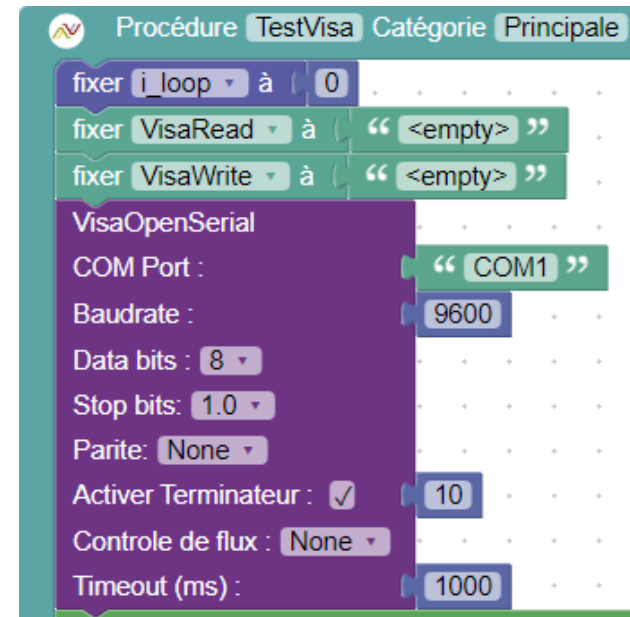
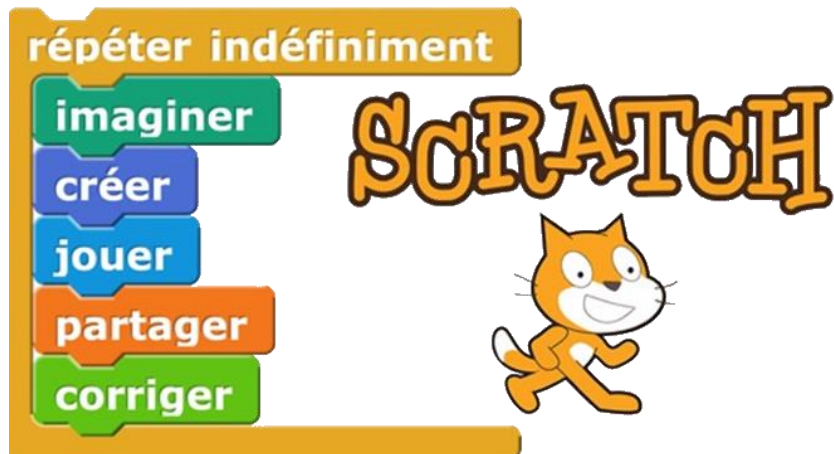




