



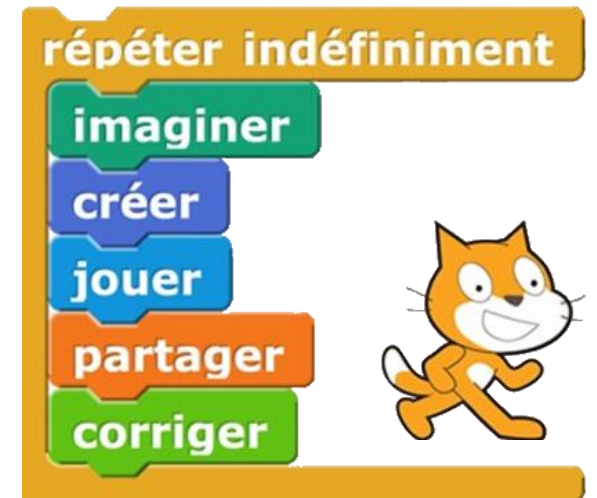
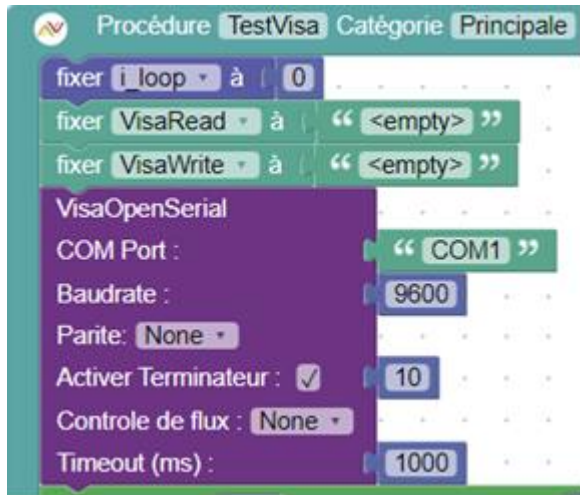
LUGE 2022.4



Nerys Group companies

- Blockly -

Automatiser des actions sous LabVIEW avec l'interface de SCRATCH



Le présentateur



Nerys Group companies

- Amaury LAURENT
- Développeur LabVIEW depuis 2014
- Certifications : CLED et CLA
- Responsable projets systèmes embarqués
- Aéromodéliste

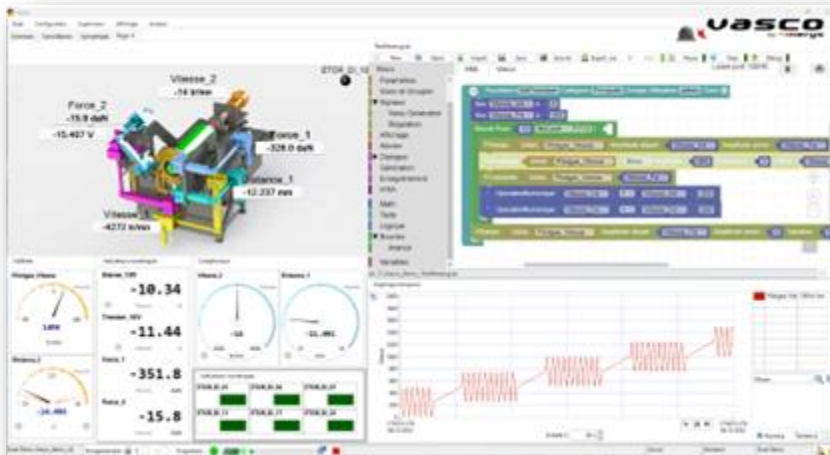
ARCHITECT



EMBEDDED
DEVELOPER



- Fondée en 2007 à Gardanne (Bouches du Rhône), une équipe de 15 personnes ;
- Concepteur de bancs de tests, de caractérisation et de systèmes de mesure :
 - Collaborateurs spécialisés en électrotechnique, électronique et mécanique ;
 - Pôle développement logiciels, avec une expertise LabVIEW.
- NERYS propose une solution personnalisable de pilotage de moyens d'essais appelée "Vasco" pouvant être déployée sur cible temps-réel (NI RT).



