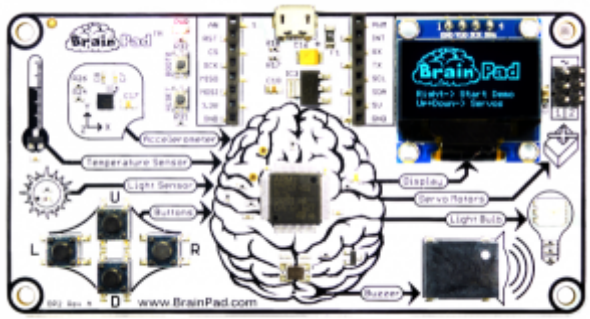
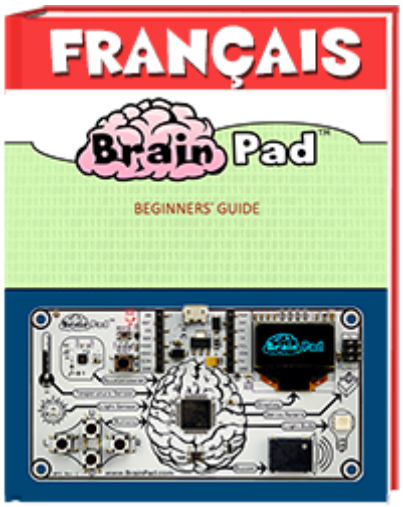
Les classes de la carte BrainPad 2

[Mise à jour le 4/9/2020]

La carte BrainPad 2



Un guide en français pour débuter



Liste des classes de la bibliothèque BrainPad

	Nom	Description
S	**Accelerometer**	Accès à l'accéléromètre MMA8453Q de la carte. Lecture de l'accélération sur les axes X, Y et Z.
S	**Buttons**	Accès aux 4 touches de la carte par scrutation ou par événements.
S	**Buzzer**	Accès au buzzer (déclenchement, arrêt, génération d'une note (fréquence), diffusion de musique)
S	**Display**	Contrôle de l'afficheur monochrome (128 x 64) à circuit SSD1306 (I2C). Texte, tracé de forme géométrique etc. **Ressource documentaire** : Les afficheurs graphiques
S	**Expansion**	Définition des entrées / sorties du connecteur d'extension.
S	**LightBulb**	Accès à la Led RVB haute luminosité.

	Nom	Description
S	**LightSensor**	Accès au capteur de luminosité (LDR).
S	**Picture**	Affichage d'une image.
S	**ServoMotors**	Accès aux sorties dédiées à la commande des servomoteurs 1 et 2.
S	**TemperatureSensor**	Accès au capteur analogique de température MCP9701.
S	**Wait**	Offre des méthodes de temporisation (en secondes ou millisecondes)..

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte BrainPad v1 ou v2 "Étape par Étape"

Accelerometer

L'accéléromètre est un périphérique d'entrée qui mesure la **force d'accélération** sur trois axes (**x**, **y** et **z**). On appelle habituellement cette force **g-force** et on l'exprime comme un multiple de la force de gravité. Par exemple, si un pilote d'avion reçoit une force de 2 g dans un virage à 60 degrés, cela signifie qu'il est poussé dans son siège avec une force égale au double de son poids. Les pilotes de chasse peuvent recevoir jusqu'à 9 g. Le BrainPad renvoie une valeur d'accélération égale à la force g multipliée par 100.

Si le BrainPad est à plat sur une table avec l'écran à votre droite, l'axe des x est horizontal de gauche à droite, l'axe des y est horizontale et s'éloigne de vous, et l'axe des z s'étend verticalement de haut en bas. Si le BrainPad est immobile dans cette position, les axes x et y indique 0, mais l'axe z indique -100. En effet, la force de gravité pousse sur l'accéléromètre avec une force égale à 1 g (la force de gravité de la Terre). Si vous retournez le BrainPad, l'axe des z indique 100. Le seul cas où tous les axes sont à zéro est le moment où le BrainPad est en chute libre.

