Mixly 第三方制作库(make 库)教程

一、下载和安装 make 库

1、make 库介绍和下载

make 库是一个制作 Mixly 中图形化模块的第三方库，其源码已经发布到 gitee 和 github 上，如果电脑上没有这个库可搜索 Libraries_For_Mixly 进行下载。

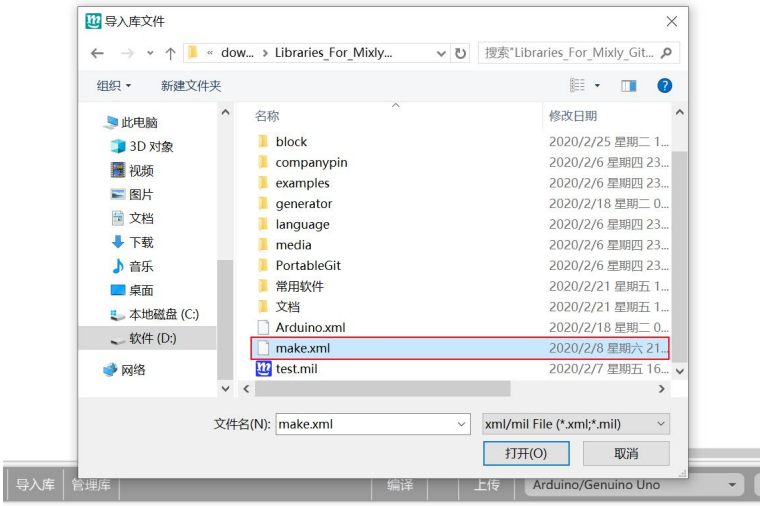
（1）Gitee：https://gitee.com/smilebrightly/Libraries_for_Mixly.git

（2）Github:https://github.com/3294713004wlb/Libraries_for_Mixly.git

2、make 库的导入

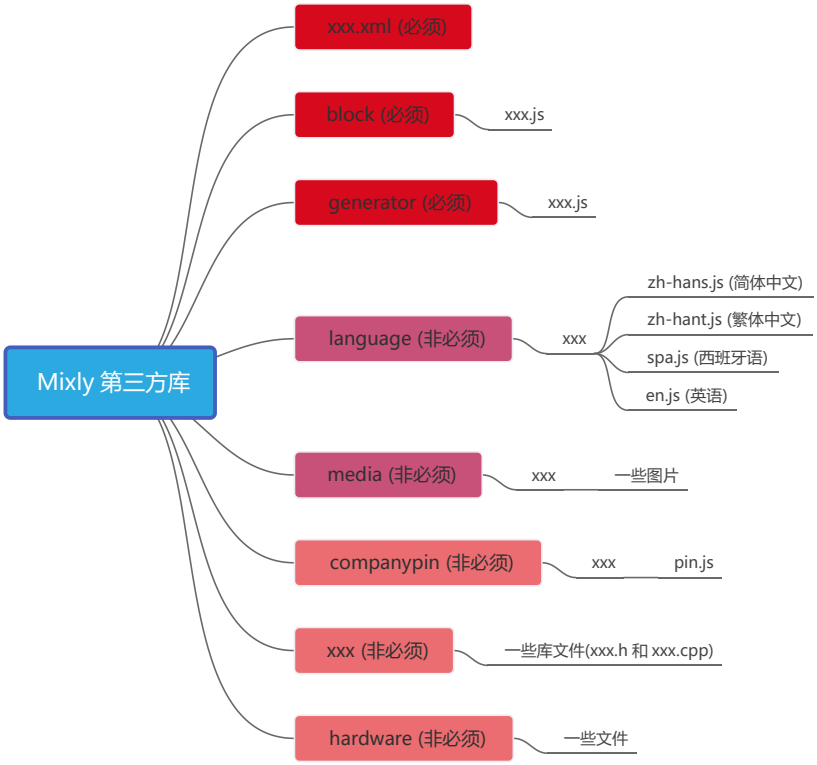
Make 库下载完成后，进入 Mixly，单击导入库。

选中 make.xml，点击打开，就可以完成 make 库的导入。



导入完成后，把滑动条拖到最下方，就可以看到刚刚导入的 make 库，这个库的结构和下图相似。

二、Mixly 第三方库结构介绍



一个完整的 Mixly 第三方库结构如上图所示。其中 xxx.xml、block/xxx.js、generator/xxx.js 这三个是库文件中必不可少的，若缺少其中某一个，导入库后可能会出现错误。

关于第三方库结构的详细说明请阅读 如何为 Mixly 写一个公司库.pdf。

下面讲一讲在开始编辑 xxx.xml、block/xxx.js、generator/xxx.js 时我们需要注意的地方。

1、xxx.xml

```
<!--
type="company"
block="block/STM32.js"
generator="generator/STM32.js"
language="language/STM32/"
examples="examples/STM32"
media="media/STM32"
pin="companypin/STM32"
lib="STM32"
hardware="hardware/arduino"
-->
<!--
Author:
Date:
E-mail:
-->
<script type="text/javascript" src="../../blocks/company/STM32.js"></script>
<script type="text/javascript" src="../../generators/arduino/company/STM32.js"></script>
```

在 xxx.xml 的最顶端，我们需要按照上面两个红框的格式写出各个文件/文件夹的所在位置，或者我们也可以直接拷贝 STM32.xml 里的，然后把 STM32 全部替换为现在的文件名(xxx)。之前我试过不加第一个框里的那些注释，结果发现导入库时 companypin/xxx/pin.js 这个文件没有被导入到 Mixly 里。所以如果我们在做库时添加了 companypin 这个文件夹，一定要记住在 xml 的最顶段添加第一个框的那些注释！

2、block/xxx.js

```
'use strict';
goog.provide('Blockly.Blocks.STM32');
goog.require('Blockly.Blocks');
```

在 block 文件夹里，我们需要把上图的三行代码写在 xxx.js 的最顶端，并把 STM32 替换为现在的文件名(xxx)。

3、generator/xxx.js

```
'use strict';
goog.provide('Blockly.Arduino.STM32');
goog.require('Blockly.Arduino');
```

