Plusieurs modules sont disponibles dans le commerces pour l'écran OLED DSD TECH I2C 1,3 pouce. Certains utilisent les trois lignes RES, DC et CS. Le module utilisé ici n'utilise que la ligne I2C sur deux fils au standard Arduino c'est à dire SDA sur **A4** et SCL sur **A5**. Comme la majorité des afficheurs OLED sont basés sur le circuit électroniques intégré SSH1106, la "library" fournie peut en gérer un très grand nombre. Les exemples fournis comportent pas moins de 94 options relatives à des types spécifiques disponibles sous forme de remarques. Pour celui utilisé il faut valider :

```
U8GLIB_SH1106_128X64 u8g(U8G_I2C_OPT_NONE);
```

NOTE : L'afficheur OLED de 0,96" bicolore est 100% compatible et peut se brancher à la place. La réciproque n'est pas vraie. Le DSD TECH 1,3" ne fonctionne pas avec <Adafruit_ssd1306syp.h>.

➤ *Boucle d'affichage.*

La mémoire d'affichage ne peut pas être mise à jour élément par élément. Par exemple on veut dessiner "A", attendre une seconde, puis dessiner "B" à l'écran. En écrivant uniquement "B" à l'écran on efface "A". Il est donc nécessaire de dessiner "A" et "B" puis afficher la page entièrement. Par ailleurs il n'est pas possible de copier des régions dans la mémoire d'affichage ou de la faire défiler.

La procédure pour modifier un affichage est la suivante :

1) Configurer éventuellement les paramètres. *(Texte etc ...)*
2) Déclencher une boucle d'affichage avec u8g.firstPage();
   (Efface le contenu de la mémoire écran.)
3) Débuter la construction image avec l'instruction do.
4) Tracer tous les éléments dans la mémoire écran.
5) Terminer le balayage avec while(u8g.nextPage());

Les variables internes ne sont pas réinitialisées au début du corps de la boucle image. ***On peut donc définir toutes les variables requises*** telles que la police de caractère, sa taille etc ***avant le corps de la boucle image***. Bien que l'écran soit construit parfois plusieurs fois avec presque le même contenu, le taux de rafraîchissement est tout à fait acceptable. Des cadences de plus de 20 fps sont possibles.

Disponible sur : https://bintray.com/olikraus/u8glib/Arduino
Doc. sur : https://github.com/olikraus/u8glib/wiki/userreference#firstpage

Bibliothèque U8glib

Afficheur monochrome OLED 1,3 pouces SSD1306 128 x 64 pixels.

Méthodes de la bibliothèque U8glib.h.
Cette librairie impose la ligne I2C standard sur Arduino, c'est à dire **SDA** sur **A4** et **SCL** **A5**. L'activation du petit programme utilitaire Scanner_I2C.ino a trouvé comme adresse sur la ligne I2C la valeur 0x3C pour l'afficheur.

TABLE DES MATIÈRES

Rafraîchissement des données P02
Initialisations diverses. P03
Affichages textuels P04
Position de référence des textes P05
Calcul de l'interlignage des textes P05
Calcul automatique du positionnement des textes P06
Valeurs des paramètres internes P07
Fonctions d'affichage d'un curseur P08
Afficher le curseur P09
Police font_cursor et font_cursorr. P09
Polices de caractères disponibles P10
Fonctions de tracé d'une image Octets. P12
Fonctions de tracé d'une image Pixels. P13
Fonctions de tracé d'entités graphiques P14
Tracer des points P14
Tracer des lignes P14
Tracer des Rectangles P15
Tracer un triangle plein P15
Tracer des cercles et des quarts de cercles P16
Tracer des disques et des quarts de disques P16
Tracer des ellises et des quarts d'ellipses P17
Affichages de variables P18
Stocker les images en AVR PROGMEM P19
Informations générales P20

➢ Rafraîchissement des données.

void u8g.**firstPage**();

L'appel à cette procédure efface la page et marque le début d'une boucle d'image. Elle ne peut pas être utilisée dans la boucle d'image de même que les boucles d'image ne peuvent pas être imbriquées.

Cette procédure ne réinitialise ni ne modifie aucune valeur interne comme la police de caractères actuelle par exemple. Les paramètres et les propriétés de dessin à la fin du corps d'une boucle d'image sont toujours les mêmes lorsqu'une nouvelle boucle de construction d'image est redémarrée.

Fonction u8g.**nextPage**()

Un appel à cette fonction marque la fin du corps d'une boucle d'image. Elle retourne **0** si la boucle image est terminée et **1** si un nouveau balayage de construction de l'image est requis. Cet appel ne peut pas être utilisé à l'intérieur de la boucle d'image.

Programmation d'une page écran :

```
u8g.firstPage(); // Début de la boucle d'image.
do {  ...
      Instructions de tracé des entités à l'écran.
      ...
while(u8g.nextPage());} // Fin de la boucle d'image.
```

NOTE : D'une façon générale les procédures de cette bibliothèque sont du type u8g.**draw**Xxx(*Paramètres*), u8g.**set**Yyy(*Paramètres*) ou des fonctions du genre u8g.**get**Paramètre().

NOTE : Quand on inscrit des pixels sur l'écran, ils se superposeront en bleu ou en noir à ceux déjà présents. On peut donc "surcharger".

