Table des matières

Mais rien n'apparaît ??! C'est normal...

En effet, n'ont été créés que les 2 fichiers utiles au fonctionnement de Blockly : le créateur du bloc et le générateur de code. Mais Blockly@rduino ne sait toujours pas que ces fichiers existent et qu'il faut les utiliser, voici dnc comment terminer la procédure.

On les appelle

Il faut éditer les **2** fichiers '**\blocks\arduino_resume.js**' (qui contient la liste de toutes les définitions de blocs dans lesquelles piocher pour les construire) ***ET*** '**\generators\arduino_resume.js**' (qui contient la liste de toutes les traductions des blocs en langage Arduino dans lesquelles piocher pour les traduire).

Respectons un peu d'ordre, il suffit de copier une ligne existante et de **changer le dossier ET le nom du fichier** :

- tout d'abord les définitions du bloc '**\blocks\arduino_resume.js**', autant les mettre par ordre alphabétique du **nom de dossier***(qui est identique au nom du fichier javascript)* :

```
"blocks/display-oled-128x64-i2c/display-oled-128x64-i2c.js"));
"blocks/fischertechnik/fischertechnik.js"));
"blocks/flycamone-eco-v2/flycamone-eco-v2.js"));
"blocks/grove/grove.js"));
"blocks/jeulin_maquette_feux/jeulin_maquette_feux.js"));
```

- ensuite faire de même dans la liste '**\generators\arduino_resume.js**' des **générateurs** de code :

```
"generators/arduino/fischertechnik.js"));
"generators/arduino/flycamone-eco-v2.js"));
"generators/arduino/grove.js"));
"generators/arduino/jeulin_maquette_feux.js"));
```

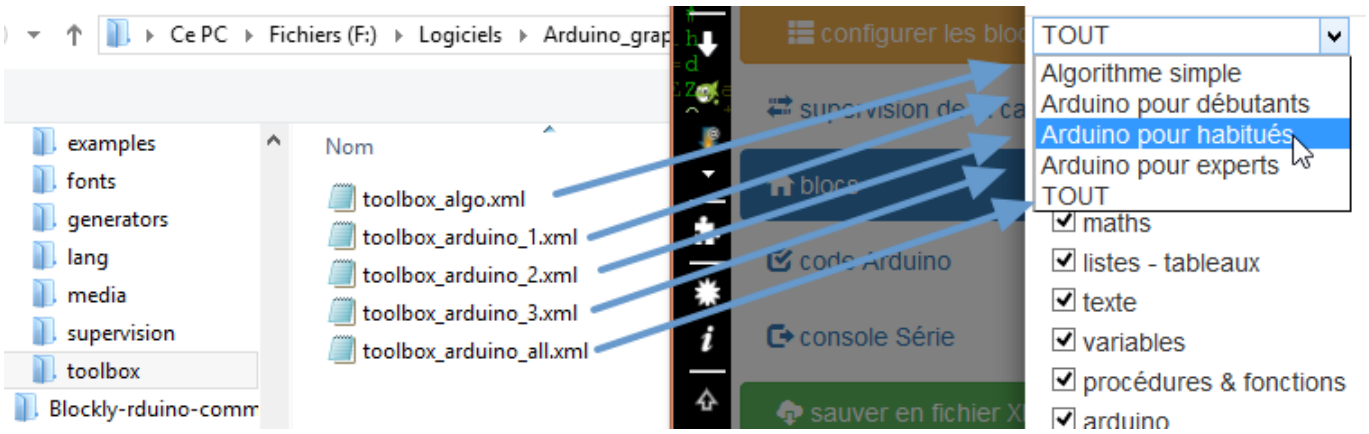
Il ne reste plus qu'à rafraîchir la page de Blockly@rduino et admirer le travail !!!

Bon, ben là, il ne se passe toujours rien...si tout est normal c'est déjà que vous n'avez pas rajouté d'erreur...

Il en manque...

Il sait qu'il doit examiner ces contenus, mais il les affiche quand ? A quel endroit dans le menu ? Ah mais oui, nom de Zeus la toolbox !!!