在 generator 文件夹里，我们需要把上图的三行代码写在 xxx.js 的最顶端，并把 STM32 替换为现在的文件名(xxx)。

三、make 库结构介绍

1、分类①介绍

分类①(显示界面)用于查看制作出的图像化模块的外观。

当未开始制作图形化模块时，分类①中块外观可能如下图所示。会发现只有一个黑块，这是一个正常的现象；当制作块时，这里将实时显示块的外观。

2、分类②介绍

分类②(输入)用于定义图形化模块的输入接口。

输入接口分为以下三种：

其中 NAME 可以看做是这个输入接口的 ID，在 generator/xxx.js 文件里，

我们通过这个输入接口的 ID 来获取输入接口中的数据。一定要牢记这三个输入模块所对应的输入接口，后面在做图形化模块时，会依据需要来选择不同的输入接口。

3、分类③介绍

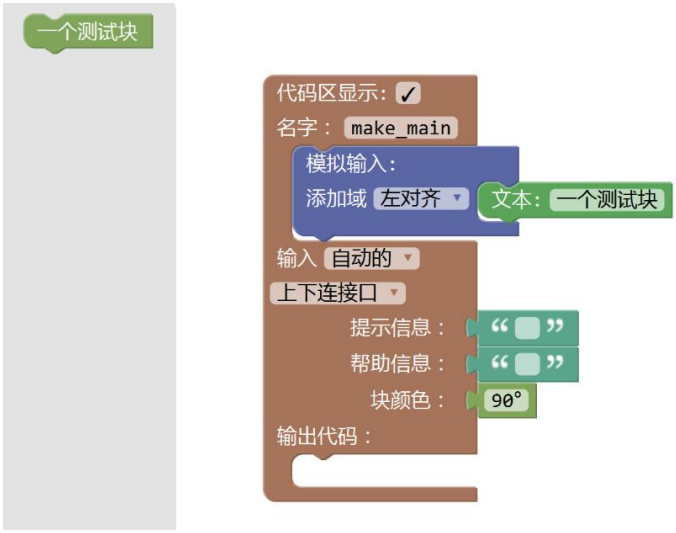
分类③(类型)用于在三个输入接口左侧添加不同的数据输入类型。

3.1 分类③中块介绍

3.1.1 文本块

当在某个输入模块上添加文本块，就可以在这个输入接口的左侧添加一段文字。

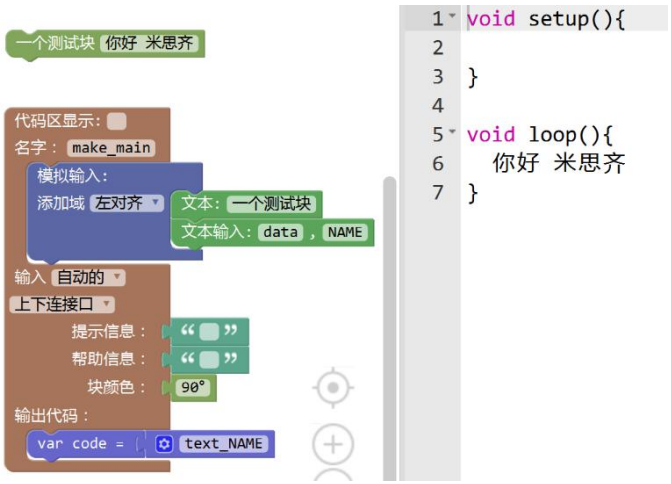
示例：



3.1.2 文本输入块

当在某个输入模块内添加文本输入块，就可以在这个输入接口的左侧添加一个文本输入框。其中 data 是这个输入框中的初始数据，NAME 可以看做是这个文本框的 ID，在 generator/xxx.js 文件里，我们通过这个文本框的 ID 来获取文本框中的数据。

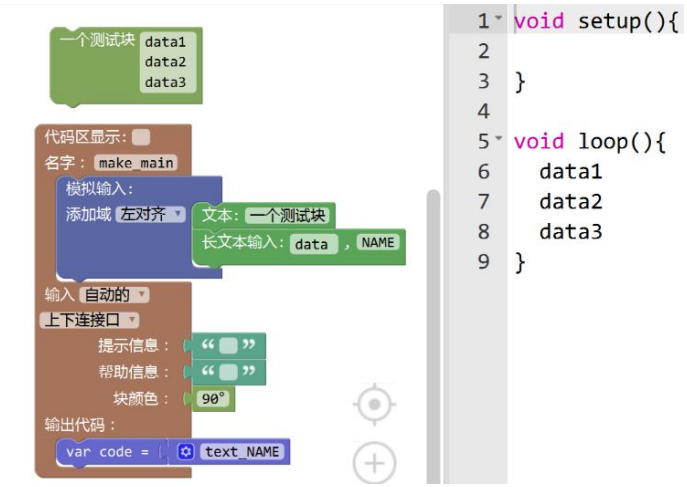
示例：



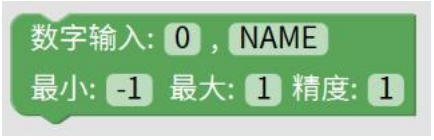
3.1.3 长文本输入块

当在某个输入模块内添加长文本输入块，就可以在这个输入接口的左侧添加一个长文本输入框，这个输入框和上一个的区别是，这个输入框可以显示更长的数据并且可以多行输入。其中 data 是这个输入框中的初始数据，NAME 可以看做是这个文本框的 ID，在 generator/xxx.js 文件里，我们通过这个文本框的 ID 来获取文本框中的数据。

示例：

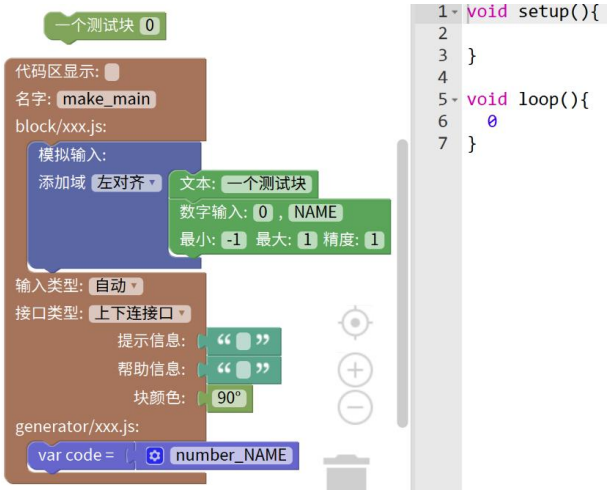


3.1.4 数字输入块



当在某个输入模块内添加数字输入块，就可以在这个输入接口的左侧添加一个数字输入框，这个输入框和上两个的区别是，这个输入框只可以输入数值，并且输入的数值会有一定的约束范围及精度。其中 0 是这个输入框中的初始数据，NAME 可以看做是这个文本框的 ID，在 generator/xxx.js 文件里，我们通过这个文本框的 ID 来获取文本框中的数据。

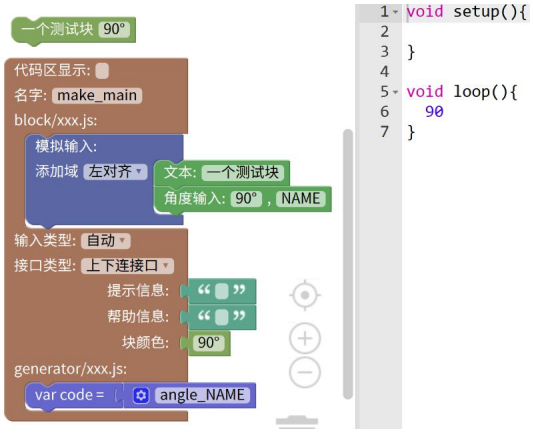
示例：



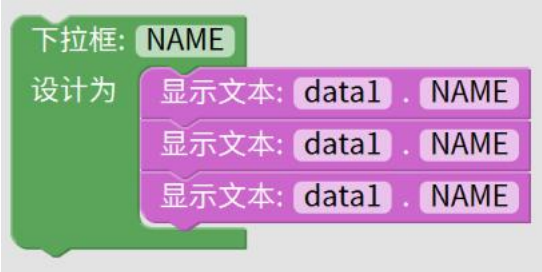
3.1.5 角度输入块

当在某个输入模块内添加角度输入块，就可以在这个输入接口的左侧添加一个角度选择框。其中 90° 是这个输入框中的初始数据，NAME 可以看做是这个文本框的 ID，在 generator/xxx.js 文件里，我们通过这个角度输入框的 ID 来获取输入框中的数据。