---

void u8g.**sleepOn**(); void u8g.**sleepOff**();

Active ou désactive le mode VEILLE pour l'afficheur. Semble se désactiver si une procédure de rafraichissement de page est invoquée. Il est donc conseillé de passer en sommeil et de ne pas provoquer de boucle d'affichage jusqu'au réveil. Ce n'est pas un effacement de la mémoire écran. Ne fait qu'éteindre l'afficheur, son contenu est conservé au moment du réveil.

```
void setup() {PAGE1();} // Appel à la boucle de construction écran.
void loop() {u8g.sleepOn(); delay(20); u8g.sleepOff(); delay(30);}
```


➢ Stocker les images en AVR PROGMEM.

u8g.**drawBitmap**(...); et u8g.**drawBitmapP**(...);
u8g.**drawXBM**(...); et u8g.**drawXBMP**(...);

Dans la plupart des cas, il sera préférable de placer l'image Bitmap dans la zone AVR PROGMEM. Ainsi elle n'encombrera pas la mémoire dynamique. Il faut ajoutez U8G_PROGMEM après la définition du tableau et avant la séquence d'initialisation.
Avec cette modification, appeler alors la variante draw...P.

```
Exemple de codage :
byte IMAGE1[ ] U8G_PROGMEM = {B10101010,
B10101010, B10101010,  B11111111, B11111111, ...
B11111111};
void Trace_image() { u8g.firstPage();
  do {u8g.drawBitmapP( 26, 14, 3, 9, IMAGE1);
```

Dans le démonstrateur PAPILLON.ino la description de l'image est codée sur 584 octets. En fonction des deux versions on obtient :

Version	drawXBM()	drawXBMP()
Programme	5186 octets 16%	5190 octets 16%
MEM dynamique	822 octets 11%	238 octets 11%

On constate que la taille du programme ne change pas, mais que l'occupation de la mémoire dynamique a bien diminué de 584 octets.

---

void u8g.**drawStrP90**(**X**, **Y**, "Chaîne de caractères");

L'argument **P** est supposé faire pointer vers une chaîne qui sera située dans la zone AVR PROGMEM. Il importe dans ce cas de déclarer la chaîne avec U8G_PROGMEM, mais aucun exemple correct n'est disponible dans la documentation ou sur Internet. De ce fait aucune tentative de validation n'a fonctionné correctement.

Fonction u8g.**getColorIndex**();

Fonction qui renvoie la valeur de l'indice de couleur actuel. *(Routine qui fonctionne mais peu utile.)*

Fonction u8g.**getMode**();

Renvoie la valeur du mode d'affichage. Cette donnée peut servir pour extraire le nombre de bits par pixel. *(Peu utile.)*

➢ *Affichages de variables.*

Bien que la procédure u8g.**Print**(Identificateur) puisse afficher des constantes de type chaînes ou caractères, elle est surtout conçue pour afficher des entités variables dans le format de notre choix. Cette subroutine se comporte globalement comme le classique print standard. Toutes les chaînes et valeurs transmises à la procédure d'impression sont ***écrites dans la "position actuelle de l'affichage"*** : Divers appels successifs à u8g.**Print**("Chaîne") sans prépositionnements initiaux affichent les textes les uns à la suite des autres "au kilomètre". Les débordements ne sont pas gérés, les textes hors limites ne seront pas affichés.

void u8g.**setPrintPos**(**X**, **Y**);
Assigne la position (**X**, **Y**) pour le prochain appel de la procédure d'impression de type Print.

void u8g.**Print**(**Identificateur**); *(Constante ou variable quelconque)*
Invoque la procédure d'impression de la classe de base Print. Se comporte de manière similaire à ***u8g.drawStr*** dont tous les paramètres de police s'appliquent également à cette procédure. L'écriture u8g.**Println**() est acceptée mais **ln** n'a aucun effet. En standard les valeurs numériques peuvent se coder en base 2, 10 ou 16, et sont transposées en écriture décimale. Réciproquement on peut imposer les classiques conversions inverses, limiter le nombre de décimales sur les float etc. Exemples :

```
u8g.print(' '); affichera  un espace   ( ' ' ou " " acceptés)
u8g.println(73); affichera 73    (ln est ignoré.)
u8g.print(0x20); affichera 32    (Valeur exprimée en Hexa.)
u8g.print(B011); affichera 3     (Valeur donnée en Binaire.)
u8g.print("ABC"); affichera  le texte entre les ". @
u8g.print(123.456,1); affichera  123,5  (Valeur arrondie.)
u8g.print(73,HEX); affichera  49   (73 converti en Hexa.)
u8g.print(123,BIN); affichera  1111011   (Binaire.)
```

Les affichages de u8g.**Print**(**...**) tiennent compte d'éventuelles directives qui précèdent l'instruction comme un changement d'orientation de l'écran avec ***u8g.setRot90();*** ou d'un changement d'échelle de construction de la page-écran avec ***u8g.setScale2x2()***.

@ : Plus commode que drawStr quand on veut afficher "en ligne" des entités sans avoir à indiquer à chaque fois leurs coordonnées.

➢ *Initialisations diverses.*

void u8g.**setColorIndex**(**1**);
"L'indice de couleur" est utilisé par toutes les procédures de tracé pour définir une valeur de pixel sur l'affichage. Pour un affichage monochrome tel que celui du modèle approvisionné, l'indice de couleur **0** fait tracer en noir, alors que **1** allume les pixels en bleu.

void u8g.**setScale2x2**();
Cette commande ***multiplie par deux les dimensions*** L et H de tout ce qui sera construit ensuite jusqu'à rencontrer la commande d'annulation undoScale(), y compris les coordonnées de position. GetHeight() et getWidth() ne renvoient que la moitié des valeurs caractéristiques de l'afficheur. Les positionnement des éléments seront forcément "paires" puisqu'il y a doublement des entités.

void u8g.**undoScale**();
Cette instruction annule le doublement géométrique des affichages initialisé par setScale2x2().

void u8g.**setRot90**(); ou u8g.**setRot180**(); ou u8g.**setRot270**();
Fait pivoter le sens des aiguilles d'une montre ***l'intégralité de l'affichage de l'écran*** de l'angle indiqué. *(90°, 180° ou 270°.)*
Généralement on adopte en standard un affichage de ***type paysage*** qui est le mode par défaut. Une rotation de 90 ou 270 engendrera un affichage en ***mode portrait***.
NOTE : Les attribus du type u8g.drawStr90(...) sont conservés par rapport à la nouvelle orientation de l'écran. Pour l'exemple choisi le texte serait vertical en relatif par rapport au "portrait". On peut à l'affichage de chaque page en changer librement et individuellement l'orientation relative par rapport "au cadre".

void u8g.**undoRotation**();
Cette instruction supprime une rotation de l'écran initiée par la procédure u8g.setRot*NN*(). Après avoir appelé cette commande, l'affichage aura son orientation "paysage par défaut. Rien ne se passe si l'affichage présente déjà l'orientation par défaut.

void u8g.**setContrast**(**N**)
void u8g.**setRGB**(**N**, **N**, **N**)
void u8g.**setDefautbackgroundColor**()
void u8g.**setDefautForegroundColor**()
void u8g.**setDefautMidColor**()

☞ n'ont aucun effet sur l'afficheur utilisé.