La '**toolbox**' est le nom donné par Blockly pour les menus, les différents fichiers sont enregistrés dans le **dossier '\toolbox\'** :



Libre à vous de créer d'autres toolbox en n'oubliant pas de modifier index.html :

```
511         <div id="divToolbox">
512             <label id="labelToolboxDefinition"></label>
513             <select id="toolboxes">
514                 <option value="./toolbox/toolbox_algo">Algorithme simple</option>
515                 <option value="./toolbox/toolbox_arduino_1">Arduino pour débutants</option>
516                 <option value="./toolbox/toolbox_arduino_2">Arduino pour habitués</option>
517                 <option value="./toolbox/toolbox_arduino_3">Arduino pour experts</option>
518                 <option value="./toolbox/toolbox_arduino_all" selected="selected">TOUT</option>
519             </select>
520         </div>
```

Bref ouvrons donc la ou les *toolbox* à modifier et rajoutons les lignes correspondantes en s'inspirant de ce qui existe déjà :

```
773     </category>
774     <category name="CAT_FLYCAMONE" colour="#46C286">
775         <block type="flycam_switch">
776             <value name="PIN">
777                 <shadow type="math_number">
778                     <field name="NUM">2</field>
779                 </shadow>
780             </value>
781         </block>
782         <block type="flycam_record">
783             <value name="PIN">
784                 <shadow type="math_number">
785                     <field name="NUM">2</field>
786                 </shadow>
787             </value>
788         </block>
789         <block type="flycam_stop">
790             <value name="PIN">
791                 <shadow type="math_number">
792                     <field name="NUM">2</field>
793                 </shadow>
794             </value>
795         </block>
796     </category>
797     <category name="CAT_BQ" colour="#608621">
798         <category name="CAT_BQ_IN" colour="#608621">
```

C'est facile à décrypter :

- je ne vous explique pas la différence entre **category** et **block**...
- je mets en nom de catégorie un **variable CAT_FLYCAMONE**
- je rajoute un appel à chacun des 3 blocs que j'ai créés **en utilisant le même nom que le nom de la fonction** ! block type="**flycam_switch**" <→ Blockly.Blocks.**flycam_switch**
- **colour =** la même valeur que celle qui définit graphiquement les blocs
- l'entrée '*shadow*' permet de pré-remplir l'entrée dont le nom de **variable** est **PIN** avec un bloc '*fantôme*', à tester pour comprendre vite

Dans mon exemple j'ai choisi de le disposer avant la catégorie 'CAT_BQ'. Ce n'est pas le nom qui apparaît...

Courage on y est presque !

Comme nous n'avons utilisé que des **variables**, les textes portent pour l'instant le nom de la **variable** et non pas son contenu ! Là c'est très facile car il suffit de porter les *équivalences* dans les fichiers de langue ad-hoc dans le dossier '**\lang\blocks**'(*merci de compléter au moins le english en plus*) :

- dans **fr.js** on va retrouver :
  - le nom de la catégorie avec les autres (*merci de les laisser par ordre alphabétique*) :

```
Blockly.Msg.CAT_FISCHERTECHNIK = "fischertechnik";  //added march 26th 2016
Blockly.Msg.CAT_FISCHERTECHNIK_IN = "capteurs";
Blockly.Msg.CAT_FISCHERTECHNIK_OUT = "actionneurs";
Blockly.Msg.CAT_FISCHERTECHNIK_MOTORS_CC = "moteurs CC";

Blockly.Msg.CAT_FLYCAMONE = "FlyCamOne Eco v2"; //added august 20th 2016
```