Remarque : la valeur renvoyée par les méthodes ci-dessous est l'accélération multipliée par 100. (1g = 9,81m/s²)

Attribut

	Syntaxe	Description
S	**EnableFullRange**	Si la valeur est définie sur false, les valeurs de l'accéléromètre vont de -100 à 100 (-1 g à 1 g). Si la valeur est true, les valeurs de l'accéléromètre sont comprises entre -200 et 200 (-2 g à 2 g).

ex.cs

```
BrainPad.Accelerometer.EnableFullRange = true;
```

Méthodes

	Syntaxe	Description
S	*int* **ReadX**()	Renvoie une valeur entière comprise entre -100 et 100 (ou -200 à 200 si BrainPad.Accelerometer.EnableFullRange = true) en fonction de l'accélération sur l'axe x.

	Syntaxe	Description
S	*int* **ReadY**()	Renvoie une valeur entière comprise entre -100 et 100 (ou -200 à 200 si BrainPad.Accelerometer.EnableFullRange = true) en fonction de l'accélération sur l'axe y.
S	*int* **ReadZ**()	Renvoie une valeur entière comprise entre -100 et 100 (ou -200 à 200 si BrainPad.Accelerometer.EnableFullRange = true) en fonction de l'accélération sur l'axe z.

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte BrainPad v1 ou v2 "Étape par Étape"

Exemple

Accelerometer.cs

```
static void Main()
{
    BrainPad.Display.Clear();

    while (true)
    {
        BrainPad.Display.Clear();
        BrainPad.Display.DrawSmallText(0, 0, "X : " +
BrainPad.Accelerometer.ReadX().ToString("F2"));
        BrainPad.Display.DrawSmallText(0, 10, "Y : " +
BrainPad.Accelerometer.ReadY().ToString("F2"));
        BrainPad.Display.DrawSmallText(0, 20, "Z : " +
BrainPad.Accelerometer.ReadZ().ToString("F2"));
        BrainPad.Display.RefreshScreen();
        BrainPad.Wait.Milliseconds(100);
    }
}
```

Le projet **TCLR_BRAINPAD_Accel_vx.x.x** est **téléchargeable** ici

---

Buttons

Les quatre **boutons** directionnels (**haut**, **bas**, **gauche** et **droi**t) servent d'entrées et peuvent être lus par votre programme pour déterminer si le bouton est enfoncé ou non. Ils peuvent être testés de deux manières différentes.

- Une solution consiste à **vérifier périodiquement** si le bouton est enfoncé (**scrutation**).

Cela fonctionne bien, mais il est possible de manquer une action sur un bouton si votre programme ne le vérifie pas assez souvent. L'action sur un bouton du clavier peut être déterminée par scrutation à l'aide des méthodes (S) ci-dessous.

Méthodes

	Syntaxe	Description
S	*bool* **IsDownPressed**()	Renvoie la valeur *vrai* si le bouton **Down** a été pressé, la valeur faux sinon.
S	*bool* **IsLeftPressed**()	Renvoie la valeur *vrai* si le bouton **Left** a été pressé, la valeur faux sinon.
S	*bool* **IsRightPressed**()	Renvoie la valeur *vrai* si le bouton **Right** a été pressé, la valeur faux sinon.
S	*bool* **IsUpPressed**()	Renvoie la valeur *vrai* si le bouton **Up** a été pressé, la valeur faux sinon.

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte BrainPad v1 ou v2 "Étape par Étape"

Exemple 1 : scrutation du clavier

Clavier1.cs

```
class Program
{
    static int increment = 0;

    static void Main()
    {
        while (true) // Scrutation du clavier (pas d'anti-rebond)
        {
            if (BrainPad.Buttons.IsUpPressed())
                increment += 10;
            if (BrainPad.Buttons.IsDownPressed())
                increment -= 10;
            BrainPad.Display.Clear();
            BrainPad.Display.DrawSmallText(0, 0, "Valeur: " +
increment.ToString());
            BrainPad.Display.RefreshScreen();
            BrainPad.Wait.Milliseconds(100);
        }
    }
}
```

Le projet **TCLR_BRAINPAD_BP_vx.x.x** est **téléchargeable** ici

- L'autre façon de vérifier les boutons consiste à utiliser un **gestionnaire d'événements**.