➢ *Affichages textuels.*

void u8g.**drawStr90**(X, Y, "Chaîne de caractères");

Dessine une chaîne à la position **X**, **Y** spécifiée. La position **X**, **Y** correspond au coin inférieur gauche du premier caractère de la chaîne. Il est nécessaire de déclarer la police de caractère utilisée avant le premier appel à cette procédure. Cette procédure utilise également l'index de couleur actuel pour dessiner les caractères. Pour un affichage monochrome, l'indice de couleur **0** supprime généralement un pixel et l'indice de couleur **1** définit un pixel.

Les arguments (**X**, **Y**) sont influencés par le mode de calcul du point de référence paramétré avec setFontPos*REF*. L'affichage du texte tiendra également compte des variantes **90**, **180**, **270** pour orienter l'affichage du texte de 90, 180 ou 270 degrés. *(Seul le texte est concerné par la rotation, l'argument est indépendant d'une rotation de l'affichage complet initié par* ***u8g.setRot****Angle.)*

ATTENTION : ***Aucun texte ne sera affiché*** par drawStr ***si une police de caractère n'a pas été préalablement invoquée*** par setFont, directive qui fait compiler les tables de caractères.

void u8g.**setFont**(*Nom_de_la_police_à_utiliser*);

Directive qui impose l'usage d'une police de caractère particulière. La police invoquée sera utilisée pour toute procédure d'affichage de textes qui suit. La bibliothèque U8glib a beaucoup de polices intégrées qui peuvent être utilisées comme arguments.

```
Exemple de polices :
u8g.setFont(u8g_font_4x6); // Plus petite des polices.
u8g.setFont(u8g_font_6x10);
u8g.setFont(u8g_font_9x15);
u8g.setFont(u8g_font_osb18);
u8g.setFont(u8g_font_osb21); // Police très grande.
```

On peut changer librement de police de caractère n'importe où dans le programme. **Attention :** Chaque police utilisée est intégralement ajoutée dans la mémoire dynamique et prend beaucoup de place.

void u8g.**drawStrAnglP**(...) : L'argument **P** est supposé faire pointer vers une chaîne qui sera située dans la zone AVR PROGMEM. Mais aucune information n'est disponible. *(Voir p19)*

➢ *Tracer des ellises et des quarts d'ellipses.*

void u8g.**drawEllipse**(Xc,Yc,Rx,Ry *Option*);

Dessine une ***ellipse "creuse"*** centrée sur (Xc,Yc) avec le rayon Rx pour demi-axe horizontal et Ry pour demi-axe vertical exprimés en Pixels et construit "contre le centre". Les diamètres ont donc pour valeur (2 x **R**) + 1 Pixels. En fonction de l'*Option* éventuelle, il est possible de ne dessiner qu'un quart de l'ellipse. Les valeurs pour l'*Option* sont données dans le tableau ci-dessus et peuvent être combinées avec l'opérateur **OU** codé avec le symbole | en C++. Naturellement le centre n'est pas visible, seul le périmètre est tracé.

U8G_DRAW_UPPER_RIGHT,
U8G_DRAW_UPPER_LEFT,
U8G_DRAW_LOWER_RIGHT,
U8G_DRAW_LOWER_LEFT,
U8G_DRAW_ALL. *(Complet.)*

void u8g.**drawFilledEllipse**(Xc,Yc,Rx,Ry,*Option*);

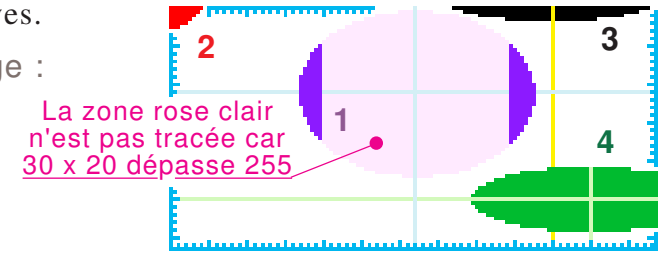
Dessine une ***"ellipse pleine"*** centrée sur (Xc,Yc) avec le rayon Rx pour demi-axe horizontal et Ry pour demi-axe vertical exprimés en Pixels et construit "contre le centre". Les diamètres ont donc pour valeur (2 x **R**) + 1 Pixels. En fonction de l'*Option* éventuelle, il est possible de ne dessiner qu'un quart de l'ellipse. Les valeurs pour l'*Option* sont données dans le tableau ci-dessus et peuvent être combinées avec l'opérateur **OU** codé avec le symbole | en C++.

ATTENTION : Pour ces deux procédures le produit (Rx * Ry) ***doit être inférieur à 255*** car traité en mode 8 bits par u8glib.

*(Concrètement la documentation précise que (**Rx** * **Ry**) doit être inférieurs à **1024** mais l'expérience montre qu'au dessus de la valeur de 255 indiquée dans l'encadré les résultats sont incorrects.)*

Les ellipses pleines et creuses, les quarts géométriques peuvent être en dehors des limites d'affichage. Les coordonnées du centre Xc et Yc peuvent être négatives.