  - la définition de chaque variable pour lesquels on veut voir du texte apparaître dans le bloc :

```
//Added May 1rst 2016
Blockly.Msg.ROMEO_HELPURL = "http://www.dfrobot.com/wiki/index.php/Romeo_

//Added august 20th 2016
Blockly.Msg.FLYCAM_SWITCH_HELPURL = "http://tic.technologiescollege.fr/wi
Blockly.Msg.FLYCAM_SWITCH_TEXT = "changer le mode";
Blockly.Msg.FLYCAM_SWITCH_INPUT = "de la flycam sur la broche n°";
Blockly.Msg.FLYCAM_SWITCH_TOOLTIP = "patienter car la commande doit se fi
Blockly.Msg.FLYCAM_RECORD_HELPURL = Blockly.Msg.FLYCAM_SWITCH_HELPURL;
Blockly.Msg.FLYCAM_RECORD_TEXT = "lancer la capture";
Blockly.Msg.FLYCAM_RECORD_INPUT = Blockly.Msg.FLYCAM_SWITCH_INPUT;
Blockly.Msg.FLYCAM_RECORD_TOOLTIP = "envoi d'une impulsion d'1s de type s
Blockly.Msg.FLYCAM_STOP_HELPURL = Blockly.Msg.FLYCAM_SWITCH_HELPURL;
Blockly.Msg.FLYCAM_STOP_TEXT = "arrêter la capture";
Blockly.Msg.FLYCAM_STOP_INPUT = Blockly.Msg.FLYCAM_SWITCH_INPUT;
Blockly.Msg.FLYCAM_STOP_TOOLTIP = "envoi d'une impulsion d'1s de type
```

```
Blockly.Blocks.flycam_stop = {
  init: function() {
    this.setColour(Blockly.Blocks.flycam.HUE);
    this.setHelpUrl(Blockly.Msg.FLYCAM_STOP_HELPURL);
    this.appendDummyInput("")
        .appendField(Blockly.Msg.FLYCAM_STOP_TEXT)
        .appendField(new Blockly.FieldImage(Blockly.pat
          'blocks/flycamone-eco-v2/flycam_stop.jpg', Bloc
    this.appendValueInput("PIN", 'Number')
        .setCheck('Number')
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(Blockly.Msg.FLYCAM_STOP_INPUT);
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setTooltip(Blockly.Msg.FLYCAM_STOP_TOOLTIP);
  }
};
```

- dans le fichier **en.js**, à peu près aux mêmes endroits vous rajoutez la version anglaise :

```
//Added august 20th 2016
Blockly.Msg.FLYCAM_SWITCH_HELPURL = "http://tic.technologiescollege.fr/wi
Blockly.Msg.FLYCAM_SWITCH_TEXT = "change mode";
Blockly.Msg.FLYCAM_SWITCH_INPUT = "of Flycam on PIN#";
Blockly.Msg.FLYCAM_SWITCH_TOOLTIP = "be patient because it sends a signal
Blockly.Msg.FLYCAM_RECORD_HELPURL = Blockly.Msg.FLYCAM_SWITCH_HELPURL;
Blockly.Msg.FLYCAM_RECORD_TEXT = "start capture";
Blockly.Msg.FLYCAM_RECORD_INPUT = Blockly.Msg.FLYCAM_SWITCH_INPUT;
Blockly.Msg.FLYCAM_RECORD_TOOLTIP = "send order for 1s, like a servo=180°
Blockly.Msg.FLYCAM_STOP_HELPURL = Blockly.Msg.FLYCAM_SWITCH_HELPURL;
Blockly.Msg.FLYCAM_STOP_TEXT = "stop capture";
Blockly.Msg.FLYCAM_STOP_INPUT = Blockly.Msg.FLYCAM_SWITCH_INPUT;
Blockly.Msg.FLYCAM_STOP_TOOLTIP = "send order for 1s, like a servo=0°";
```

TRUC & ASTUCE: comme il s'agit de variables, par aspect pratique, j'ai fait des équivalences de variables : Blockly.Msg.FLYCAM_STOP_INPUT = Blockly.Msg.FLYCAM_SWITCH_INPUT; car la variable Blockly.Msg.FLYCAM_SWITCH_INPUT est définie juste au-dessus. Car **le système lit les informations dans l'ordre des lignes !**

Ca y est !!! A vous la gloire d'avoir contribué à ce magnifique projet !!!

Et la joie de debugger si comme moi vous faites régulièrement des fautes de frappe, d'oubli de **points virgule ;** , etc...

Ok, mais ce n'est plus suffisant !

En effet, en programmation héritée des langages C, il est indispensable de définir la *nature des objets*, donc le **type** des **variables**. Grâce à l'excellent script du créateur d'Ardublockly (*et pas Ardublock*), tous les types sont recensés dans le fichier '**\core_Ardublockly\static_typing.js**' :

```
/** Single character. */
Blockly.Types.CHARACTER = new Blockly.Type({
  typeId: 'Character',
  typeMsgName: 'ARD_TYPE_CHAR',
  compatibleTypes: []
});

/** Text string. */
Blockly.Types.TEXT = new Blockly.Type({
  typeId: 'Text',
  typeMsgName: 'ARD_TYPE_TEXT',
  compatibleTypes: [Blockly.Types.CHARACTER]
});

/** Boolean. */
Blockly.Types.BOOLEAN = new Blockly.Type({
  typeId: 'Boolean',
  typeMsgName: 'ARD_TYPE_BOOL',
  compatibleTypes: []
});
```