vasco
by  Nerys

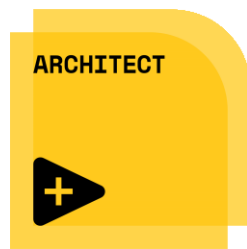


NERYS GROUP en bref



Nerys Group companies

- Depuis Mai 2022 : NERYS + MESULOG → NERYS GROUP
- Compétences logicielles NI :
 - LabVIEW (Windows, RT, DSC, FPGA)
 - TestStand
 - VeriStand
- Partenaire National Instruments
- Développeurs certifiés LabVIEW et TestStand



Plus d'infos : site internet

Télécharger cette présentation technique sur nos sites internet :

<https://nerysgroup.com/fr/entreprise/documentation>



<http://mesulog.fr/presentation>



<https://nerysgroup.com>

<http://mesulog.fr>

Nous contacter



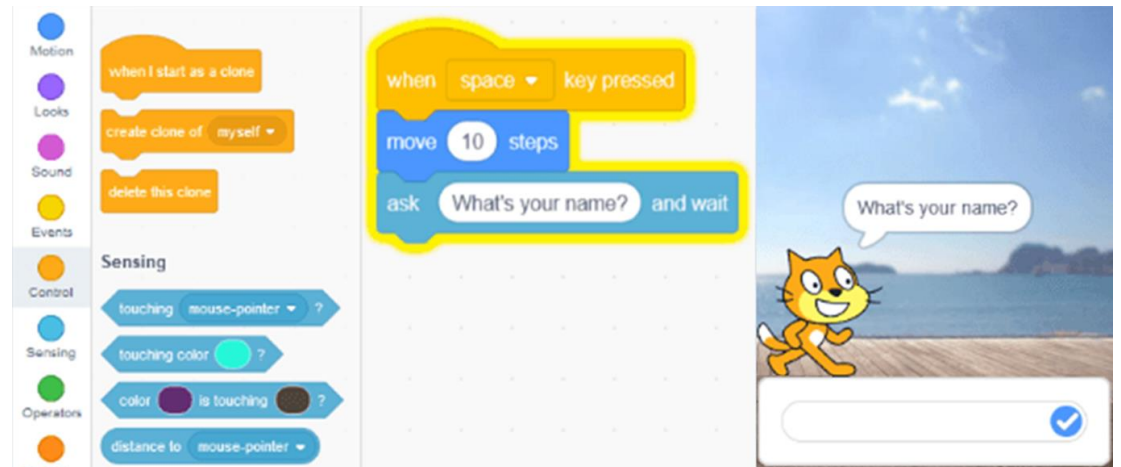
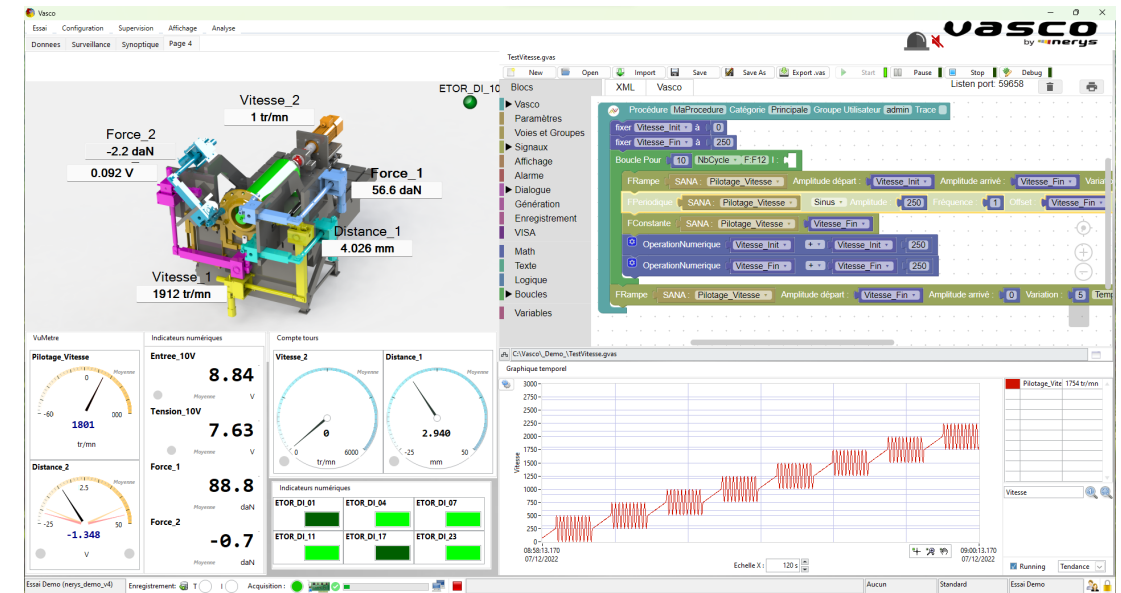
Nerys Group companies