Exemple de codage :
1 : (64,22, 30,20);
2 : (-10,-5, 20,12);
3 : (100,-5, 30,8);
4 : (110,50, 30,8);



➢ Tracer des cercles et des quarts de cercles.

void u8g.**drawCircle**(**Xc**,**Yc**,**R**, *Option*);

Trace un cercle centré sur (**Xc**,**Yc**) avec le rayon **R** exprimé en Pixels et construit "contre le centre". Le diamètre du cercle est donc de (2 x **R**) + 1 Pixels. En fonction de l'*Option* éventuelle, il est possible de ne dessiner qu'un quart du cercle. Les valeurs pour l'*Option* sont données dans le tableau ci-dessus et peuvent être combinées avec l'opérateur **OU** codé avec le symbole | en C++. Naturellement les centres ne sont pas visibles, seul le périmètre est tracé.

U8G_DRAW_UPPER_RIGHT, U8G_DRAW_UPPER_LEFT, U8G_DRAW_LOWER_RIGHT, U8G_DRAW_LOWER_LEFT, U8G_DRAW_ALL. *(Complet.)*

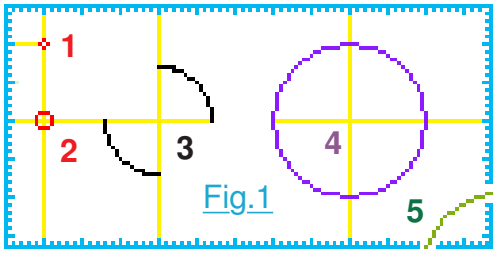


Fig.1

Exemple de codage :

1 : (10,10,1);
2 : (10,30,2); *(Voir la Fig.1)*
3 : (40, 30, 14, ...DRAW_UPPER_RIGHT | DRAW_LOWER_LEFT);
4 : (90,30,21,U8G_DRAW_ALL); *(Cercle complet;)*
5 : (130,70,21,U8G_DRAW_UPPER_LEFT); *(Hors limites;)*

➢ Tracer des disques et des quarts de disques.

void u8g.**drawDisc**(**Xc**,**Yc**,**R**, *Option*);

Tracez un disque centré sur (**Xc**,**Yc**) avec le rayon **R** exprimé en Pixels et tracé "contre le centre". Le diamètre du disque est donc de (2 x **R**) + 1. Selon l'*Option* éventuelle, il est possible de ne dessiner qu'un quart du disque. Les valeurs pour l'*Option* sont identiques à celles pour le cercle. Elles peuvent être combinées avec l'opérateur **OU** codé avec | en C++. *(Les centres sont tracés.)*

Note : l'option U8G_DRAW_ALL n'est pas utile quand il s'agit de tracer un cercle complet. Elle ne sera pertinente que dans des options particulières. *(Un switch / case par exemple.)*

Les cercles, les disques et les quarts de cercle ou de disques peuvent être en dehors des limites d'affichage. Les coordonnées du centre de l'élément géométrique (**Xc**,**Yc**) peuvent être négatives.

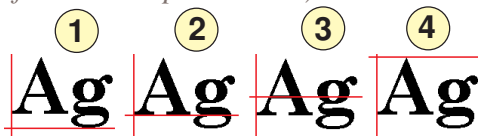
➢ Position de référence des textes.

Intrinsèquement l'instruction **setFontPos** permet de décaler en hauteur la ligne de texte, les coordonnées de u8g.**drawStr**(...) restant inchangées. On peut ainsi décaler verticalement une zone d'affichage dans un texte "global" pour la mettre en évidence.

void u8g.**setFontPosBottom**(); *(Bas : **1**)*
void u8g.**setFontPosBaseline**(); *(Option par défaut : **2**)*
void u8g.**setFontPosCenter**(); *(Centre : **3**))*
void u8g.**setFontPosTop**(); *(Haut : **4**))*

Procédures qui définissent la référence en hauteur pour le tracé des caractères et des chaînes pour u8g.**DrawStr** (X, Y, "Ag").

Bottom : X et Y imposent getFontDescent() sous la ligne de base.
Baseline : X et Y définissent le début gauche de la ligne de base.
Center : La position de référence centrée par rapport aux valeurs de getFontAscent() et de getFontDescent().
Top : La position de référence est getFontAscent() + 1 au-dessus de la ligne de base. *(Soit un pixel au-dessus du caractère de référence le plus haut.)*



Utilisation libre soit dans une Boucle de construction d'image, ou à l'extérieur.

➢ Calcul de l'interlignage des textes.

void u8g.**setFontLineSpacingFactor**(**Coefficient**);

Précise le facteur de proportion pour calculer automatiquement l'interlignage d'un texte. La valeur à affecter au Coeficient sera comprise de 32 à 128 pour avoir une plage d'interlignage comprise entre la moitié et le double de l'espace par défaut de la police de caractère active. Le tableau indique les valeurs à adopter.

Proportion	0.5	0.8	1	1.2	1.5	2
Coefficient	32	51	64	77	96	128

Cette instruction peut être placée soit dans le corps d'une Boucle de construction de "page image", ou à l'extérieur.
Cette procédure tient compte d'une activation de l'instruction u8g.setScale2x2() et de l'usage éventuel de u8g.**setRot90**() ...

Calcul automatique du positionnement des textes.