示例：

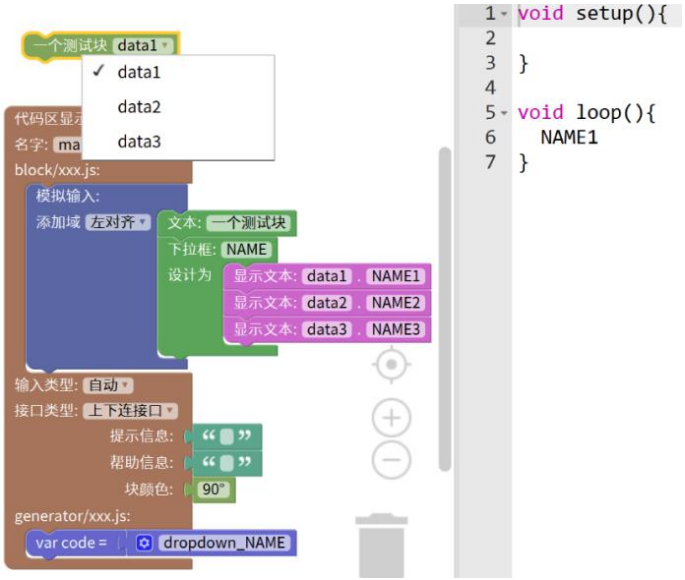


3.1.6 下拉框块

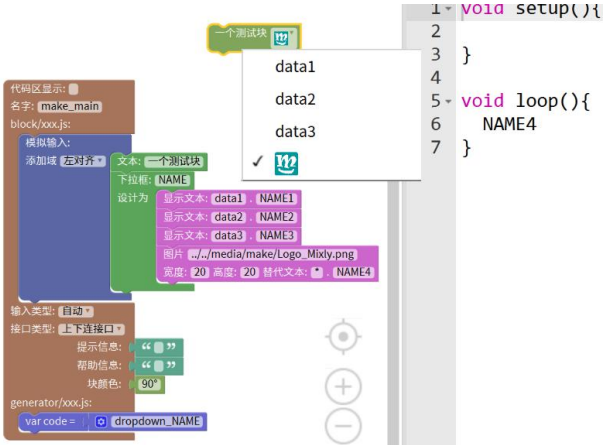


当在某个输入模块内添加下拉框块，就可以在这个输入接口的左侧添加一个下拉框。其中 data1 、data1、data1 是这个下拉框中的初始数据，NAME 可以看做是这个文本框的 ID，在 generator/xxx.js 文件里，我们通过这个下拉框的 ID 来获取下拉框中的数据。这里我们需要注意的一点是，在获取数据时，获取的并不是 data1，而是 data1 右边的 NAME。

示例 1(下拉数据全是文本)：



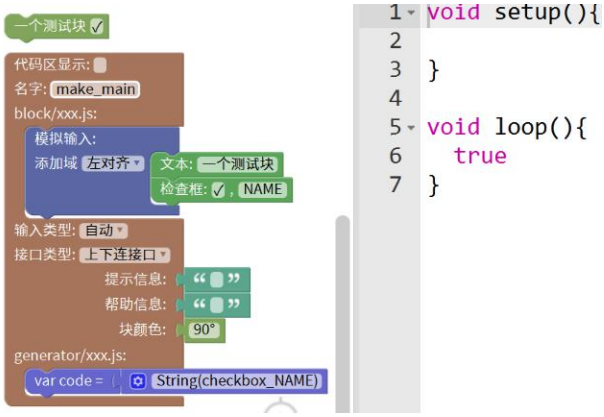
示例 2(下拉数据为文本+图片)：



3.1.7 检查框块

当在某个输入模块内添加检查框块，就可以在这个输入接口的左侧添加一个下拉框。其中 √ 是这个下拉框中的初始数据，NAME 可以看做是这个文本框的 ID，在 generator/xxx.js 文件里，我们通过这个检查框的 ID 来获取检查框中的数据。

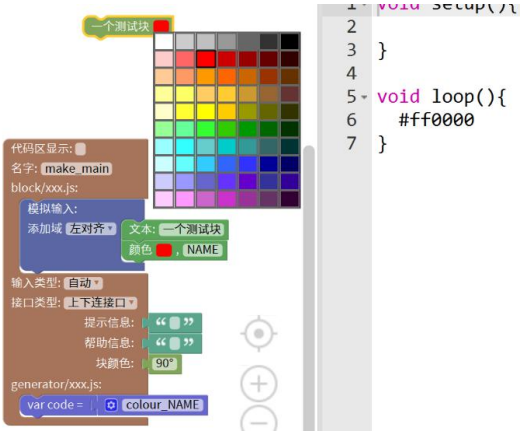
示例：



3.1.8 颜色块

当在某个输入模块内添加颜色块，就可以在这个输入接口的左侧添加一个颜色选择框。其中红色是这个颜色选择框中的初始数据，NAME 可以看做是这个文本框的 ID，在 generator/xxx.js 文件里，我们通过这个颜色选择框的 ID 来获取选择框中的数据。

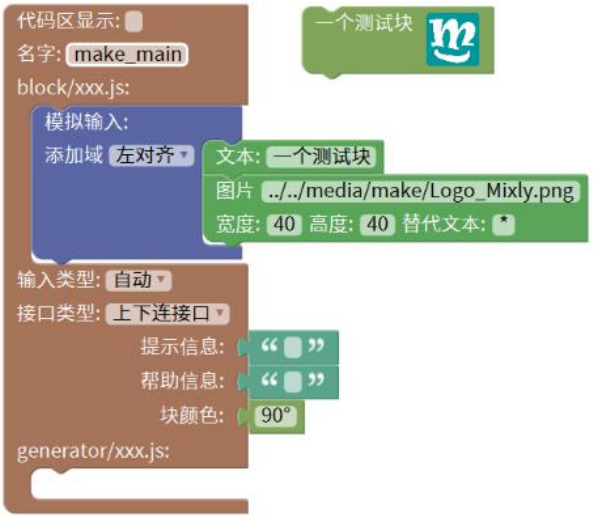
示例：



3.1.9 图片块

当在某个输入模块内添加图片块，就可以在这个输入接口的左侧添加一张图片。其中替代文本是指当在所给的路径上没有找到图片时，将会用此文本来代替图片。

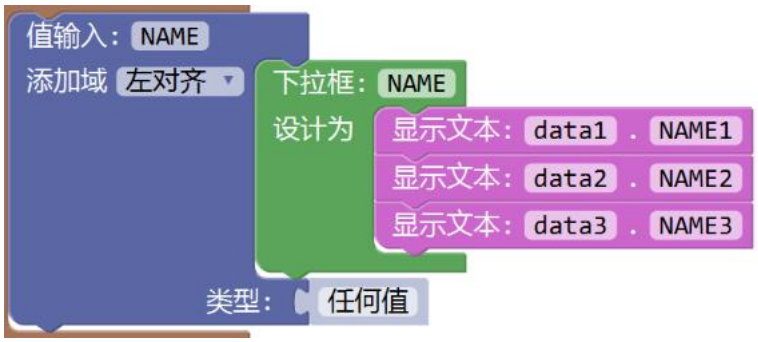
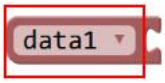
示例：



3.2 值输入接口+分类③块

当为值输入接口时，以下列出此接口左侧可添加的不同的输入类型：

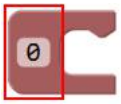
	
	
	