- Nos sites :
 - Gardanne (13)
 - Moirans (38)
 - La Rochette (73)
- Sur le web :
 - <https://nerysgroup.com/fr/>
 - <https://www.mesulog.fr/>
- Nous contacter :
 - contact@nerysgroup.com
 - contact@mesulog.fr



- Projet présenté en 2012
- Open Source
- Bibliothèques JavaScript
- But : faire des interfaces de programmation graphique
- Entièrement personnalisable

- A notamment servi pour:
 - AppInventor (Android)
 - Scratch (MIT)
 - Vasco (NERYS)



Principe d'utilisation de Blockly

Editeur graphique

HTML 5

Multiplateformes

Ecriture du code

Descripteurs block

Javascript

Définitions des
blocks

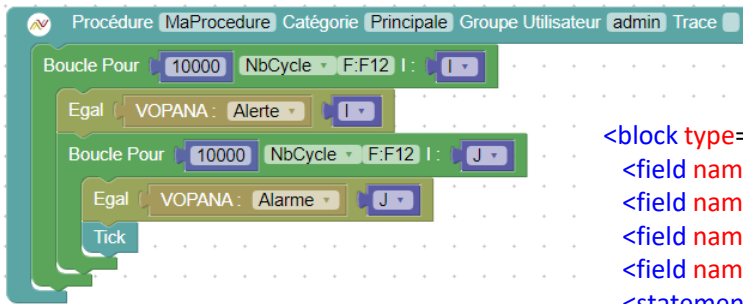
Générateurs code

Javascript

Convertisseur
block -> code

Code source

```
Debut ( MaProcedure , Principale , admin , )  
Pour ( 10000 , NbCycle , F:F12 , , null , , I )  
Egal ( O:Alerte , N:I )  
Pour ( 10000 , NbCycle , F:F12 , , null , , J )  
Egal ( O:Alarme , N:J )  
Tick ( )  
FinPour  
FinPour  
Fin
```

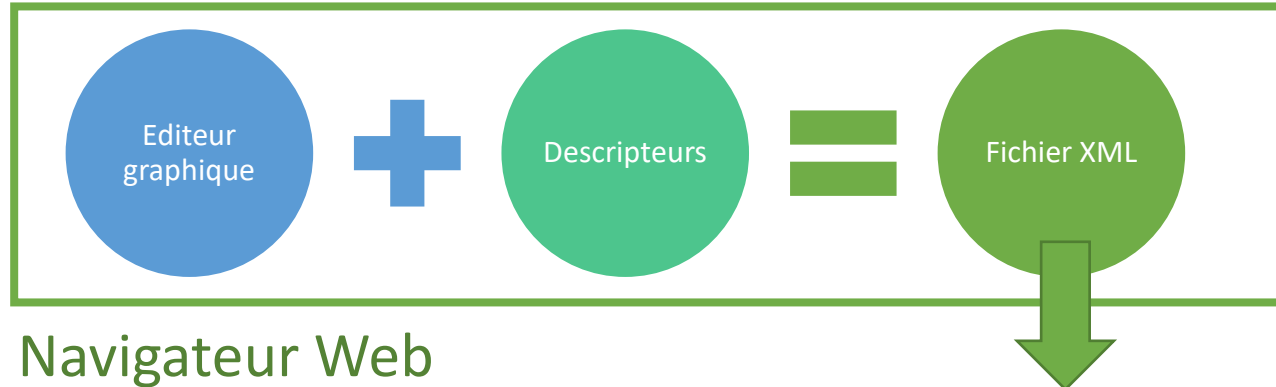


```
<block type="procedure">  
<field name="PROC_NAME">MaProcedure</field>  
<field name="PROC_CAT">Principale</field>  
<field name="PROC_GROUP">admin</field>  
<field name="PROC_TRACE">FALSE</field>  
<statement name="PROC">  
  <block type="bky_for_loop" id="Et:[fwd6Ato!|9c8x:%M]">  
    <field name="UNIT">NbCycle</field>
```