Ces techniques ne sont pertinentes que si l'on désire réaliser un programme portable sur différents types d'afficheurs présentant des caractéristiques différentes. Elles ne sont présentées dans ces deux pages que pour expliciter toutes les possibilités de la bibliothèque, mais ***peu utiles pour un seul type envisagé***.
Dans l'exemple qui suit, une police est invoquée en ①, puis en ② on adopte un écran "vertical" avec en ③ un affichage en "taille double". En ④ on désire un interlignage de facteur 1,5 avec en ⑤ une "référence au sommet". *(Voir la page 5.)* Enfin en ⑥ / ⑦ on calcule le ***nombre maximum de lignes affichables en hauteur***.

```
① u8g.setFont(u8g_font_6x10);
② u8g.setRot90(); // Option PORTRAIT.
③ u8g.setScale2x2(); // Option double taille.
④ u8g.setFontLineSpacingFactor(96);
⑤ u8g.setFontPosTop();
⑥ Nb_lignes_Max =
⑦   (u8g.getHeight() / u8g.getFontLineSpacing()) -1;
```

Pour calculer automatiquement les paramètres à utiliser par le programme, diverses fonctions ou procédure sont disponibles :

void u8g.setFontRefHeightAll();
Les fonctions Ascent et Descent sélectionneront la plus grande valeur de tous les glyphes de la police actuelle.

void u8g.setFontRefHeightExtendedText();
Ascent adoptera le dépassement du 'A', du '1' ou de '(' de la police actuelle. Descent utilisera le 'g' ou '(' de la police actuelle. C'est l'option par défaut après le redémarrage du programme.

void u8g.setFontRefHeightText();
Ascent n'utilise que le 'A' ou le '1' de la police actuelle. Descent ne prend en compte que le 'g' de la police actuelle.

Un appel à l'une de ces trois procédures imposera la méthode de calcul pour les décalages en hauteur d'affichage des textes. Cette méthode sera utilisée pour la police actuelle et toutes les autres qui seront invoquées avec setFont(). La sélection de la méthode de calcul a un effet sur getFontAscent() et getFontDescent() ainsi que sur les procédures autres que setFontPosBaseline ().

➢ Tracer des Rectangles.

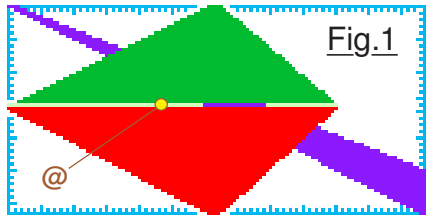
void u8g.**drawRFrame**(XG,YH,L,H,r);
Trace un ***rectangle vide*** en partant du coin supérieur gauche de coordonnées (XG,YH). La largeur L et la hauteur H sont exprimées en Pixels. Le coin en bas et à droite peut être en dehors des limites d'affichage. Les coordonnées (XG,YH) peuvent être négatives.

void u8g.**drawRBox**(XG,YH,L,H,r);
Trace un ***rectangle plein*** en partant du coin supérieur gauche de coordonnées (XG,YH). La largeur L et la hauteur H sont exprimées en Pixels. Les coins en haut et en bas peuvent être en dehors des limites d'affichage et (XG,YH) peuvent être négatives.

Avec l'option R les rectangles seront tracés avec des congés de raccordement dont le rayon r est exprimé en Pixels. Il importe que L ≥ (r + 1) x 2 et H ≥ (r + 1) x 2. Si cette condition n'est pas vérifiée le comportement n'est pas défini. *(Certains essais semblent montrer que pour RBox il faut ≥ ((r + 1) x 2) + 1.)*

➢ Tracer un triangle plein.

void u8g.**drawtriangle**(XA,YA, XB,YB, XC,YC);
Dessine un triangle ABC dont les coordonnées sont exprimées sur 16 bits et le polygone est limité à la fenêtre de l'afficheur. Les coordonnées (XA,YA, XB,YB, XC,YC) peuvent être négatives. ***Si plusieurs polygones triangles sont dessinés avec coïncidence de l'un des cotés, il n'y aura pas de chevauchement.*** *(Génère une discontinuité.)* Idem si deux arêtes sont voisines, une séparation sera visible entre les deux. Dans l'exemple montré sur la Fig.1 la ligne vert clair n'est pas tracée alors qu'elle est définie dans le triangle vert foncé. *(Elle jouxte le triangle rouge.)*



Exemple de codage :

```
u8g.drawTriangle(64,0, 0,30, 100,30);
u8g.drawTriangle(0, 31, 100,31, 64,60);
u8g.drawTriangle(0, 0, 127,50, 127,64);
```

@ : La ligne théorique verte n'est pas tracée, mais le triangle violet est construit car non contigu.

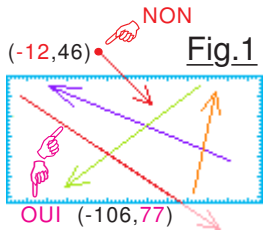
Fonctions de tracé d'entités graphiques.

Globalement toutes les procédures de construction graphique sont de structure u8g.drawEntité(Paramètres); le nombre de paramètres étant fonction des caractéristiques de l'élément à placer dans la page-écran. Pour mémoire, toutes ces procédures utilisent l'indice de setColorIndex actuel pour dessiner les points. Pour un afficheur OLED monochrome, l'indice de couleur **0** fait tracer en noir, alors que **1** allume les pixels en bleu. Toutes les constructions prendront en compte un éventuel u8g.setScale2x2(). Toutes les instructions de construction graphiques doivent se trouver dans une boucle de construction d'une page écran ou elles seront sans effet autre que consommer inutilement du code objet.

void u8g.**drawPixel**(**X**,**Y**);

Dessine un point à la position **X** / **Y** spécifiée. La position (0,0) est située dans le coin supérieur gauche de l'écran. La position peut sans engendrer d'aléas être en dehors des limites d'affichage.

➢ *Tracer des lignes.*



void u8g.**drawLine**(**XD**,**YD**,**XF**,**YF**);

Trace une ligne du point de Début de coordonnées (**XD**,**YD**) au point Final de position (**XF**,**YF**). Il n'y a pas de restrictions pour l'ordre du paramétrage, on peut tracer dans tous les sens. Les coordonnées peuvent se trouver hors page écran, mais ***elles doivent toutes être positives***. *(Voir la Fig.1)* Par exemple u8g.drawLine(3, 3, 1, 15000) tracera un trait vertical.

void u8g.**drawHLine**(**XD**,**YD**,**L**);

Trace une ligne **H**orizontale, la position de D***ébut*** de coordonnées (**XD**,**YD**) est ***situé à gauche***. La longueur **L** de la ligne est explimée en pixels et elle peut déborder à droite. **XD** négative est autorisé.

void u8g.**drawVLine**(**XD**,**YD**,**L**);

Trace une ligne **V**erticale, la position de D***ébut*** de coordonnées (**XD**,**YD**) est ***situé en haut***. La longueur **L** de la ligne est exprimée en pixels et elle peut déborder en bas. **YD** négative est autorisé.