3.3 声明输入接口+分类③块

当为声明输入接口时，以下列出此接口左侧可添加的不同的输入类型：

声明输入：NAME 添加域 左对齐 文本输入：data，NAME 类型：任何值	
声明输入：NAME 添加域 左对齐 长文本输入：data，NAME 类型：任何值	
声明输入：NAME 添加域 左对齐 数字输入：0，NAME 最小：-1 最大：1 精度：1 类型：任何值	
声明输入：NAME 添加域 左对齐 角度输入：90°，NAME 类型：任何值	
声明输入：NAME 添加域 左对齐 下拉框：NAME 设计为 显示文本：data0，NAME0 显示文本：data1，NAME1 显示文本：data2，NAME2 类型：任何值	
声明输入：NAME 添加域 左对齐 检查框：✓，NAME 类型：任何值	
声明输入：NAME 添加域 左对齐 颜色，NAME 类型：任何值	
声明输入：NAME 添加域 左对齐 图片 ../../media/make/Logo_Mixly.png 宽度：20 高度：20 替代文本：* 类型：任何值	

3.4 模拟输入接口+分类③块

当为模拟输入接口时，以下列出此接口左侧可添加的不同的输入类型：

模拟输入： 添加域 左对齐 文本：第一个图形化模块	第一个图形化模块
模拟输入： 添加域 左对齐 文本输入：data , NAME	data
模拟输入： 添加域 左对齐 长文本输入：data , NAME	data
模拟输入： 添加域 左对齐 数字输入：0 , NAME 最小：-1 最大：1 精度：1	0
模拟输入： 添加域 左对齐 角度输入：90° , NAME	90°
模拟输入： 添加域 左对齐 下拉框：NAME 设计为 显示文本：data1 . NAME1 显示文本：data2 . NAME2 显示文本：data3 . NAME3	data1
模拟输入： 添加域 左对齐 检查框：✓ , NAME	✓
模拟输入： 添加域 左对齐 颜色 , NAME	
模拟输入： 添加域 左对齐 图片 ../../media/make/Logo_Mixly.png 宽度：20 高度：20 替代文本：*	

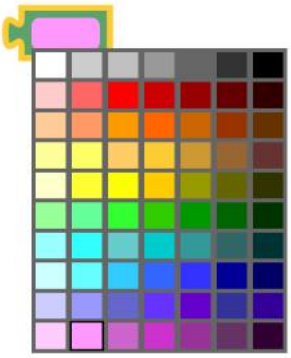
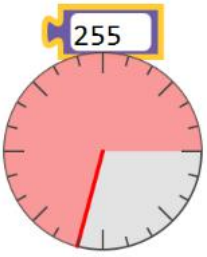
4、分类④介绍

分类④(颜色)用于定义图形化模块的颜色，此分类下有三种颜色图形化模块供我们选择。

第一种是一张色表，共有 70 种颜色。

第二种是一个度数选择器，不同的度数对应不同的颜色，其中度数的取值范围为[0-360),当我们用鼠标选取度数时，只可以取能被 15 整除的数，而如果我们想从上取任意一个数，可以在出现圆盘时，先用 backspace 删除圆盘上方输入框的数据，而后从键盘上输入一个数，再按 enter 键则输入框中就录入了刚刚输入的数据。

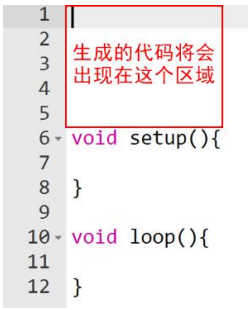
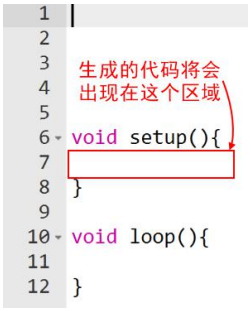
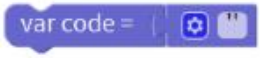
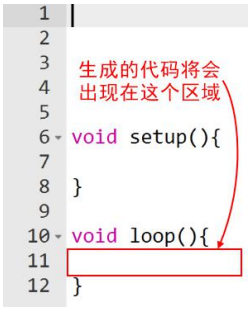
第三种是一个字符串输入框，我们可以根据下面的色表填入自己喜欢颜色的十六进制颜色值。

<table>
<tr><td>  </td><td>  </td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>  </td><td>  </td></tr>
</table>

如果以上两个块都没有喜欢的颜色，则可以浏览此文档附录部分的色表选取对应的十六进制颜色值。

5、分类⑤介绍

分类⑤(编辑生成代码)用于编辑图形化模块所要生成的代码

解析 Blockly.Arduino.definitions_[NAME] = text;
与 Blockly.Arduino.setups_[NAME] = text;

在这两段代码中：

definitions 表示生成代码将会出现在 setup()上面，也就是头文件、全局变量声明等的那个区域；

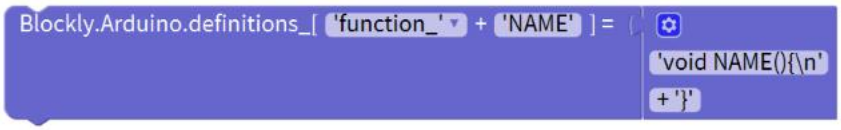
setups 表示生成代码将会出现在 setup()里面。

NAME 可以理解为生成代码的 ID，生成代码不同则 ID 也必须不同，如果 ID 相同了，将会出现后面进入工作区的图形化模块生成的代码替换了之前生成代码的情况，这种情况下有可能会导致编译报错；NAME 可以是一个文本，也可以是一个变量。

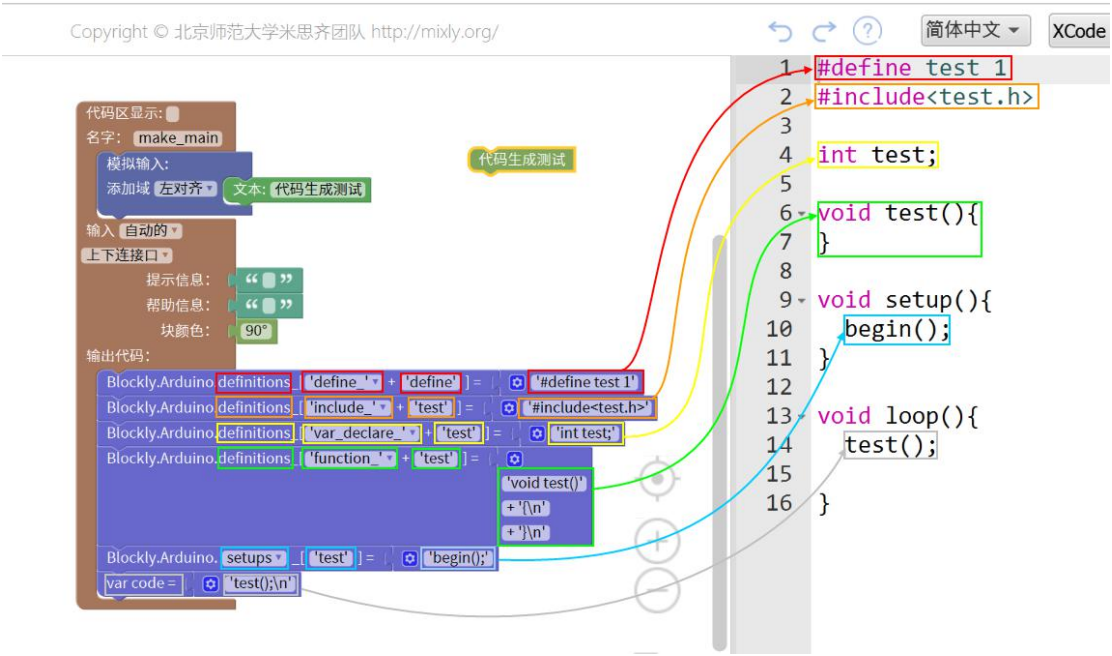
text 则为所要生成的那段代码的数据；text 可以是一个文本，也可以是一个变量。