Exemple de descripteur XML

Intégration avec LabVIEW

Principe : Utilisation du fichier XML comme source interprétée par LabVIEW

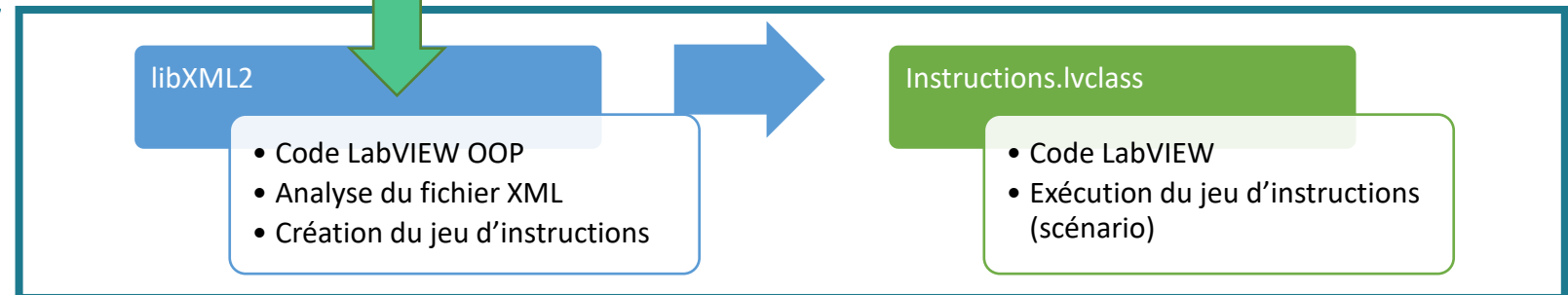


Structure XML :

- Adaptée à une représentation objet
- Récursive
- Simple à lire

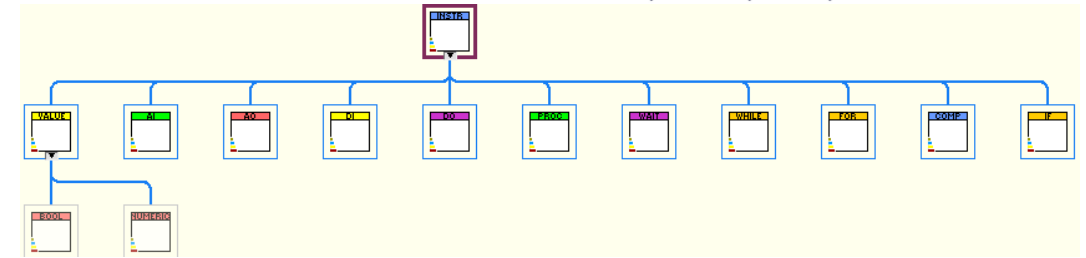
```
<block type="procedure" id="pz*o73e6+Am^zYplvpZ2" x="38" y="38">  
<field name="PROC_NAME">MaProcedure</field>  
<field name="PROC_CAT">Principale</field>  
<field name="PROC_GROUP">admin</field>  
<field name="PROC_TRACE">FALSE</field>  
<statement name="PROC">  
<block type="bky_for_loop" id="Et:[fwd6Ato!|9c8x:%M">  
<field name="UNIT">NbCycle</field>
```