NOTE : Il semble que certaines valeurs dépassant 128 pour les longueurs engendrent un "repliement sur le coté opposé".

➢ *Valeurs des paramètres internes.*

Fonction u8g.**getHeight**(); ou ***Fonction*** u8g.**getWidth**();

Retourne la définition en ***hauteur*** ou celle en ***largeur*** de l'afficheur utilisé. ATTENTION : La valeur retournée prend en compte une éventuelle rotation d'écran par l'instruction setRot*NN*().

```
Exemple de codage :
u8g.setPrintPos(30, 50); u8g.print(u8g.getHeight());
```

Fonction u8g.**getStrWidth**(Chaîne);

Retourne la longueur d'une chaine constante ou variable.

```
Exemple :
Marge = (128-u8g.getStrWidth(MENU[I]))/2;
Nb_colonnes_Max = u8g.getWidth() / u8g.getStrWidth("m");
```

Fonction u8g.**getFontLineSpacing**();

Retourne la distance verticale de deux lignes de texte, écrites avec la police actuelle. Cette valeur est dérivée des valeurs de FontAscent, de FontDescent et multipliée par le "LSF" actuel. La valeur renvoyée est influencée par la police imposée, la "Hauteur de référence" et le "LSF". *(LSF : LineSpacingFactor.)* Exemple :

```
Nb_lignes_Max = u8g.getHeight() / u8g.getFontLineSpacing();
```

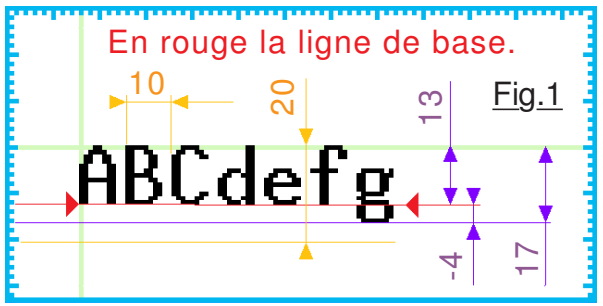
Fonction u8g.**getFontAscent**();

Retourne la hauteur de dépassement des matrices de caractères au-dessus de la ligne de base. Cette valeur dépend de la hauteur de référence actuelle. *(Voir setFontRefHeightAll et la Fig.1)*

Fonction u8g.**getFontDescent**();

Indique la hauteur de dépassement des matrices de caractère au-dessous de la ligne de base. *(Voir la Fig.1)*

```
Exemple de codage :
u8g.setFont(u8g_font_10x20);
u8g.drawStr(16, 30, "ABCdefg");
```



Avec cet exemple la police de caractères utilisée construit ses textes avec des matrices de 10 x 20 pixels. Ascent retourne 13, Descent renvoie -4, FLS indique 17. *(FontLineSpacing.)*

Fonctions d'affichage d'un curseur.

C'est le centre de la matrice qui sera positionnée aux coordonnées de u8g.setCursorPos(X,Y); de façon à pointer correctement sur l'écran quelle que soit la forme représentée. Chaque empreinte est une matrice qui peut aller jusqu'à 31 x 31 pixels. Si des pixels sont à l'état allumé sous le curseur, il n'y aura pas forcément superposition, car le programme éteint certaines de ses zones non allumées en fonction de l'empreinte.

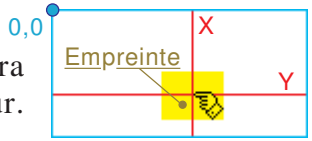
➢ Initialisations diverses.

void u8g.**SetCursorColor**(**AP**,**C**); @

Comme pour setColorIndex(N) la valeur **0** affectée aux paramètres éteindra les pixels alors que **1** les allumera. Le paramètre **AP** agit sur l'Arrière Plan alors que **C** conditionne le Curseur. On peut ainsi afficher en "standard" ou en image inversée. (1)

void u8g.**SetCursorPos**(**X**,**Y**);

Précise les coordonnées à l'écran où sera positionné le centre de l'empreinte du curseur.



void u8g.**SetCursorFont**(**u8g_font_cursorr**); @

Précise la table des empreintes qui sera installée dans le programme. Consulter le chapitre *Police font_cursor et font_cursorr*.

~~void u8g.**enableCursor**();~~

~~Active le curseur à la position spécifiée.~~ (2)

void u8g.**drawCursor**();

Bien que non spécifié dans la documentation, affiche le curseur et fonctionne systématiquement. La taille du curseur prendra en compte une éventuelle instruction u8g.setScale2x2().

~~void u8g.**disableCursor**();~~

~~Désactiver le curseur. Le curseur ne sera pas visible.~~

~~*(Bien qu'accepté par le compilateur, ne produit aucun effet.)*~~

@ : Cette instruction peut n'être utilisée qu'une seule fois et placée dans la procédure void **setup**().

(1) : u8g.SetCursorColor(**1**,**0**) fonctionne correctement, mais u8g.SetCursorColor(**0**,**1**) décale parfois les formes des empreintes.

(2) : Ne fonctionne pas systématiquement.

Fonctions de tracé d'une image Pixels.