接下来介绍几个 ID 拥有特殊前缀的图形化模块

	所要生成头文件代码的名字通常为 'include_' +

	头文件名称
	所要生成 define 代码的名字通常为 ‘define_’ +define 名
	所要生成变量代码的名字通常为 ‘var_declare_’ +变量名
	所要生成函数代码的名字通常为 ‘function_’+函数名

以下是生成代码 ID 和代码数据的书写格式以及代码生成后的位置示意图



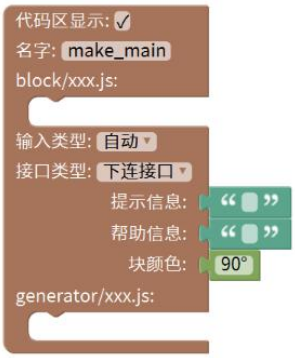
在 Mixly(1.0)之前的版本中，对 ID 有着特殊前缀的生成代码，会有一个特殊的的生成位置，Mixly(1.0)后好像把这个取消了。

虽然最新版本取消了这种代码生成的方式，但是我们在为生成代码选 ID 时，建议还是添加一个特定的前缀，比如生成变量、数组代码时前缀都加 var_declare，生成函数代码时前缀都加 function；这样日后修改代码时，查找以前代码也比较方便。

6、分类⑥介绍

分类⑥(定义块外观/代码)用于制作块外观和查看块在三个文件(xxx.xml、block/xxx.js、generator/xxx.js)下的代码。

6.1 制作块介绍



上图的块是制作块时所用到的主体部分，其中 block/xxx.js:下方通常放的是分类②(输入)+分类③(类型)中的图形化模块；generator/xxx.js:下方通常放的是分类⑤(编辑生成代码)中的图形化模块，一定不要弄错哟。

6.1.1 代码区显示

当我们勾选了代码区显示，点击上图的代码后，就可以在代码区看到生成的三个文件(xxx.xml、block/xxx.js、generator/xxx.js)。

6.1.2 块名字

每次我们在制作一个新块时，都必须要先给这个块起一个名字，并且要保证这个块的名字和已有的其他块的名字是不重复的。如果存在两个块名字相同，那么在导入库后，我们所做的这个块就有可能会出现问题。

6.1.3 输入类型

输入类型共有以下三种:

自动	
外部输入	
单行输入	

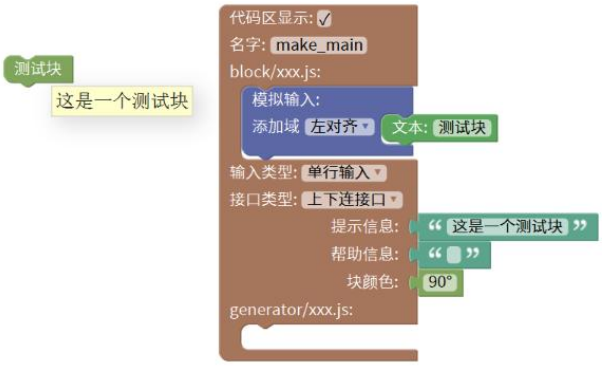
6.1.4 接口类型

接口类型共有以下五种:

无连接口	
上连接口	

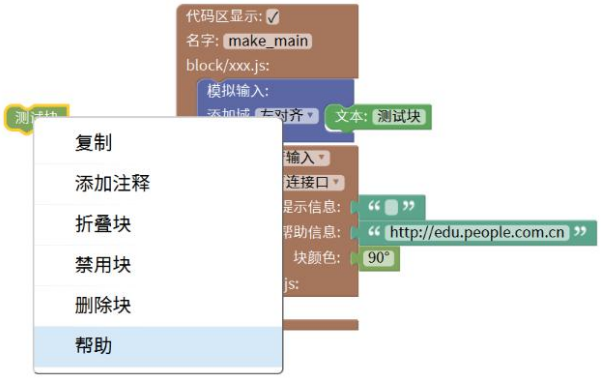
下连接口	
上下连接口	
左连接口	

6.1.5 提示信息

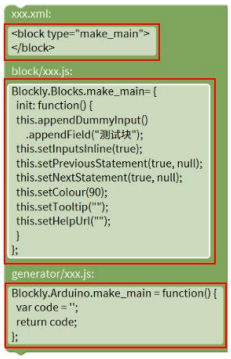


6.1.6 帮助信息

当我们在块的帮助信息里写入某一网址后，选择这个块单击右键选择帮助，就可以在网页中打开这一网址。



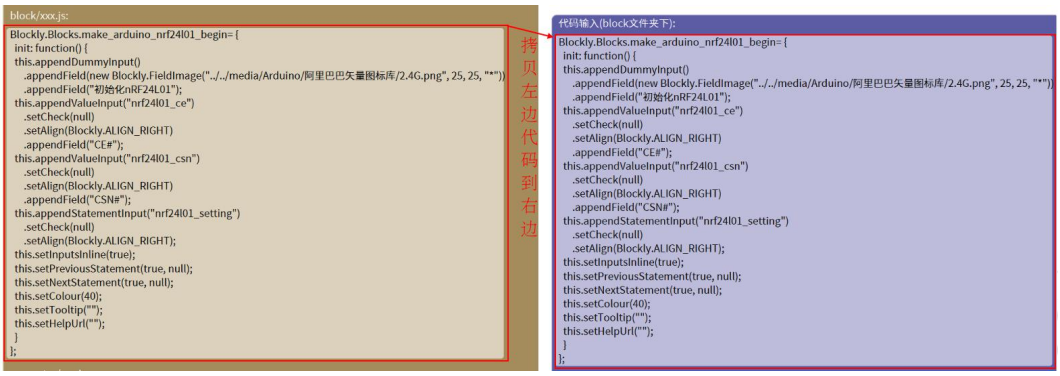
6.2 查看块介绍



当我们在制作块时，可以用这个块来同步查看生成代码的情况。当块制作完成后，我们可以直接复制三个框中的代码到各自的文件里。

7、分类⑦介绍

分类⑦(转换工具(block→block+zh_hans))用于让制作出的库可以支持多种语言。



当我们希望做出的库可以支持其他语言时，可以复制出块在 block/xxx.js 下代码，而后用分类⑦中的块提取出这段代码中所有带有简体中文的字符串，提取效果如下图。

其中输出部分顶端的那一段代码将取代原有的输入代码；底端的代码则放入到 language/xxx/zh_hans.js、zh-hant.js、spa.js、en.js 里，其中放入到 zh-hant.js、spa.js、en.js 的代码里将带有简体中文的字符串用谷歌翻译分别翻译成繁体中文、西班牙语、英语，而后替换原来的简体中文字符串即可。