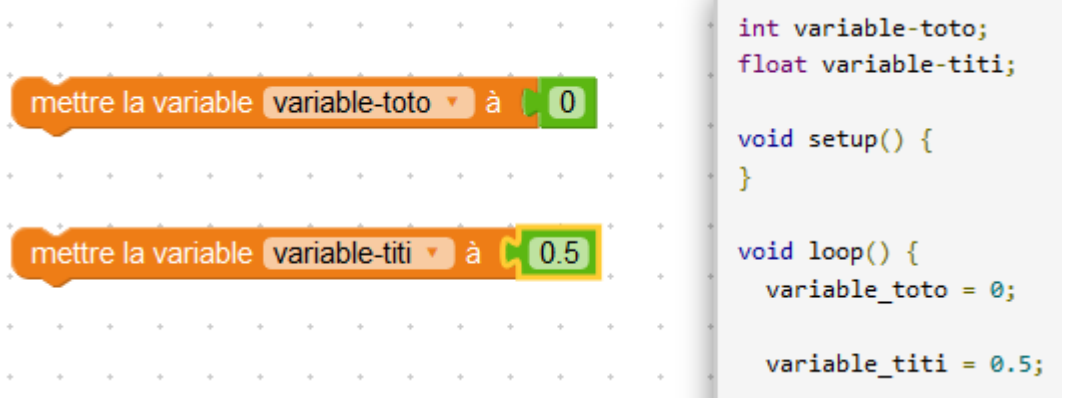
Donc il va aussi falloir expliquer à Blockly@rduino de quelle **nature/type** (*en référence aux noms dans le fichier ci-dessus*) est **chaque bloc** créé :

```
Blockly.Types.CHARACTER             // Single character
Blockly.Types.TEXT                   // General text string type
Blockly.Types.BOOLEAN                // Can only have two values, generally 0
for false, or 1 for true
Blockly.Types.NUMBER                 // A general number type
Blockly.Types.VOLATIL_NUMBER         // Volatil specific for interruption
Blockly.Types.SHORT_NUMBER           // Short integer number
Blockly.Types.LARGE_NUMBER           // Number in a large range
Blockly.Types.DECIMAL                // Number type for numbers with a
fractional part
Blockly.Types.ARRAY                  // Array of any type of items
Blockly.Types.NULL                   // Used as a "no type" wild card
natively
Blockly.Types.UNDEF                  // Can be used to delegate type
assignment
Blockly.Types.CHILD_BLOCK_MISSING    // Set when no child block (meant to
define the variable type) is connected
```

Créez pour mon exemple le fichier **'blocksflycamone-eco-v2blocks_typing.js'** et rajoutez les lignes ad-hoc en bas :

```
221
222  //*************************************************************************
223  //                              FlyCamEco v2
224  //*************************************************************************
225
226  //--------------------------------flycamone-eco-v2.js-----------------------
227
228  Blockly.Blocks.flycam_switch.getBlockType = function() {
229      return Blockly.Types.NUMBER;
230  };
231  Blockly.Blocks.flycam_record.getBlockType = function() {
232      return Blockly.Types.NUMBER;
233  };
234  Blockly.Blocks.flycam_stop.getBlockType = function() {
235      return Blockly.Types.NUMBER;
236  };
```

Cela permet de typer **automatiquement** les variables en fonction des blocs en entrée :



Puis finir en recensant ce nouveau fichier de typages dans le fichier centralisateur **'blocksblocks_typing.js'**