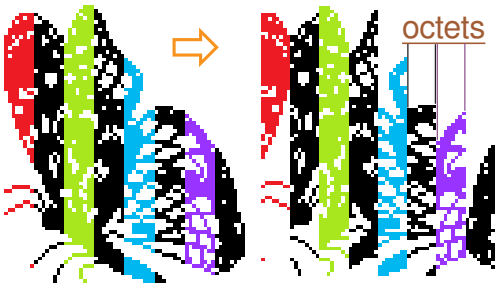
C'est le coin supérieur gauche qui sera positionnée aux coordonnées de u8g.drawXBMP(X, Y, L, H, Matrice) quelle que soit la ***forme du balayage*** du dessin. L'image sera définie dans un tableau exploré "horizontalement" dont les OCTETs représenteront les divers points de l'image.

void u8g.**drawXBMP**(**X**,**Y**,**L**,**H**,**Matrice**);

Dessine un bitmap à la position spécifiée par **X** et **Y** pour le coin supérieur gauche du bitmap. La zone du tracé peut déborder les limites d'affichage. **Cette procédure ne dessine que les valeurs de Pixel à 1. Les Pixels codés avec la valeur 0 ne sont pas dessinés.** *(Ils sont transparents.)* Le paramètre **L** indique le nombre de Pixels pour la largeur du dessin tracé. Avec **H** on précise le nombre de Pixels pris en compte pour la hauteur. *(On peut en afficher plus ou moins que contenus dans la matrice du dessin.)*

NOTE : La taille du dessin tracé prendra en compte une éventuelle instruction u8g.setScale2x2().

IMPORTANT : Le balayage de la **Matrice** se fait du Bit n°0 en "montant" dans la zone mémoire du tableau représentatif. Le tracé à l'écran puise Bit par Bit et trace les Pixels par un balayage horizontal effectué de la gauche vers la droite et verticalement par un tracé dirigé du haut vers le bas. ***Il faut "retourner" les Octets pour coder la matrice du dessin.***



La déclaration du tableau qui définit le dessin doit obligatoirement comporter U8G_PROGMEM avant le listage des valeurs si XBMP où le résultat obtenu ne sera pas celui attendu. Les valeurs peuvent librement être définies en décimal, en Hexadécimal ou en Binaire et exprimées dans ces trois bases au sein du listage :

byte PAPILLON[ ] U8G_PROGMEM = {....}

En décimal : {255, ... 170, 85}
En Hexadécimal : {0xFF, 0xAA, 0x55,}
En Binaire : {B11111111, ... B10101010, B01010101}

Fonctions de tracé d'une image Octets.

C'est le coin supérieur gauche qui sera positionnée aux coordonnées de u8g.drawBitmapP(X, Y, L, H, Tableau) quelle que soit la *forme du balayage* du dessin. L'image sera définie dans un tableau exploré "horizontalement"dont les OCTETs représenteront les divers points de l'image.

void u8g.**drawBitmapP**(**X**,**Y**,**L**,**H**,**Tableau**);

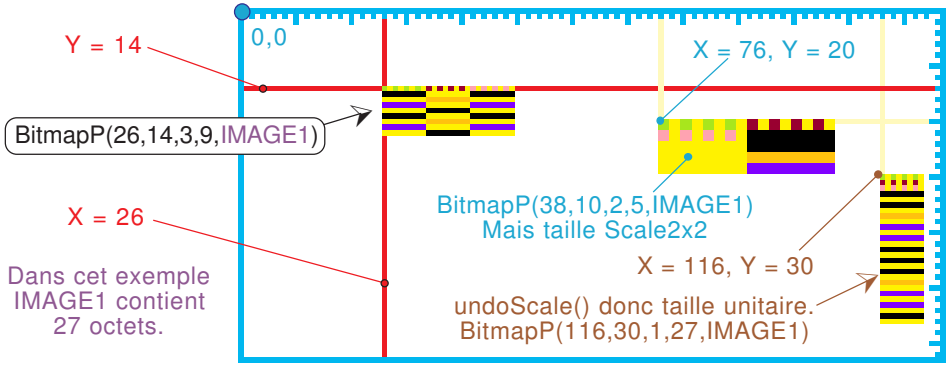
Dessine un bitmap à la position spécifiée par **X** et **Y** pour le coin supérieur gauche du bitmap. Certaines parties du tracé peuvent être en dehors des limites d'affichage. Le bitmap est décrit dans un tableau de type bytes. Chaque BIT à **0** dans les octets du **Tableau** correspond à "noir",et **1** allumera le point. Le paramètre **L** indique le nombre d'octets pour la largeur du dessin tracé. *(Balayage du tableau horizontal et gestion verticale à l'écran.)* Avec **H** on précise le nombre d'octets pris en compte pour la hauteur. *(On peut en afficher moins que la taille du tableau n'en autorise.)*
ATTENTION : La taille du dessin tracé prendra en compte une éventuelle instruction u8g.setScale2x2().

Pour balayer la matrice le format est libre, mais L x H ne doit pas dépasser la taille du tableau ou des points tracés seront "aléatoires".

Exemple de codage :

```
byte IMAGE1[ ] U8G_PROGMEM = {B10101010,
B10101010, B10101010,  B11111111, B11111111, ...
B11111111};
void Trace_image() { u8g.firstPage();
  do {u8g.drawBitmapP( 26, 14, 3, 9, IMAGE1);
  u8g.setScale2x2(); u8g.drawBitmapP( 26, 14, 3, 9, IMAGE1);
  ... }  while( u8g.nextPage() );
```

(Voie encadré P13.)



➢ Afficher le curseur.

void u8g.**SetCursorStyle**(**N**); @

Impose dans la police font-cursor quelle empreinte sera utilisée pour dessiner le curseur à la position déjà précisée. *(Voir la table de la Fig.1 donnée ci-dessous)*

L'ordre de CursorStyle et de CursorPos peut être inversé, mais ces deux instructions doivent précéder l'affichage drawCursor. Elles peuvent être définies en dehors de la boucle de création de la page écran. C'est surtout utile pour la forme de l'empreinte setCursorStyle(N) qui peut n'être définie qu'une fois dans le programme si elle ne change pas.

➢ Police font_cursor et font_cursorr.