LabVIEW



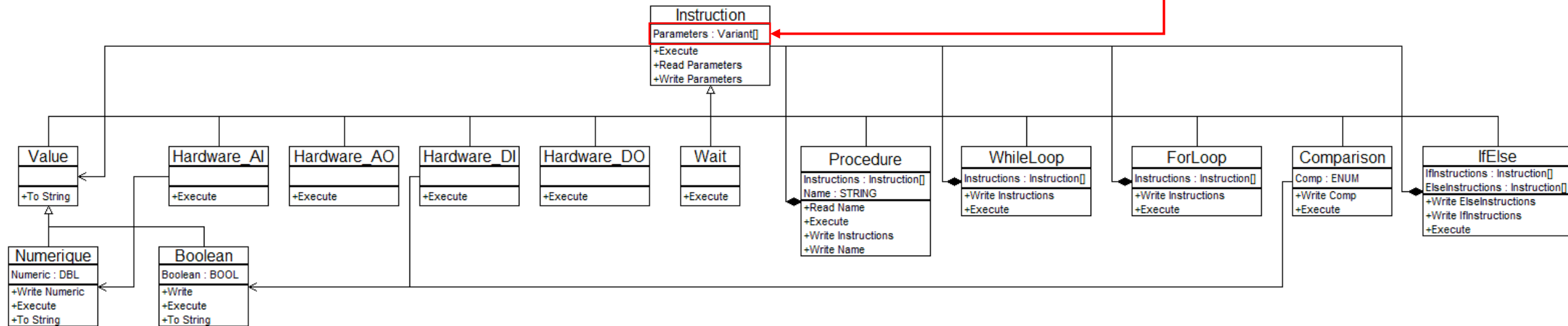
LabVIEW : Gestion des instructions

- Architecture Objets
 - Dispatch dynamique
- Exécution récursive
 - Une seule méthode : Execute
 - LabVIEW = Execute.vi



NB : Parameters : Instruction[]

Astuce pour permettre à un objet de contenir des instances de lui-même : le variant

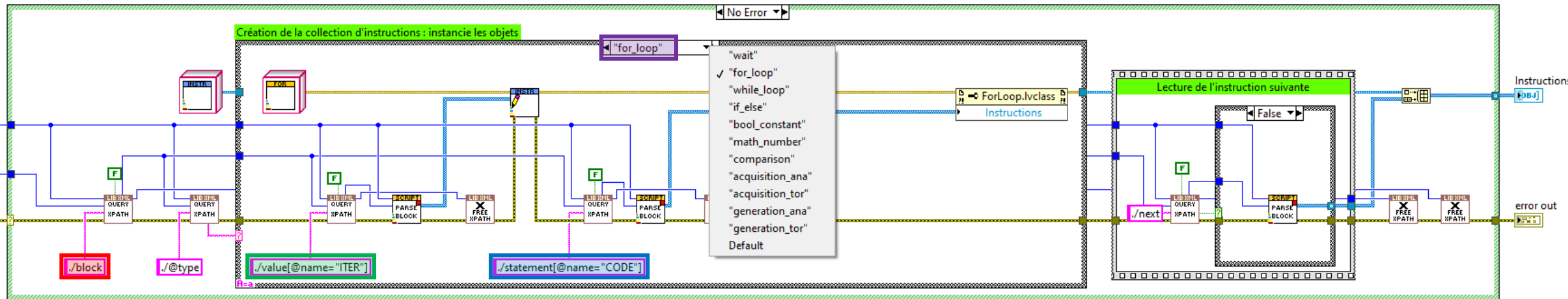


LabVIEW : Analyse du XML

LibXML2 sous LabVIEW

- Utilisation des XPath
- Lecture récursive

```
<block type="for_loop" id=":|8-T_rWn@-JjY573Tqk">  
  <value name="ITER">  
    <block type="math_number" id="/'AzaFG2xxN-L|1py^%6">  
      <field name="NUM">10</field>  
    </block>  
  </value>  
  <statement name="CODE">  
    <block type="wait" id="!+4ik@.Ms$GVpXnblq9">  
      <value name="DURATION">  
        <block type="math_number" id="@#kFEYc))7%jYcOvRwQ7">  
          <field name="NUM">5</field>  
        </block>  
      </value>  
    </block>  
  </statement>  
</block>
```



[Lien vers le toolkit LabXML \(SourceForge\)](#)