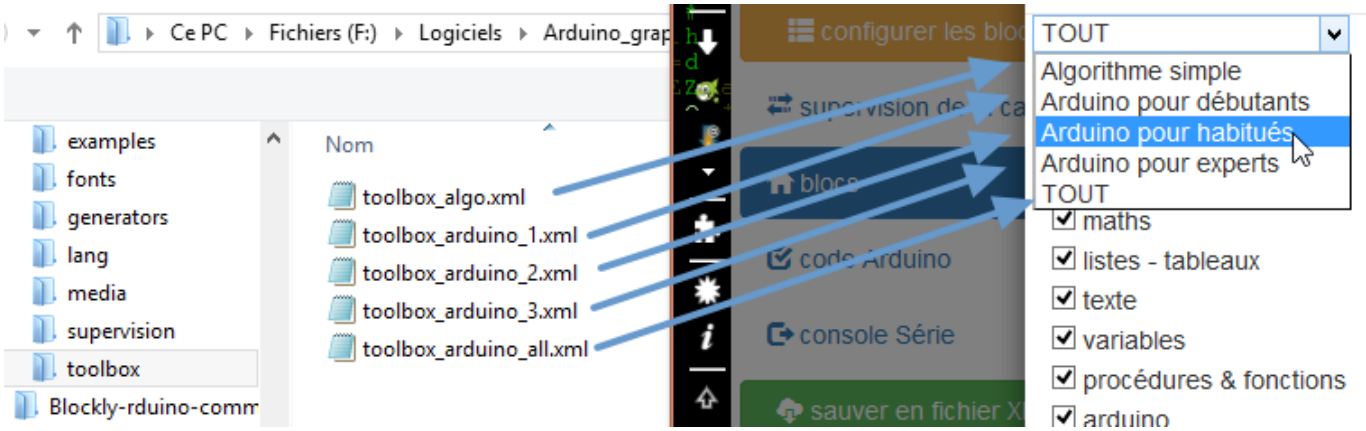
Ouf c'est fini !

Il ne vous reste plus qu'à envoyer un petit mail à canet.s@free.fr pour un petit merci, un petit coucou et surtout **contribuer en proposant de rajouter à cette aventure collective vos travaux !**

Il en manque...

Il sait qu'il doit examiner ces contenus, mais il les affiche quand ? A quel endroit dans le menu ? Ah mais oui, nom de Zeus la toolbox !!!

La '**toolbox**' est le nom donné par Blockly pour les menus, les différents fichiers sont enregistrés dans le **dossier '\toolbox\'** :



Libre à vous de créer d'autres toolbox en n'oubliant pas de modifier index.html :

```
511     <div id="divToolbox">
512         <label id="labelToolboxDefinition"></label>
513         <select id="toolboxes">
514             <option value="./toolbox/toolbox_algo">Algorithme simple</option>
515             <option value="./toolbox/toolbox_arduino_1">Arduino pour débutants</option>
516             <option value="./toolbox/toolbox_arduino_2">Arduino pour habitués</option>
517             <option value="./toolbox/toolbox_arduino_3">Arduino pour experts</option>
518             <option value="./toolbox/toolbox_arduino_all" selected="selected">TOUT</option>
519         </select>
520     </div>
```

Bref ouvrons donc la ou les *toolbox* à modifier et rajoutons les lignes correspondantes en s'inspirant de ce qui existe déjà :

```
773     </category>
774     <category name="CAT_FLYCAMONE" colour="#46C286">
775         <block type="flycam_switch">
776             <value name="PIN">
777                 <shadow type="math_number">
778                     <field name="NUM">2</field>
779                 </shadow>
780             </value>
781         </block>
782         <block type="flycam_record">
783             <value name="PIN">
784                 <shadow type="math_number">
785                     <field name="NUM">2</field>
786                 </shadow>
787             </value>
788         </block>
789         <block type="flycam_stop">
790             <value name="PIN">
791                 <shadow type="math_number">
792                     <field name="NUM">2</field>
793                 </shadow>
794             </value>
795         </block>
796     </category>
797     <category name="CAT_BQ" colour="#608621">
798         <category name="CAT_BQ_IN" colour="#608621">
```

C'est facile à décrypter :

- je ne vous explique pas la différence entre **category** et **block**...
- je mets en nom de catégorie un **variable CAT_FLYCAMONE**
- je rajoute un appel à chacun des 3 blocs que j'ai créés **en utilisant le même nom que le nom de la fonction** ! block type="**flycam_switch**" <→ Blockly.Blocks.**flycam_switch**
- **colour =** la même valeur que celle qui définit graphiquement les blocs
- l'entrée '*shadow*' permet de pré-remplir l'entrée dont le nom de **variable** est **PIN** avec un bloc '*fantôme*', à tester pour comprendre vite

Dans mon exemple j'ai choisi de le disposer avant la catégorie 'CAT_BQ'. Ce n'est pas le nom qui apparaît...

Courage on y est presque !