Utiliser la police complète font_cursor consomme 5286 octets en mémoire de l'ATmega328 alors que l'on peut facilement se contenter d'une police **réduite** à douze empreintes. Dans le tableau de la Fig.1 seules les formes rouges seront disponibles si l'on utilise la police réduite **font_cursorr**. Dans ce cas, faire appel à une empreinte qui n'est pas "rouge" n'affiche pas le curseur.

Fig.1

➢ *Polices de caractères disponibles.*

Déclarer une police de caractère consiste à utiliser la directive u8g.**setFont**(*Nom_de_la_police_à_utiliser*) qui inscrit dans la mémoire de l'ATmega328 la table de description des divers éléments. La toute première déclaration d'une police engendre un cout de 184 octet pour la gestion des textes, augmenté de la taille d'occupation propre à la fonte invoquée. Naturellement le paramètre de la procédure doit correspondre à une police existante. Pour avoir la liste des variantes possibles on peut aller par exemple sur :

https://nodemcu-build.com/u8g-fonts.php#collapseEight

Le site ouvre sur la fenêtre de la Fig.1 listant sur neuf items les groupes possibles des polices qui sont classées par thèmes. Cliquer sur le symbole **1** pour choisir un groupe, action qui déroule sa liste comme en Fig.2, le symbole se change en représentation **2** permettant de refermer la fenêtre contextuelle. Le curseur de la souris **3** se change en une main rouge **4** quand il passe sur une zone de police telle que celle repérée en jaune, ouvrant la zone de description de sa fonte de caractères. En rouge figure le nom exact à passer en paramètres à **setFont**(). Dans la zone bleue sont précisées la largeur et la hauteur des

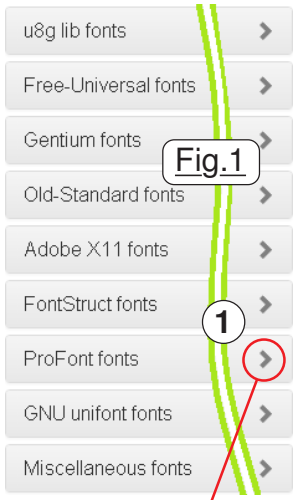


Fig.1

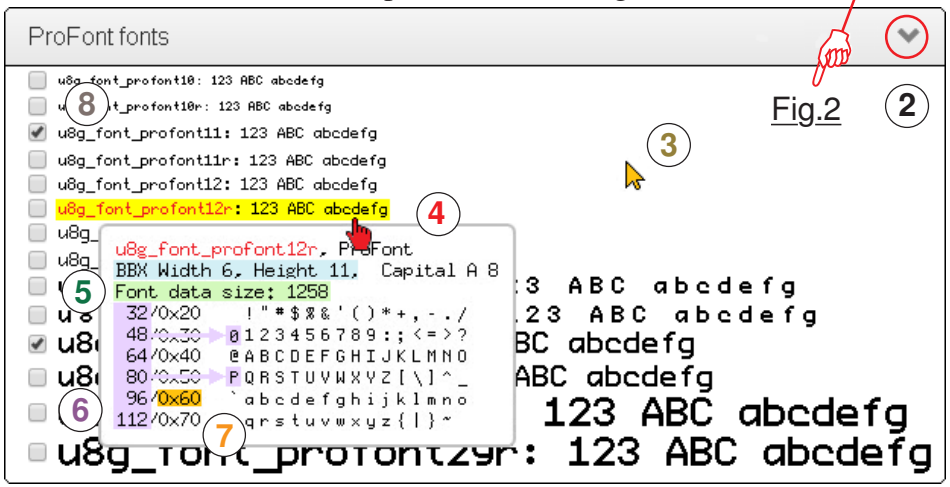


Fig.2

caractères, valeurs qui seraient retournées respectivement par les fonctions u8g.getStrWidth("A") et u8g.getFontLineSpacing(). Noter que pour la largeur, bien que l'on utilise qu'un seul caractère, c'est une chaîne qu'il faut déclarer comme paramètre.
En **5** dans la zone verte est indiquée ***la taille*** exprimée en octets ***occupée par la police*** en mémoire de l'ATmega328 quand elle est déclarée dans le programme.
En **6** on trouve la valeur du ***code du premier caractère de la ligne*** visualisée. Dans l'exemple 48 sera le code pour '0' et 80 celui pour la lettre 'P'. En zone orange **7** voisine également la transposition en Hexadécimal du code caractère.

```
u8g.setPrintPos(40, 20); u8g.print(char(48)); // Affiche un zéro.
u8g.setPrintPos(40, 40); u8g.print(char(0x50)); // Affiche  'P'.
```

Utiliser un char dans la construction d'un texte sera impératif pour pouvoir afficher correctement les caractères spéciaux tels que les accentués. Par exemple :

u8g.print("Caractère."); affichera "CaractÃ¨re.".

Pour construire correctement la page écran il faut :
• Utiliser une fonte qui contient les caractères spéciaux désirés,
• Imposer le caractère souhaité en char et préciser son code :

```
EXEMPLE :
u8g.setFont(u8g_font_profont12);
u8g.setPrintPos(40, 20); u8g.print("Caract");
u8g.print(char(0xE8)); u8g.print("re.");
```

ATTENTION : Toutes les fontes n'ont pas forcément les accentués. Donc, ***dès le début du développement d'un programme il sera judicieux de choisir dans l'éventail des possibilités la ou les polices les plus judicieuses***. Noter que dans les fenêtres de spécification des matrices, la taille des éléments visualisés à l'écran est proportionnelle à la taille des effets produits sur l'afficheur OLED.

Les cases dans la colonne 8 peuvent être cochées pour indexer à convenance des polices désirées. On clique alors sur Apply *pour les "valider".* ***Instructions*** *précise : "Cette fenêtre se fermera et remplira le champ de texte des polices u8g dans la page parente". Il semble que cette commande soit sans effet, donc l'ignorer.*