Un gestionnaire d'événements vérifiera en permanence si un bouton a été appuyé ou relâché. Une fois qu'un gestionnaire d'événement est configuré, le BrainPad lance un écouteur d'événement qui vérifie le bouton pour vous. Cela permet à votre programme d'effectuer d'autres tâches sans vous soucier de manquer un événement de bouton. Une fois le bouton enfoncé (ou relâché), l'écouteur d'événements appelle votre gestionnaire d'événements. Le gestionnaire d'événements est un code que vous écrivez pour indiquer au BrainPad comment réagir lorsqu'un bouton est enfoncé (ou

relâché). Bien qu'un gestionnaire d'événements soit un peu plus difficile à configurer, il **permet au programme de s'occuper d'autres tâches tout en ne manquant jamais un événement** de bouton. Les **évènements** ( ) disponibles sont listés ci-dessous.

Evènements (type ButtonEventHandler)

	Syntaxe	Description
	WhenUpButtonPressed	Un événement est déclenché quand le bouton **Up** est appuyé.
	WhenUpButtonReleased	Un événement est déclenché quand le bouton **Up** est relâché.
	WhenLeftButtonPressed	Un événement est déclenché quand le bouton **Left** est appuyé.
	WhenLeftButtonReleased	Un événement est déclenché quand le bouton **Left** est relâché.
	WhenRightButtonPressed	Un événement est déclenché quand le bouton **Right** est appuyé.
	WhenRightButtonReleased	Un événement est déclenché quand le bouton **Right** est relâché.
	WhenDownButtonPressed	Un événement est déclenché quand le bouton **Down** est appuyé.
	WhenDownButtonReleased	Un événement est déclenché quand le bouton **Down** est relâché.

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte BrainPad v1 ou v2 "Étape par Étape"

Exemple 2 : gestion du clavier avec des évènements

Clavier2.cs

```
class Program
{
    static void Main()
    {
        BrainPad.Buttons.WhenUpButtonPressed +=
Buttons_WhenUpButtonPressed;
        BrainPad.Buttons.WhenDownButtonPressed +=
Buttons_WhenDownButtonPressed;

        while (true)
        {
            BrainPad.Wait.Minimum(); // Endort régulièrement la tâche
Main
        }
    }

    private static void Buttons_WhenDownButtonPressed()
    {
        BrainPad.LightBulb.TurnOff();
    }

    private static void Buttons_WhenUpButtonPressed()
    {
        BrainPad.LightBulb.TurnBlue();
    }
}
```

Le projet **TCLR_BRAINPAD_BP_Event_vx.x.x** est **téléchargeable** ici

---

Buzzer

Le buzzer est plutôt un petit haut parleur capable de jouer de la musique et même de diffuser une voix.

Méthodes

	Syntaxe	Description
S	*void* **Beep**()	Génère un bip court.
S	*void* **StartBuzzing**(*double* frequency)	Joue un son dont la fréquence prend la valeur *frequency*.
S	*void* **StopBuzzing**()	Coupe le son.

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte BrainPad v1 ou v2 "Étape par Étape"

Exemple 1 : réglage de la fréquence jouée par le buzzer avec les boutons Up et Down

Buzzer1.cs

```
static void Main()
 {
    int frequency = 0, increment = 0;
    BrainPad.Display.Clear();
    BrainPad.Display.DrawSmallText(0, 0, "Frequence = " + frequency);
    BrainPad.Display.RefreshScreen();

    while (true)
    {
        if (BrainPad.Buttons.IsUpPressed())
         {
            increment = 100;
         }
        if (BrainPad.Buttons.IsDownPressed())
         {
            increment = -100;
         }
        if (increment != 0)
         {
            frequency = frequency + increment;
            BrainPad.Display.Clear();
            BrainPad.Display.DrawSmallText(0, 0, "Frequence = " +
frequency);
            BrainPad.Display.RefreshScreen();
            increment = 0;
            BrainPad.Buzzer.StartBuzzing(frequency);
```

```
            BrainPad.Wait.Seconds(0.2);
            BrainPad.Buzzer.StopBuzzing();
        }
    }
}
```