Exécution récursive : Execute.vi

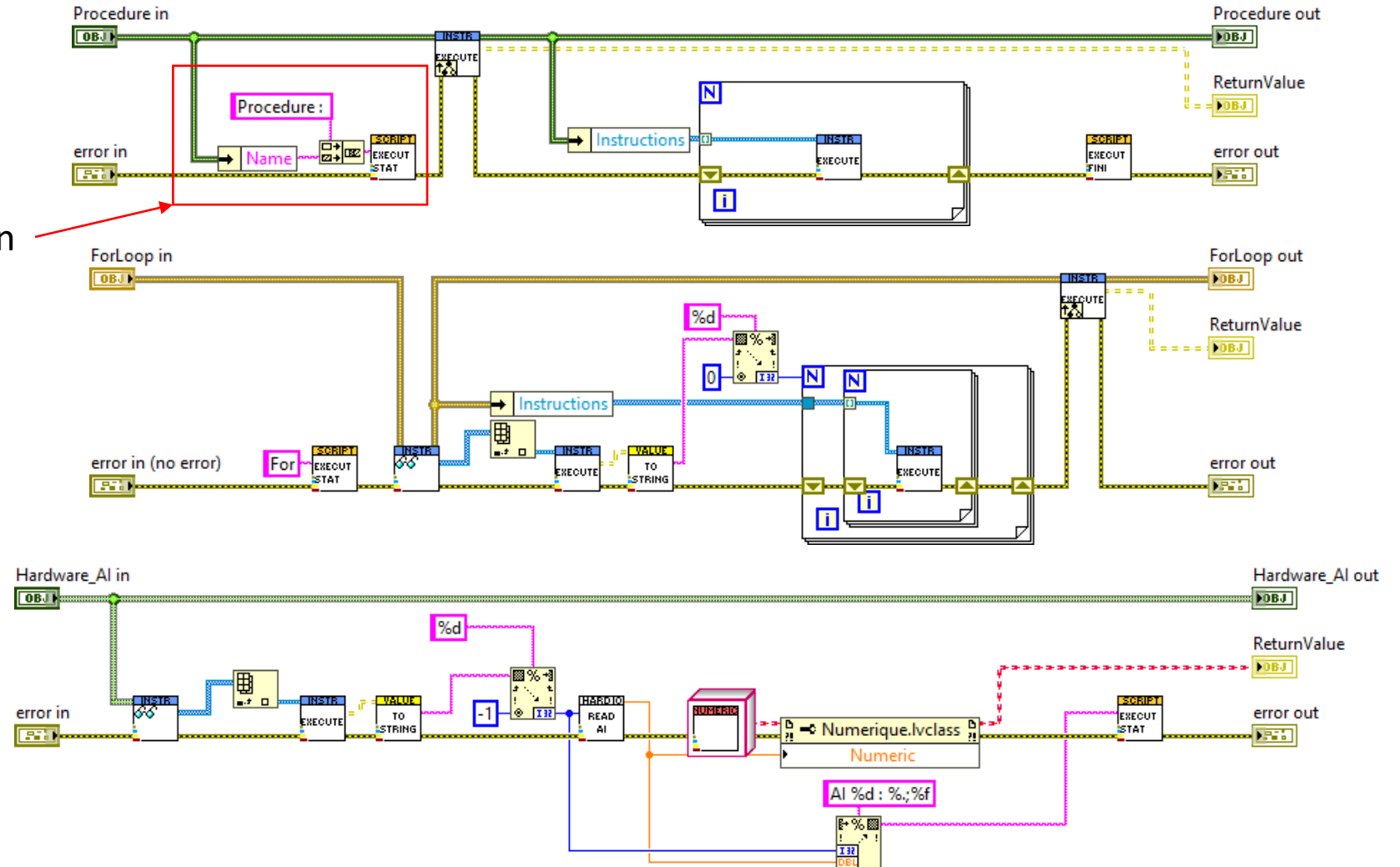
Dispatch dynamic :