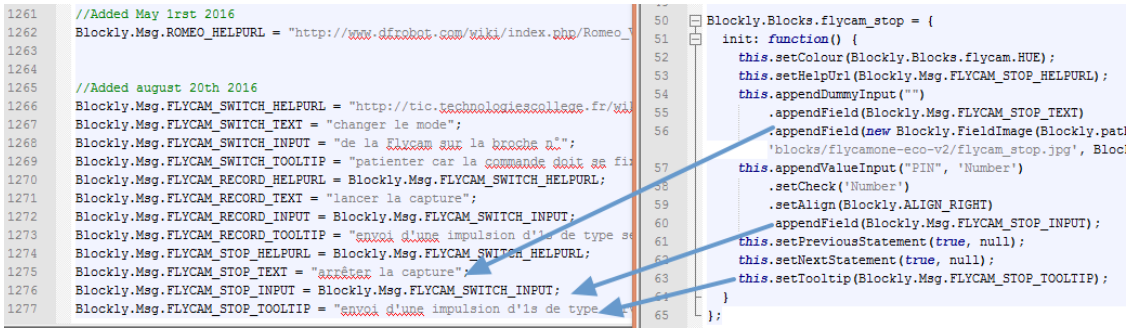
Comme nous n'avons utilisé que des **variables**, les textes portent pour l'instant le nom de la **variable** et non pas son contenu ! Là c'est très facile car il suffit de porter les *équivalences* dans les fichiers de langue ad-hoc dans le dossier '**\lang\blocks**'(*merci de compléter au moins le english en plus*) :

- dans **fr.js** on va retrouver :
  - le nom de la catégorie avec les autres (*merci de les laisser par ordre alphabétique*) :

```
Blockly.Msg.CAT_FISCHERTECHNIK = "fischertechnik";  //added march 26th 2016
Blockly.Msg.CAT_FISCHERTECHNIK_IN = "capteurs";
Blockly.Msg.CAT_FISCHERTECHNIK_OUT = "actionneurs";
Blockly.Msg.CAT_FISCHERTECHNIK_MOTORS_CC = "moteurs CC";

Blockly.Msg.CAT_FLYCAMONE = "FlyCamOne Eco v2"; //added august 20th 2016
```

- la définition de chaque variable pour lesquels on veut voir du texte apparaître dans le bloc :



- dans le fichier **en.js**, à peu près aux mêmes endroits vous rajoutez la version anglaise :

```
//Added august 20th 2016
Blockly.Msg.FLYCAM_SWITCH_HELPURL = "http://tic.technologiescollege.fr/wi
Blockly.Msg.FLYCAM_SWITCH_TEXT = "change mode";
Blockly.Msg.FLYCAM_SWITCH_INPUT = "of Flycam on PIN#";
Blockly.Msg.FLYCAM_SWITCH_TOOLTIP = "be patient because it sends a signal
Blockly.Msg.FLYCAM_RECORD_HELPURL = Blockly.Msg.FLYCAM_SWITCH_HELPURL;
Blockly.Msg.FLYCAM_RECORD_TEXT = "start capture";
Blockly.Msg.FLYCAM_RECORD_INPUT = Blockly.Msg.FLYCAM_SWITCH_INPUT;
Blockly.Msg.FLYCAM_RECORD_TOOLTIP = "send order for 1s, like a servo=180°
Blockly.Msg.FLYCAM_STOP_HELPURL = Blockly.Msg.FLYCAM_SWITCH_HELPURL;
Blockly.Msg.FLYCAM_STOP_TEXT = "stop capture";
Blockly.Msg.FLYCAM_STOP_INPUT = Blockly.Msg.FLYCAM_SWITCH_INPUT;
Blockly.Msg.FLYCAM_STOP_TOOLTIP = "send order for 1s, like a servo=0°";
```

TRUC & ASTUCE: comme il s'agit de variables, par aspect pratique, j'ai fait des équivalences de variables : Blockly.Msg.FLYCAM_STOP_INPUT = Blockly.Msg.FLYCAM_SWITCH_INPUT; car la variable Blockly.Msg.FLYCAM_SWITCH_INPUT est définie juste au-dessus. Car **le système lit les informations dans l'ordre des lignes !**

Ca y est !!! A vous la gloire d'avoir contribué à ce magnifique projet !!!

Et la joie de debugger si comme moi vous faites régulièrement des fautes de frappe, d'oubli de **points virgule ;** , etc...

Ok, mais ce n'est plus suffisant !

