = true;
  setFont(ARIAL_8);     // May be set to TIMESNR_8, CENTURY_8, COMICS_8 or TEST (for testing candidate fonts)
 }

void Adafruit_GFX::setFont(uint8_t f) {
  font = f;
  switch(font) {
    case TIMESNR_8:
      fontData = timesNewRoman_8ptBitmaps;
      fontDesc = timesNewRoman_8ptDescriptors;
      fontKern = 1;
      break;
    case CENTURY_8:
      fontData = centuryGothic_8ptBitmaps;
      fontDesc = centuryGothic_8ptDescriptors;
      fontKern = 1;
      break;
    case ARIAL_8:
      fontData = arial_8ptBitmaps;
      fontDesc = arial_8ptDescriptors;
      fontKern = 1;
      break;
    case COMICS_8:
      fontData = comicSansMS_8ptBitmaps;
      fontDesc = comicSansMS_8ptDescriptors;
      fontKern = 1;
      break;
    case TEST:
```

Also note that the default font defined in the constructor function may be changed as required.

You’re done! Compile and send to your Spark to have hours of viewing enjoyment! ☺