Procedure.lvclass:Execute.vi

Trace d'exécution

ForLoop.lvclass:Execute.vi

Hardware_AI.lvclass:Execute.vi

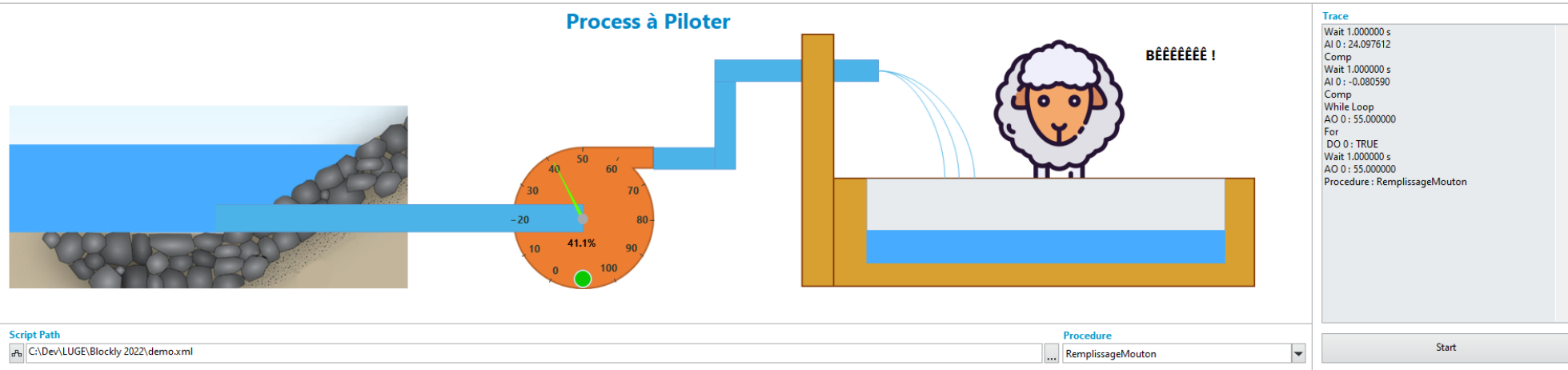


Pour aller plus loin

- Communication entre l'éditeur Blockly et LabVIEW
 - Animation d'exécution
 - Chargement automatique du fichier XML dans Blockly
- Gestion d'exécution :
 - Debugger
 - Breakpoints
 - Variables

Utilisation de blockly avec une application LabVIEW

Process à Piloter



Script Path: C:\Dev\LUGE\Blockly 2022\demo.xml

Procedure: RemplissageMouton

Start

Trace

```
Wait 1.000000 s
AI 0 : 24.097612
Comp
Wait 1.000000 s
AI 0 : -0.080590
Comp
While Loop
AO 0 : 55.000000
For
DO 0 : TRUE
Wait 1.000000 s
AO 0 : 55.000000
Procedure : RemplissageMouton
```

Démonstration

#DQMH #Blockly #LabVIEW #Mouton

Blockly dans un logiciel Industriel



Nerys Group companies

The screenshot displays the Vasco software interface, which integrates Scratch-style Blockly for industrial automation. The main window shows a 3D model of a mechanical test setup with various parameters labeled: Force_2 (-2.2 daN), Force_1 (56.6 daN), Distance_1 (4.026 mm), Vitesse_2 (1 tr/mn), and Vitesse_1 (1912 tr/mn). A central panel displays a Blockly script for controlling speed, including blocks for setting initial and final speeds, a loop for a specific number of cycles, and ramps for speed changes. The interface also features several numerical indicators (VuMetre) for parameters like Pilotage_Vitesse (1801 tr/mn), Tension_10V (7.63 V), Force_1 (88.8 daN), and Force_2 (-0.7 daN). A graph at the bottom shows the speed profile over time, with a scale of 120 s. The software is running on a Windows system, as indicated by the taskbar and system tray.

#Vasco #NERYS #PlacementDeProduit



———— Nerys Group companies ————

A detailed photograph of industrial machinery, likely a robotic assembly line or a precision manufacturing station. The machine is primarily black and blue, with various mechanical components, cables, and sensors visible. A central cylindrical component is prominent, surrounded by various tools and fixtures. The background is a plain, light-colored wall.

Merci à tous pour votre attention

www.nerysgroup.com

Annexe : Blockly Factory

- <https://blockly-demo.appspot.com/static/demos/blockfactory/index.html>

The image shows the Blockly Factory interface. On the left, a block definition for 'block_type' is shown. It has a name 'block_type' and several inputs: a 'dummy input', a 'value input' of type 'Boolean', and another 'value input' of type 'String'. Each value input has a 'fields' dropdown set to 'left' and a text field containing 'Ma 1ère entrée' and 'Ma 2nd entrée' respectively. Below the inputs, there are properties for 'automatic', 'inputs', 'top+bottom connections', 'tooltip', 'help url', 'top type', 'bottom type', and 'colour' (set to 'hue: 135°').