Le projet **TCLR_BRAINPAD_Buzzer1_vx.x.x** est **téléchargeable** ici

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte BrainPad v1 ou v2 "Étape par Étape"

Exemple 2 : Mélodie

Buzzer2.cs

```
static void Main()
 {
    const int NoteC = 261;
    const int NoteD = 294;
    const int NoteE = 330;
    const int NoteF = 349;
    const int NoteG = 391;

    const int Whole = 1000;
    const int Half = Whole / 2;
    const int QuarterDot = Whole / 3;
    const int Quarter = Whole / 4;
    const int Eighth = Whole / 8;

    int[] note = {
            NoteE, NoteE, NoteF, NoteG, NoteG, NoteF, NoteE,
            NoteD, NoteC, NoteC, NoteD, NoteE, NoteE, NoteD,
            NoteE, NoteD, NoteC, NoteC, NoteD, NoteE, NoteD,
            NoteC, NoteC
          };

    int[] duration = {
            Quarter,  Quarter,  Quarter, Quarter, Quarter, Quarter,
            Quarter,  Quarter,  Quarter, Quarter, Quarter, Quarter,
QuarterDot,
            Eighth, Half, Quarter,  Quarter,  Quarter, Quarter,
Quarter, Quarter,
            Quarter,  Quarter,  Quarter, Quarter, Quarter, Quarter,
QuarterDot,
            Eighth,Whole
          };


     while (true)
```

```
        {
          for (int i = 0; i < note.Length; i++)
          {
              BrainPad.Buzzer.StopBuzzing();
              BrainPad.Buzzer.StartBuzzing(note[i]);
              BrainPad.wait(duration[i]);
          }

          Thread.Sleep(100);
        }
}
```

Le projet **TCLR_BRAINPAD_Buzzer2_vx.x.x** est **téléchargeable** ici

---

Display

L'unité d'affichage est le pixel. Les commandes affectent uniquement la mémoire dédiée à l'écran (buffer d'affichage). Rien ne s'affiche tant que *BrainPad.Display.RefreshScreen ()* n'est pas exécuté.

Ressource documentaire : Les afficheurs graphiques

Attributs

	Syntaxe	Description
S	*int* **Height**	Retourne la hauteur de l'affichage en pixels (64).
S	*int* **Width**	Retourne la largeur de l'affichage en pixels (128).

Méthodes

	Syntaxe	Description
S	*void* **DrawSmallNumber**(*int* x, *int* y, *long* number)	Ecrit le nombre **number** à la position (**x,y**) dans un petit format. ***Surchargée*** : *double* number
S	*void* **DrawNumber**(*int* x, *int* y, *long* number)	Ecrit le nombre **number** à la position (**x,y**) dans un format large. ***Surchargée*** : *double* number
S	*void* **DrawSmallText**(*int* x, *int* y, *string* text)	Ecrit le texte **text** à la position (**x,y**) dans un petit format. Les caractères accentués provoquent une exception *System.IndexOutOfRangeException*.
S	*void* **DrawText**(*int* x, *int* y, *string* text)	Ecrit le texte **text** à la position (**x,y**) dans un format large. Les caractères accentués provoquent une exception *System.IndexOutOfRangeException*.