En effet, en programmation héritée des langages C, il est indispensable de définir la *nature des objets*, donc le **type** des **variables**. Grâce à l'excellent script du créateur d'Ardublockly (*et pas Ardublock*), tous les types sont recensés dans le fichier '**\core_Ardublockly\static_typing.js**' :

```
/** Single character. */
Blockly.Types.CHARACTER = new Blockly.Type({
  typeId: 'Character',
  typeMsgName: 'ARD_TYPE_CHAR',
  compatibleTypes: []
});

/** Text string. */
Blockly.Types.TEXT = new Blockly.Type({
  typeId: 'Text',
  typeMsgName: 'ARD_TYPE_TEXT',
  compatibleTypes: [Blockly.Types.CHARACTER]
});

/** Boolean. */
Blockly.Types.BOOLEAN = new Blockly.Type({
  typeId: 'Boolean',
  typeMsgName: 'ARD_TYPE_BOOL',
  compatibleTypes: []
});
```

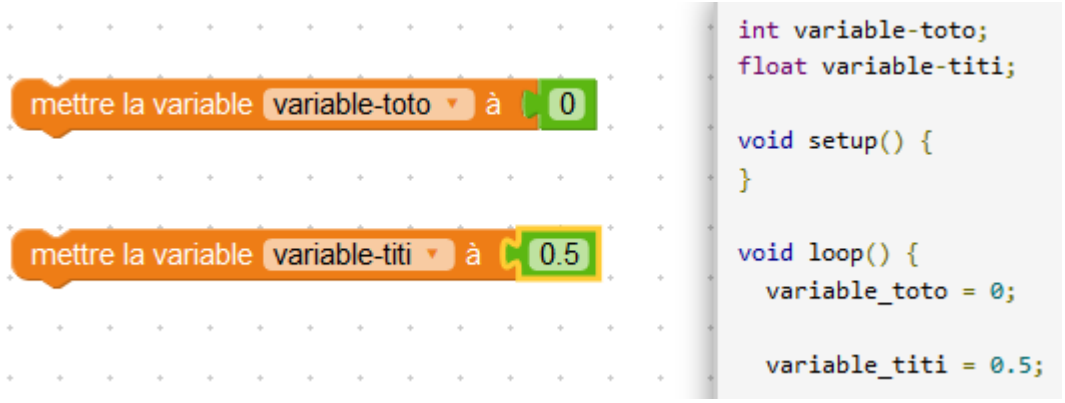
Donc il va aussi falloir expliquer à Blockly@rduino de quelle **nature/type** (*en référence aux noms dans le fichier ci-dessus*) est **chaque bloc** créé :

```
Blockly.Types.CHARACTER             // Single character
Blockly.Types.TEXT                   // General text string type
Blockly.Types.BOOLEAN                // Can only have two values, generally 0
for false, or 1 for true
Blockly.Types.NUMBER                 // A general number type
Blockly.Types.VOLATIL_NUMBER         // Volatil specific for interruption
Blockly.Types.SHORT_NUMBER           // Short integer number
Blockly.Types.LARGE_NUMBER           // Number in a large range
Blockly.Types.DECIMAL                // Number type for numbers with a
fractional part
Blockly.Types.ARRAY                  // Array of any type of items
Blockly.Types.NULL                   // Used as a "no type" wild card
natively
Blockly.Types.UNDEF                  // Can be used to delegate type
assignment
Blockly.Types.CHILD_BLOCK_MISSING    // Set when no child block (meant to
define the variable type) is connected
```

Créez pour mon exemple le fichier **'blocksflycamone-eco-v2blocks_typing.js'** et rajoutez les lignes ad-hoc en bas :

```
221
222  //*************************************************************************
223  //                                FlyCamEco v2
224  //*************************************************************************
225
226  //---------------------------------flycamone-eco-v2.js-----------------------
227
228  Blockly.Blocks.flycam_switch.getBlockType = function() {
229      return Blockly.Types.NUMBER;
230  };
231  Blockly.Blocks.flycam_record.getBlockType = function() {
232      return Blockly.Types.NUMBER;
233  };
234  Blockly.Blocks.flycam_stop.getBlockType = function() {
235      return Blockly.Types.NUMBER;
236  };
```

Cela permet de typer **automatiquement** les variables en fonction des blocs en entrée :



Puis finir en recensant ce nouveau fichier de typages dans le fichier centralisateur **'blocksblocks_typing.js'**

Ouf c'est fini !

Il ne vous reste plus qu'à envoyer un petit mail à canet.s@free.fr pour un petit merci, un petit coucou et surtout **contribuer en proposant de rajouter à cette aventure collective vos travaux !**