On the right, the 'Block Definition' and 'Generator stub' are shown in JavaScript. The 'Block Definition' is:

```
Blockly.Blocks['block_type'] = {
  init: function() {
    this.appendDummyInput()
      .appendField("Mon super block !");
    this.appendValueInput("INPUT1")
      .setCheck("Boolean")
      .appendField("Ma 1ère entrée");
    this.appendValueInput("INPUT1")
      .setCheck("String")
      .appendField("Ma 2nd entrée");
  }
};
```

The 'Generator stub' is:

```
Blockly.JavaScript['block_type'] = function(block) {
  var value_input1 = Blockly.JavaScript.valueToCode(block, 'INPUT1', Blockly.JavaScript.ORDER_ATOMIC);
  var value_input1 = Blockly.JavaScript.valueToCode(block, 'INPUT1', Blockly.JavaScript.ORDER_ATOMIC);
  // TODO: Assemble JavaScript into code variable.
  var code = '...;\n';
  return code;
};
```

Annexe : descripteur de block

.\Blockly\Specifique\addons-blockly\Blocks\LUGE.js

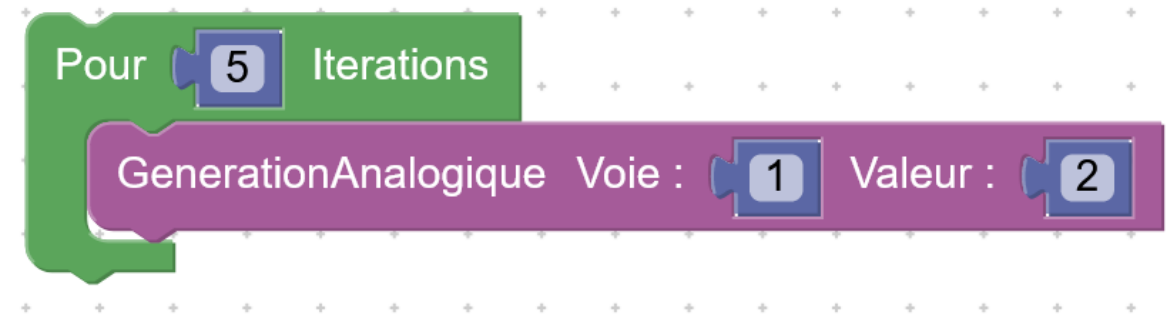
```
Blockly.Blocks['for_loop'] = {  
  init: function() {  
    this.loop_index = new Blockly.FieldTextInput("");  
    this.appendDummyInput()  
      .appendField("Pour");  
    this.appendValueInput("ITER");  
    this.appendDummyInput()  
      .appendField("Iterations");  
    this.appendStatementInput("CODE")  
      .setCheck(null);  
    this.setInputsInline(true);  
    this.setPreviousStatement(true, null);  
    this.setNextStatement(true, null);  
    this.setColour("%{BKY_LOOPS_HUE}");  
    this.setTooltip("");  
    this.setHelpUrl("");  
    this.loop_index.showEditor_=(()=>{  
  
      });  
  }  
};
```



Annexe : générateur de code

.\Blockly\Specifique\addons-blockly\Generators\Luge\LUGE.js

```
Blockly.Luge['for_loop'] = function(block) {  
  
  var value_iter = Blockly.Luge.valueToCode(block, 'ITER', Blockly.Luge.ORDER_ATOMIC);  
  var statements_code = Blockly.Luge.statementToCode(block, 'CODE');  
  
  var code = 'Pour ( ' + value_iter + ' )\n'+  
  statements_code +  
  'FinPour\n';  
  return code;  
};  
  
Blockly.Luge['generation_ana'] = function(block) {  
  var value_channel = Blockly.Luge.valueToCode(block, 'CHANNEL',  
  Blockly.Luge.ORDER_ATOMIC);  
  var value_value = Blockly.Luge.valueToCode(block, 'VALUE', Blockly.Luge.ORDER_ATOMIC);  
  var code = 'GenerationAnalogique ( ' + value_channel + ' , ' + value_value + ' )\n';  
  return code;  
};
```



```
Pour ( 5 )  
  GenerationAnalogique ( 1 , 2 )  
FinPour
```