	Syntaxe	Description
S	*void* **DrawScaledText**(*int* x, *int* y, *string* text, *int* HScale, *int* VScale)	Ecrit le texte **text**, pouvant être agrandit, à la position (**x,y**). **HScale** change la hauteur. **VScale** change la largeur. Par exemple HScale = 2 double la largeur du texte à l'écran. - HScale=1, VScale=1 ⇒ DrawSmallText - HScale=2, VScale=2 ⇒ DrawText Les caractères accentués provoquent une exception *System.IndexOutOfRangeException*.
S	*void* **CreatePicture**(*int* width, *int* height,*byte[]* data)	Stocke une image de largeur **width** et de hauteur **height** constituée des n pixels enregistrés sous la forme d'octets dans le tableau **data**. Un pixel prend les valeurs 0 (éteint) ou 1 (éclairé).
S	*void* **CreateScaledPicture**(*int* width, *int* height,*byte[]* data, *int* scale)	Stocke une image susceptible d'une agrandit par le facteur d'échelle **scale**. Par exemple, régler le facteur d'échelle à 2 affichera l'image avec le double de sa taille d'origine.
S	*void* **DrawCircle**(*int* x, *int* y, *int* r)	Dessine un cercle de rayon **r** à la position (**x,y**).
S	*void* **DrawRectangle**(*int* x, *int* y, *int* width, *int* height)	Dessine un rectangle de largeur **width** et de hauteur **height** à la position (**x,y**).
S	*void* **DrawFilledRectangle**(*int* x, *int* y, *int* width, *int* height)	Dessine un rectangle plein de largeur **width** et de hauteur **height** à la position (**x,y**).
S	*void* **DrawLine**(*int* x0, *int* y0, *int* x1, *int* y1)	Dessine une ligne entre la position (**x0,y0**) et (**x1,y1**).
S	*void* **DrawPoint**(*int* x, *int* y)	Dessine un point à la position (**x,y**).
S	*void* **DrawPicture**(*int* x, *int* y, *Picture* picture)	Dessine l'image **picture** à la position (**x,y**).
S	*void* **DrawPictureFlippedHorizontally**(*int* x, *int* y, *Picture* picture)	Retourne horizontalement l'image **picture** et la dessine à la position (**x,y**).
S	*void* **DrawPictureFlippedVertically**(*int* x, *int* y, *Picture* picture)	Retourne verticalement l'image **picture** et la dessine à la position (**x,y**).
S	*void* **DrawPictureRotated180Degrees**(*int* x, *int* y, *Picture* picture)	Tourne l'image **picture** d'un angle égale à **180°** dans le sens des aiguilles d'une montre et la dessine à la position (**x,y**).
S	*void* **DrawPictureRotated270Degrees**(*int* x, *int* y, *Picture* picture)	Tourne l'image **picture** d'un angle égale à **270°** dans le sens des aiguilles d'une montre et la dessine à la position (**x,y**).
S	*void* **DrawPictureRotated90Degrees**(*int* x, *int* y, *Picture* picture)	Tourne l'image **picture** d'un angle égale à **90°** dans le sens des aiguilles d'une montre et la dessine à la position (**x,y**).
S	*void* **ClearPartOfScreen**(*int* x, *int* y, *int* width, *int* height)	Efface un rectangle de largeur **width** et de hauteur **height** dont le coin supérieur gauche est situé à la position (**x,y**).
S	*void* **ClearPoint**(*int* x, *int* y)	Efface un point à la position (**x,y**).
S	*void* **ClearScreen**()	Efface l'écran.
S	*void* **InvertColors**(*bool* invert)	Inverse l'état des pixels sur l'écran. Les pixels allumés seront éteints et vice versa.

	Syntaxe	Description
S	*void* **RefreshScreen()**()	Ecrit l'intégralité du tampon d'affichage sur l'écran. Indispensable pour faire apparaître le contenu du buffer d'affichage.

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte BrainPad v1 ou v2 "Étape par Étape"

Exemple : Le programme suivant dessine un point, une ligne, un rectangle, un cercle et un cœur dans deux dimensions.