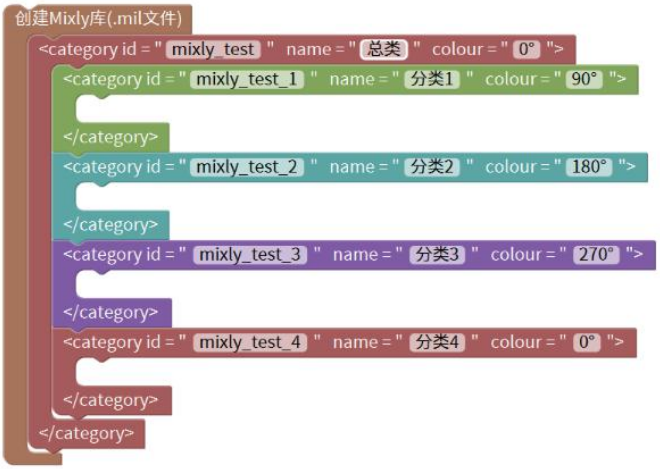
8、分类⑧介绍

分类⑧(基础块)用于制作库中的各个分类。

下面举个栗子：

下图是一个用分类⑧拼接出的分类结构，在这个结构里，有一个总类，这个

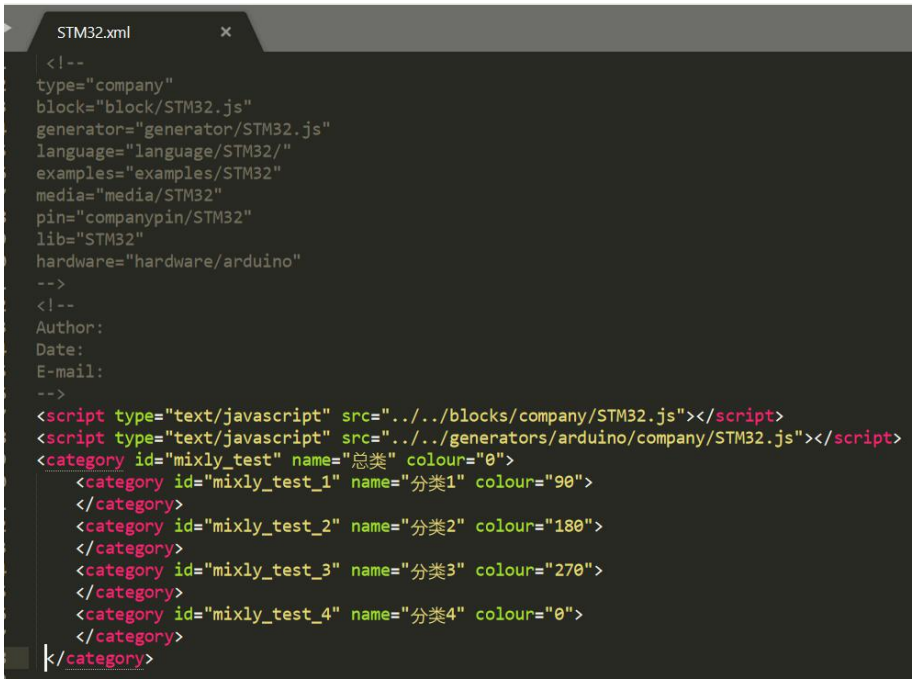
总类里包含了四个分类(分类 1 - 分类 4)。



拷贝代码区中下图红色框中代码到 xxx.xml

```
<!-- lib="" -->
<xml board="mylib" xmlns="http://www.w3.org/1999/xhtml">
  <category id="mixly_test" name="总类" colour="0">
    <category id="mixly_test_1" name="分类1" colour="90">
    </category>
    <category id="mixly_test_2" name="分类2" colour="180">
    </category>
    <category id="mixly_test_3" name="分类3" colour="270">
    </category>
    <category id="mixly_test_4" name="分类4" colour="0">
    </category>
  </category>
</xml>
```

以 STM32.xml 为例，拷贝后代码放置的位置如下图。



当再次导入 STM32 库时，就会发现出现了一个下拉框。

9、分类⑨介绍

分类⑨(示例)是分类⑧(基础块)的一个例子。

10、分类⑩介绍

分类⑩(测试)是一个测试界面，对于一些不太确定代码是否正确的块可以先放入这个界面来调试。

四、对于制作时可能遇到的一些情况的说明

1、导入库后出现黑块

出现黑块的原因有很多，比如某些块的名字和其他块重名了，或者就是代码有问题等等。不过出现黑块如果不是块重名则很大概率是 block/xxx.js 里的代码有问题，我们找到出现黑块位置的代码，仔细检查有没有语句错误或漏写分号、大括号啥的。

2、点到代码区后无法返回到模块区

这种情况出现的原因是由于 generator/xxx.js 里的代码有错误。

有的时候我们我们拖动一个块到模块区就出现了这种情况，但是拖动其他块到模块区一切却都是正常的，那我们就找到了代码出错的位置，就是刚刚那个块的 generator/xxx.js 里的代码有错，找到那个块在 generator/xxx.js 里的代码，仔细检查有没有语句错误或漏写啥东西。

3、导入库后发现有些块的外观不是之前制作的那个

例如：

为啥比之前的变量声明多了一个下拉框？

如果我们打开 Arduino 库在 block/Arduino.js 和 generator/Arduino.js 下的代码，就会发现下图这段代码。

Block/Arduino.js:

```
//更改mixly中变量声明的图形块
Blockly.Blocks['variables_declare'] = {
  // Variable setter.
  init: function() {
    var DATATYPES = [["整数","int"],["无符号整数","unsigned int"],["长整数"

    this.setColour(Blockly.Blocks.variables.HUE);
    this.appendValueInput('VALUE', null)
        .appendField(Blockly.MIXLY_DECLARE)
        .appendField(new Blockly.FieldDropdown([["全局变量","global_varia
        .appendField(new Blockly.FieldTextInput('item'), 'VAR')
        .appendField(Blockly.MIXLY_AS)
      .appendField(new Blockly.FieldDropdown(DATATYPES), "TYPE")
      .appendField(Blockly.MIXLY_VALUE);
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip(Blockly.MIXLY_TOOLTIP_VARIABLES_DECLARE);
  },
  getVars: function() {
    return [this.getFieldValue('VAR')];
  },
  renameVar: function(oldName, newName) {
    if (Blockly.Names.equals(oldName, this.getFieldValue('VAR'))) {
      this.setTitleValue(newName, 'VAR');
    }
  }
};
```

Generator/Arduino.js:

```
//更改mixly中变量声明的图形块
Blockly.Arduino.variables_declare = function() {
  var dropdown_type = this.getFieldValue('TYPE');

  var dropdown_variables_type = this.getFieldValue('variables_type');

  var argument0;

  var code = '';
  //TODO: settype to variable
  if(dropdown_variables_type == 'global_variate')
  {
    if(dropdown_type=='String'){
      argument0 = Blockly.Arduino.valueToCode(this, 'VALUE',Blockly.Arduino.ORDER_ASSIGNMENT) || '""';
    }else{
      argument0 = Blockly.Arduino.valueToCode(this, 'VALUE',Blockly.Arduino.ORDER_ASSIGNMENT) || '0';
    }
    var varName = Blockly.Arduino.variableDB_.getName(this.getFieldValue('VAR'),
        Blockly.Variables.NAME_TYPE);

    if (dropdown_type == 'String' || dropdown_type == 'uint8_t' || dropdown_type == 'uint16_t' || dropd
      Blockly.Arduino.definitions_['var_declare' + varName] = dropdown_type + ' ' + varName + ';';
    else
      Blockly.Arduino.definitions_['var_declare' + varName] = 'volatile ' + dropdown_type + ' ' + varNa

    Blockly.Arduino.setups_['setup_var'+varName] = varName + ' = ' + argument0 + ';';
  }
  else if(dropdown_variables_type == 'local_variate')
  {
    if(dropdown_type=='String'){
      argument0 = Blockly.Arduino.valueToCode(this, 'VALUE',Blockly.Arduino.ORDER_ASSIGNMENT) || '""';
    }else{
```

