Table des matières

Le principe ?!

En utilisant les variables à remplir (**appendValueInput**), on va alimenter du code qui n'est finalement que du texte. Mais parfois le texte vient d'une variable, parfois c'est le texte qui sera directement affiché.

Mais il faut connaître le code Arduino à utiliser pour ensuite l'injecter dans la moulinette de Blockly.

On prépare le code :

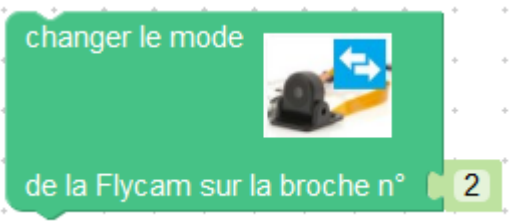
On peut s'aider des blocs existants pour générer le code à faire apparaître :

```
#include <Servo.h>

Servo servo_2;

void setup() {
   servo_2.attach(2);
}

void loop() {
   servo_2.write(180);
   delay(3000);
   servo_2.write(0);
   delay(3000);
}
```

Donc il faudra *faire* écrire :

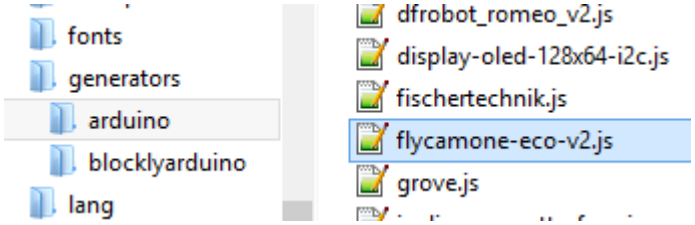
- un appel à une bibliothèque : #include <Servo.h>
- la définition du nom d'un servomoteur : Servo servo_2;
- dans le setup on explicite que le servo_2 est connecté sur la broche 2 : servo_2.attach(2);
- dans la boucle 'loop' on décrit le fonctionnement voulu

Le 2 est le numéro de la broche où sera connectée la Flycam, mais elle doit être alimentée par un bloc extérieur puis on utilise la variable ad-hoc : variable 'PIN' définie lors de **this.appendValueInput ("PIN")**



On prépare le fichier 'générateur' :

On créé un dossier de texte brut dans le dossier '**\generators\arduino**' **avec le même nom que celui qui a définit le dessin du bloc !**



On s'inspire de blocs existants et on attaque !

On complète le fichier 'générateur' :

```
/**
 * Block pour la FlyCamOne Eco v2
 * @author Seb Canet (canet.s@free.fr)
 */
'use strict';

goog.provide('Blockly.Arduino.flycam');

goog.require('Blockly.Arduino');

Blockly.Arduino.flycam_switch = function() {
  var value_pin = Blockly.Arduino.valueToCode(this, 'PIN',
Blockly.Arduino.ORDER_ATOMIC);

  Blockly.Arduino.inludes_['define_servo'] = '#include <Servo.h>\n';
  Blockly.Arduino.definitions_['var_servo' + value_pin] = 'Servo servo_' +
value_pin + ';\n';
  Blockly.Arduino.setups_['setup_servo_' + value_pin] = 'servo_' + value_pin
+ '.attach(' + value_pin + ');\n';
  var code = 'servo_' + value_pin + '.write(180);\n'
  +  'delay(3000);\n'
  +  'servo_' + value_pin + '.write(0);\n'
  +  'delay(1000);\n';
  return code;
};
```

On comprend le principe

- on définit encore notre catégorie et dépendances mais cette fois on dit qu'il s'agit de code

Arduino : goog.provide('Blockly.**Arduino**.flycam');
- on stocke dans une variable la valeur récupérée dans la programmation graphique par blocs : **var value_pin** = Blockly.Arduino.valueToCode(this, '**PIN**', Blockly.Arduino.ORDER_ATOMIC);
- pour chacune des parties du programe Arduino (include - definition - setup - loop), on écrit le texte qui doit aparaître :
    - tout en haut on commence par l'appel à la bibliothèque servo : Blockly.Arduino.inludes_['define_servo'] = '#include <Servo.h>\n';
    - puis on définit une variable de type Servo : Blockly.Arduino.definitions_['var_servo' value_pin] = 'Servo servo_' value_pin ';\n';
    - et dans l'initialisation (ou setup) on *attache* le servomoteur sur la broche n°....contenu dans la **variable value_pin**
    - ensuite le code sera automatiquement rajouté dans la boucle infinie 'loop'.

Vous remarquerez qu'il faut tout écrire, même les retours à la ligne grâce à '\n'.

On va plus loin

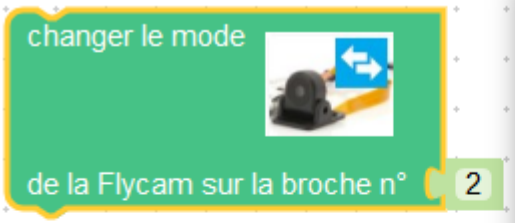
Mais, si vous allez comprendre !

Au lieu d'enchaîner les actions peu explicites servo2.write puis wait puis ...., on va créer une fonction en langage Arduino qui sera appelée dans la boucle infinie 'loop'.

```
Blockly.Arduino.flycam_switch = function() {
  var value_pin = Blockly.Arduino.valueToCode(this, 'PIN',
Blockly.Arduino.ORDER_ATOMIC);

  Blockly.Arduino.includes_['define_servo'] = '#include <Servo.h>';
  Blockly.Arduino.definitions_['var_servo' + value_pin] = 'Servo servo_' +
value_pin + ';\n';
  Blockly.Arduino.definitions_['flycam_switch_function'] = 'void
flycam_switch(Servo SERVO) {\n'
  + '  SERVO.write(180);\n'
  + '  delay(3000);\n'
  + '  SERVO.write(0);\n'
  + '  delay(1000);\n'
  + '}';
  var code = 'flycam_switch(servo_' + value_pin + ');\n';
  return code;
};
```

On passe d'une boucle peu explicite à un système de fonctions plus simples à lire, debugger ou répéter :


```
#include <Servo.h>

Servo servo_2;


void setup() {
  servo_2.attach(2);

}

void loop() {
  servo_2.write(180);
  delay(500);
  servo_2.write(0);
  delay(2000);

}
```




```
#include <Servo.h>

Servo servo_2;

void flycam_switch(Servo SERVO) {
  SERVO.write(180);
  delay(3000);
  SERVO.write(0);
  delay(1000);
}

void setup() {
}

void loop() {
  flycam_switch(servo_2);

}
```

Ca ne fonctionne toujours pas ??!!

C’est normal, il reste un chapitre...