Display.cs

```
static void Main()
 {
    BrainPad.Display.DrawPoint(64, 32);
    BrainPad.Display.DrawLine(0, 52, 127, 52);
    BrainPad.Display.DrawRectangle(48, 20, 32, 24);
    BrainPad.Display.DrawCircle(64, 32, 20);

    byte[] pictureData = new byte[]
            {  0, 0, 1, 0, 0 ,0, 1, 0, 0,
               0, 1, 0, 1, 0, 1, 0, 1, 0,
               1, 0, 0, 0, 1, 0, 0, 0, 1,
               1, 0, 0, 0, 1, 0, 0, 0, 1,
               1, 0, 0, 0, 0, 0, 0, 0, 1,
               0, 1, 0, 0, 0, 0, 0, 1, 0,
               0, 0, 1, 0, 0, 0, 1, 0, 0,
               0, 0, 0, 1, 0, 1, 0, 0, 0,
               0, 0, 0, 0, 1, 0, 0, 0, 0,
            };

    BrainPad.Display.DrawPicture(0, 0,
BrainPad.Display.CreatePicture(9, 9, pictureData));
    BrainPad.Display.DrawPicture(10, 10,
BrainPad.Display.CreateScaledPicture(9, 9, pictureData, 2));
    BrainPad.Display.RefreshScreen();
 }
```

Le projet **TCLR_BRAINPAD_Display1_vx.x.x** est **téléchargeable** ici

---

Light Bulb

Le Light Bulb contient trois diodes électroluminescentes (LED) dans un seul boîtier. Une LED est rouge, l'autre est verte et l'autre est bleue. L'intensité lumineuse produite par chaque LED peut être réglée indépendamment pour créer jusqu'à un million de couleurs.

Méthodes

	Syntaxe	Description
S	*void* **TurnColor**(*double* red, *double* green, *double* blue)	Permet de créer une variété de couleurs en contrôlant la luminosité des LED rouge, verte et bleue. L'intensité de la couleur **minimum** est 0 et la **maximum** est **100**.
S	*void* **TurnRed**()	Le Light Bulb s'éclaire en rouge.
S	*void* **TurnBlue**()	Le Light Bulb s'éclaire en bleu.
S	*void* **TurnGreen**()	Le Light Bulb s'éclaire en vert.
S	*void* **TurnWhite**()	Le Light Bulb s'éclaire en blanc.
S	*void* **TurnOff**()	Le Light Bulb s'éteind. Correspond à red = 0.0, green = 0.0, blue = 0.0

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte BrainPad v1 ou v2 "Étape par Étape"

Exemple

LightBulb.cs

```
static void Main()
{
  while (true)
   {
     BrainPad.LightBulb.TurnRed();
     BrainPad.Wait.Seconds(1);
     BrainPad.LightBulb.TurnBlue();
     BrainPad.Wait.Seconds(1);
     BrainPad.LightBulb.TurnGreen();
     BrainPad.Wait.Seconds(1);
     BrainPad.LightBulb.TurnColor(5, 25, 100);
     BrainPad.Wait.Seconds(1);
    }
}
```

Le projet **TCLR_BRAINPAD_LightBulb_vx.x.x** est **téléchargeable** ici

---

Light Sensor

Méthodes

	Syntaxe	Description
S	*double* **ReadLightLevel**()	Lit la luminosité ambiante. Renvoie une valeur comprise entre 0 à 100.

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte

BrainPad v1 ou v2 "Étape par Étape"

Exemple : Le Light Bulb s'éclaire si la luminosité est < 50

LightSensor.cs

```
static void Main()
{
    double level = 0;

    while (true)
    {
       level = BrainPad.LightSensor.ReadLightLevel();

       if (level < 50)
       {
         BrainPad.LightBulb.TurnWhite();
       }
       else
       {
         BrainPad.LightBulb.TurnOff();
       }
    }
}
```

Le projet **TCLR_BRAINPAD_LightSensor_vx.x.x** est **téléchargeable** ici

---

Temperature Sensor

Méthodes

	Syntaxe	Description
S	*double* **ReadTemperatureInCelsius**()	Lit la température et renvoie sa valeur en degrés Celcius.
S	*double* **ReadTemperatureInFahrenheit**()	Lit la température et renvoie sa valeur en degrés Fahrenheit.