Arduino 库中有一个块的名字和 Mixly 软件里的变量定义块名字相同，导致

Mixly 软件里的块的代码失去了作用。所以如果发现所做的块有外观变了的或直接是黑块，那可能就是这个块的名字和其他块冲突了。

不过刚刚那个用库中代码覆盖 Mixly 里的方法最好不要用，因为如果给覆盖了，下一次再更新 Mixly 时，如果这个块有更新我们是无法看到的！

4、几个块拼接是如何放到分类里面的

例如：

这种拼接块我们用分类⑩(测试)里的那个块就可以做出来。

具体步骤如下：

(1) 用 Mixly 中块拼接出一个我们想要的拼接块。

(2) 将分类⑩(测试)里的那个测试块拖到模块区，注意一定不要和那个拼接块拼接在一起。

(3) 进入代码区，复制<block>...</block>中的代码(包括<block></block>)。

(4) 粘贴这段代码到 xxx.xml 中的一个分类里，及<category>...</category>中的...里。

(5) 再次导入库就可以看到刚刚的那个拼接块。

示例：

（1）

（2）

（3）

```
<xml xmlns="http://www.w3.org/1999/xhtml">
  <block type="controls_whileUntil">
    <field name="MODE">WHILE</field>
    <value name="BOOL">
      <shadow type="logic_boolean">
        <field name="BOOL">TRUE</field>
      </shadow>
    </value>
    <statement name="DO">
      <block type="serial_println">
        <field name="serial_select">Serial</field>
        <value name="CONTENT">
          <block type="text">
            <field name="TEXT">一个测试块</field>
          </block>
        </value>
      </block>
    </statement>
  </block>
</xml>
```

（4）

```
<script type="text/javascript" src="../../blocks/company/STM32.js"></script>
<script type="text/javascript" src="../../generators/arduino/company/STM32.js"></script>
<category id="mixly_test" name="总类" colour="0">

    <category id="mixly_test_1" name="分类1" colour="90">

        <block type="controls_whileUntil">
            <field name="MODE">WHILE</field>
            <value name="BOOL">
              <shadow type="logic_boolean">
                <field name="BOOL">TRUE</field>
              </shadow>
            </value>
            <statement name="DO">
              <block type="serial_println">
                <field name="serial_select">Serial</field>
                <value name="CONTENT">
                  <block type="text">
                    <field name="TEXT">一个测试块</field>
                  </block>
                </value>
              </block>
            </statement>
          </block>

    </category>

    <category id="mixly_test_2" name="分类2" colour="180">
    </category>
    <category id="mixly_test_3" name="分类3" colour="270">
    </category>
```

（5）

5、有些块的输入接口上连了一个隐藏的块是如何做的

例如：

如果想把红色框中的块变成隐藏的，那我们还需要用到刚刚的那个测试块。

（1）把上图块和测试块都放入模块区，注意两个块不可以拼接在一起。

（2）打开工作区，查看这个这个块的 xml 文件。

```
<xml xmlns="http://www.w3.org/1999/xhtml">
  <block type="math_random_seed">
    <value name="NUM">
      <shadow type="math_number">
        <field name="NUM">997</field>
      </shadow>
      <block type="controls_millis">
        <field name="UNIT">millis</field>
      </block>
    </value>
  </block>
</xml>
```

（3）会发现有一个<block type="controls_millis">...</block>，
我们把这段语句中首尾两处 block 修改为 shadow，修改后的代码如下图。

```
<xml xmlns="http://www.w3.org/1999/xhtml">
  <block type="math_random_seed">
    <value name="NUM">
      <shadow type="math_number">
        <field name="NUM">997</field>
      </shadow>
      <shadow type="controls_millis">
        <field name="UNIT">millis</field>
      </shadow>
    </value>
  </block>
</xml>
```

（4）复制修改后的代码到 xxx.xml 中的一个分类里。

```
<category id="mixly_test_1" name="分类1" colour="90">

    <block type="controls_whileUntil">
        <field name="MODE">WHILE</field>
        <value name="BOOL">
          <shadow type="logic_boolean">
            <field name="BOOL">TRUE</field>
          </shadow>
        </value>
        <statement name="DO">
          <block type="serial_println">
            <field name="serial_select">Serial</field>
            <value name="CONTENT">
              <block type="text">
                <field name="TEXT">一个测试块</field>
              </block>
            </value>
          </block>
        </statement>
      </block>

      <block type="math_random_seed">
        <value name="NUM">
          <shadow type="math_number">
            <field name="NUM">997</field>
          </shadow>
          <shadow type="controls_millis">
            <field name="UNIT">millis</field>
          </shadow>
        </value>
      </block>

</category>
```

（5）再次导入库后，就会发现之前那个红色框中的块变成了隐藏的。

从这个例子里我们可以总结出：

（1）用<block>...</block>定义出的块是可显示的。

（2）用<shadow>...</shadow>定义出的块是隐藏的。

6、左边带有警告标示的块是如何做出来的

例如：

这个功能主要写在 block/xxx.js 文件里，用到 onchange 这个函数，具体代码如下。

```
Blockly.Blocks['make_value_input'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("值输入:")
        .appendField(new Blockly.FieldTextInput("NAME"), "dummy_input_data");
    this.appendStatementInput("statement_input_data")
        .setCheck("make_type")
        .appendField("添加域")
        .appendField(new Blockly.FieldDropdown([["左对齐","left"], ["右对齐","rig
    this.appendValueInput("make_value_input_type")
        .setCheck(null)
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField("类型:");
    this.setPreviousStatement(true, "make_input");
    this.setNextStatement(true, "make_input");
    this.setColour(230);
 this.setTooltip("");
 this.setHelpUrl("");
  },
  onchange: function(e) {
    var surround_parent = this.getSurroundParent();
    if (surround_parent && surround_parent.type == 'make_main') {
      this.setWarningText(null);
    } else {
      this.setWarningText("此块需放到外观块下面");
    }
  }
};
```

```
var surround_parent = this.getSurroundParent();
```

这段代码好像是获取与这个块接口所连接的其他块的名字。

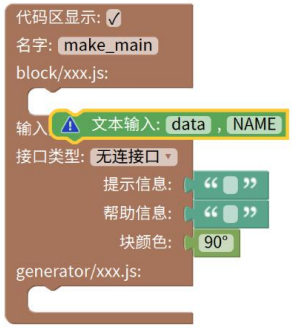
```
if (surround_parent && surround_parent.type == 'make_main') {
  this.setWarningText(null);
} else {
  this.setWarningText("此块需放到外观块下面");
}
```