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte BrainPad v1 ou v2 "Étape par Étape"

Exemple : Affiche la température ambiante en degrés Celcius sur l'écran du BrainPad

TemperatureSensor.cs

```
static void Main()
{
```

```
  while (true)
   {
     BrainPad.Display.Clear();
     BrainPad.Display.DrawSmallText(20, 12, "Temperature en dC");
     BrainPad.Display.DrawNumber(35, 24,
BrainPad.TemperatureSensor.ReadTemperatureInCelsius());
     BrainPad.Display.RefreshScreen();
     BrainPad.Wait.Minimum();
   }
}
```

Le projet **TCLR_BRAINPAD_TempSensor_vx.x.x** est **téléchargeable** ici

---

ServoMotors

La carte BrainPad prend en charge deux servomoteurs. Ceux-ci peuvent être des servomoteurs à rotation continue ou asservis en position.

Méthodes

	Syntaxe	Description
S	*void* **ServoOne.ConfigureAsContinuous**(*bool* inverted)	Défini le servomoteur 1 comme étant à rotation continue. *inverted*=vraie fera tourner le servo 1 dans la direction opposée.
S	*void* **ServoTwo.ConfigureAsContinuous**(*bool* inverted)	Défini le servomoteur 2 comme étant à rotation continue. *inverted*=vraie fera tourner le servo 2 dans la direction opposée.
S	*void* **ServoOne.ConfigureAsPositional**(*bool* inverted)	Défini le servomoteur 1 comme étant asservi en position. *inverted*=vraie inverse les repères mini et maxi du servo 1. 0° devient 180 et vice versa.
S	*void* **ServoTwo.ConfigureAsPositional**(*bool* inverted)	Défini le servomoteur 2 comme étant asservi en position. *inverted*=vraie inverse les repères mini et maxi du servo 2. 0° devient 180 et vice versa.
S	*void* **ServoOne.ConfigurePulseParameters**(*double* minimumPulseWidth, *double* maximumPulseWidth)	Règle les largeurs d'impulsion du servo 1 en ms.
S	*void* **ServoTwo.ConfigurePulseParameters**(*double* minimumPulseWidth, *double* maximumPulseWidth)	Règle les largeurs d'impulsion du servo 2 en ms.

	Syntaxe	Description
S	*void* **ServoOne.Set**(*double* value)	Pour un servomoteur 1 asservi en position, value doit être compris entre 0 et 180 degrés. Pour un servomoteur 1 à rotation continu, value doit être compris entre -100% et 100%.
S	*void* **ServoTwo.Set**(*double* value)	Pour un servomoteur 2 asservi en position, value doit être compris entre 0 et 180 degrés. Pour un servomoteur 2 à rotation continu, value doit être compris entre -100% et 100%.
S	*void* **ServoOne.Stop**()	Désactive le signal de commande du servomoteur.
S	*void* **ServoTwo.Stop**()	Désactive le signal de commande du servomoteur.

Pour tester les exemples, suivre la démarche sur la page Premiers programmes en C# avec une carte BrainPad v1 ou v2 "Étape par Étape"

Exemple : Commande d'un servomoteur asservi en position

ServoPos.cs

```
static void Main()
{
    BrainPad.ServoMotors.ServoOne.ConfigureAsPositional(false);
    BrainPad.ServoMotors.ServoOne.ConfigurePulseParameters(1.0, 2.0);

    while (true)
    {
      BrainPad.ServoMotors.ServoOne.Set(0.0);
      BrainPad.Wait.Seconds(2.0);
      BrainPad.ServoMotors.ServoOne.Set(180.0);
      BrainPad.Wait.Seconds(2.0);
    }
}
```

Le projet **TCLR_BRAINPAD_ServoPos_vx.x.x** est **téléchargeable** ici

Pour aller plus loin

Le connecteur **mikroBUS** permet d'étendre les fonctionnalités de la carte BrainPad (GPIO, I2C, SPI, UART).
Voir la page Exemples codés en C# pour la carte BrainPad BP2 (STM32F401).

From:
https://webge.fr/dokuwiki/ - **WEBGE Wikis**

Permanent link:
https://webge.fr/dokuwiki/doku.php?id=brainpad:classesbp2

Last update: **2021/08/11 09:19**