这几段代码好像是指当这个块接口连了其他块，并且所连接块的名字叫'make_main'时，不显示警告；否则显示警告信息"此块需放到外观块下面"。

在做块时如果要用这个功能，那我们直接复制 make 库中的，再粘贴修改即可。

7、Mixly 中有些块虽然接口一样却无法拼到一起，这个是如何做出来的

例如：



这种情况是由于这个声明接口启用了核对功能，如下图红框中所示。

```
Blockly.Blocks['make_main'] = {
  init: function() {
  this.appendDummyInput()
      .appendField("代码区显示:")
      .appendField(new Blockly.FieldCheckbox("true"), "main_show_is_true");
  //this.appendDummyInput()
  //    .appendField("使用显示界面中块")
  //    .appendField(new Blockly.FieldDropdown([["1","1"],["2","2"],["3","3"
  this.appendDummyInput()
      .appendField("名字:")
      .appendField(new Blockly.FieldTextInput("make_main"), "main_0");
  this.appendDummyInput()
      .appendField("block/xxx.js:");
  this.appendStatementInput("main_1")
      .setCheck("make_input");
  this.appendDummyInput()
      .appendField("输入类型:")
      .appendField(new Blockly.FieldDropdown([["自动","automatic"], ["外部输
  this.appendDummyInput()
      .appendField("接口类型:")
      .appendField(new Blockly.FieldDropdown([["无连接口","sharp_1"], ["上连接
  this.appendValueInput("main_2")
      .setCheck(null)
      .setAlign(Blockly.ALIGN_RIGHT)
      .appendField("提示信息:");
  this.appendValueInput("main_3")
      .setCheck(null)
      .setAlign(Blockly.ALIGN_RIGHT)
      .appendField("帮助信息:");
  this.appendValueInput("main_4")
      .setCheck(null)
      .setAlign(Blockly.ALIGN_RIGHT)
      .appendField("块颜色:");
  this.appendDummyInput()
```

当发现要连接块的接口 ID 与核对的字符串"make_input"不相符时，那这两个块就无法连在一起。

我们来看看刚刚那个文本输入块的接口 ID 是什么。

```
Blockly.Blocks['make_type_text_input'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("文本输入:")
        .appendField(new Blockly.FieldTextInput("data"), "type_text_data")
        .appendField(",")
        .appendField(new Blockly.FieldTextInput("NAME"), "type_text_name");
    this.setPreviousStatement(true, "make_type");
    this.setNextStatement(true, "make_type");
    this.setColour(120);
 this.setTooltip("");
 this.setHelpUrl("");
  },
  onchange: function(e) {
    var surround_parent = this.getSurroundParent();
    if (surround_parent && (surround_parent.type == 'make_value_input' | surr
      this.setWarningText(null);
    } else {
      this.setWarningText("此块需放到输入块下面");
    }
  }
};
```

由于上、下接口的 ID 都是字符串"make_type"，所以刚刚那两个块无法连到一起，如果我们把字符串"make_type"修改为"make_input"，这两个块就可以被连接。

END

感觉应该讲的差不多了，至于那个带有齿轮的块如何来做，其实我也不太清楚，等过段时间研究明白了再从这里继续写。

如果您（有好的建议||在制作时遇到难题||发现文档有错误）请加 QQ3294713004 联系我哈，谢谢。

附录

颜色名	十六进制颜色值	颜色
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFD4	
Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
Darkorange	#FF8C00	
DarkOrchid	#9932CC	
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBC8F	
DarkSlateBlue	#483D8B	

DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
Darkorange	#FF8C00	
DarkOrchid	#9932CC	
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBC8F	
DarkSlateBlue	#483D8B	
DarkSlateGray	#2F4F4F	
DarkTurquoise	#00CED1	
DarkViolet	#9400D3	
DeepPink	#FF1493	
DeepSkyBlue	#00BFFF	
DimGray	#696969	
DodgerBlue	#1E90FF	
Feldspar	#D19275	
FireBrick	#B22222	
FloralWhite	#FFFAF0	
ForestGreen	#228B22	
Fuchsia	#FF00FF	
Gainsboro	#DCDCDC	
GhostWhite	#F8F8FF	
Gold	#FFD700	
GoldenRod	#DAA520	
Gray	#808080	
Green	#008000	
GreenYellow	#ADFF2F	
HoneyDew	#F0FFF0	
HotPink	#FF69B4	
IndianRed	#CD5C5C	

Indigo	#4B0082	
Ivory	#FFFFF0	
Khaki	#F0E68C	
Lavender	#E6E6FA	
LavenderBlush	#FFF0F5	
LawnGreen	#7CFC00	
LemonChiffon	#FFFACD	
LightBlue	#ADD8E6	
LightCoral	#F08080	
LightCyan	#E0FFFF	
LightGoldenRodYellow	#FAFAD2	
LightGrey	#D3D3D3	
LightGreen	#90EE90	
LightPink	#FFB6C1	
LightSalmon	#FFA07A	
LightSeaGreen	#20B2AA	
LightSkyBlue	#87CEFA	
LightSlateBlue	#8470FF	
LightSlateGray	#778899	
LightSteelBlue	#B0C4DE	
LightYellow	#FFFFE0	
Lime	#00FF00	
LimeGreen	#32CD32	
Linen	#FAF0E6	
Magenta	#FF00FF	
Maroon	#800000	
MediumAquaMarine	#66CDAA	
MediumBlue	#0000CD	
MediumOrchid	#BA55D3	
MediumPurple	#9370D8	
MediumSeaGreen	#3CB371	
MediumSlateBlue	#7B68EE	
MediumSpringGreen	#00FA9A	
MediumTurquoise	#48D1CC	
MediumVioletRed	#C71585	
MidnightBlue	#191970	

MintCream	#F5FFFA	
MistyRose	#FFE4E1	
Moccasin	#FFE4B5	
NavajoWhite	#FFDEAD	
Navy	#000080	
OldLace	#FDF5E6	
Olive	#808000	
OliveDrab	#6B8E23	
Orange	#FFA500	
OrangeRed	#FF4500	
Orchid	#DA70D6	
PaleGoldenRod	#EEE8AA	
PaleGreen	#98FB98	
PaleTurquoise	#AFEEEE	
PaleVioletRed	#D87093	
PapayaWhip	#FFEFD5	
PeachPuff	#FFDAB9	
Peru	#CD853F	
Pink	#FFC0CB	
Plum	#DDA0DD	
PowderBlue	#B0E0E6	
Purple	#800080	
Red	#FF0000	
RosyBrown	#BC8F8F	
RoyalBlue	#4169E1	
SaddleBrown	#8B4513	
Salmon	#FA8072	
SandyBrown	#F4A460	
SeaGreen	#2E8B57	
SeaShell	#FFF5EE	
Sienna	#A0522D	
Silver	#C0C0C0	
SkyBlue	#87CEEB	
SlateBlue	#6A5ACD	
SlateGray	#708090	
Snow	#FFFAFA	

SpringGreen	#00FF7F	
SteelBlue	#4682B4	
Tan	#D2B48C	
Teal	#008080	
Thistle	#D8BFD8	
Tomato	#FF6347	
Turquoise	#40E0D0	
Violet	#EE82EE	
VioletRed	#D02090	
Wheat	#F5DEB3	
White	#FFFFFF	
WhiteSmoke	#F5F5F5	
Yellow	#FFFF00	
YellowGreen	#9ACD32	