- Blockly -

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



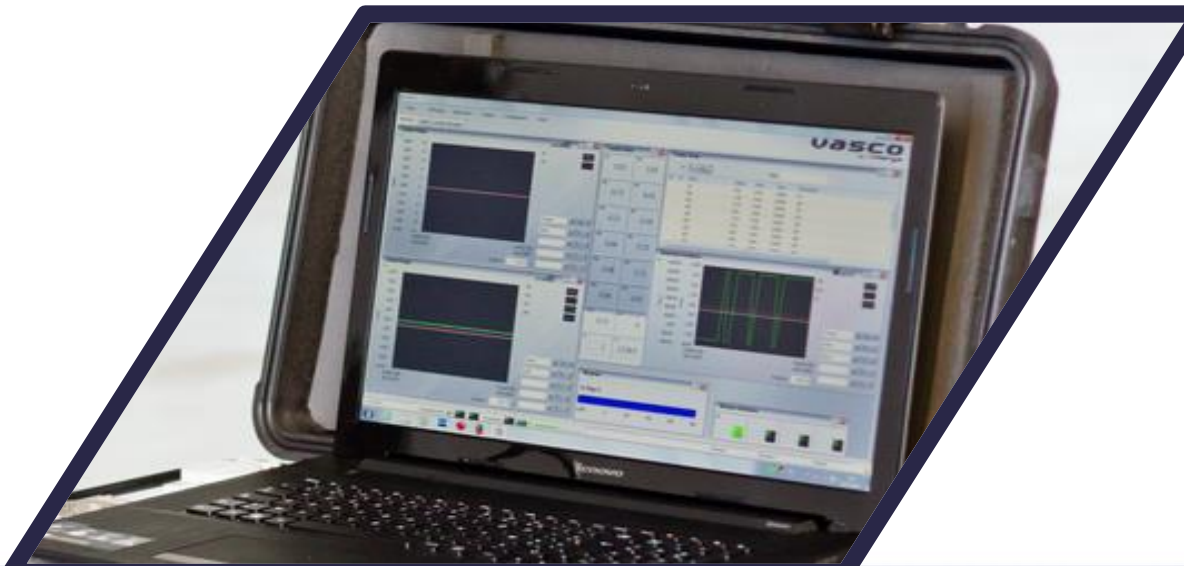
NERYS en bref



Nerys Group companies

- 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 pilotage de moyens d'essais appelée "Vasco" pouvant être déployée sur cible temps-réel (NI RT).

vasco
by  **Nerys**

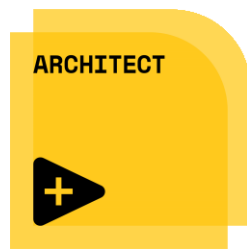
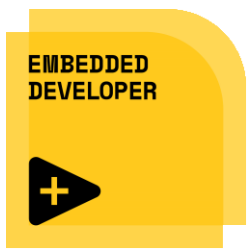


NERYS GROUP en bref



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

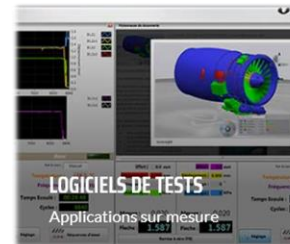


Plus d'infos : nous contacter

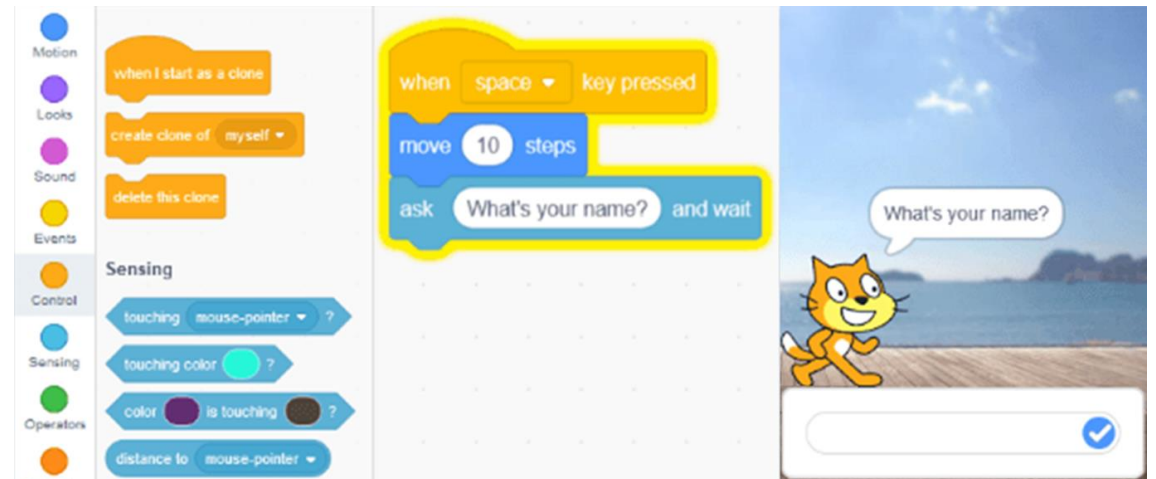
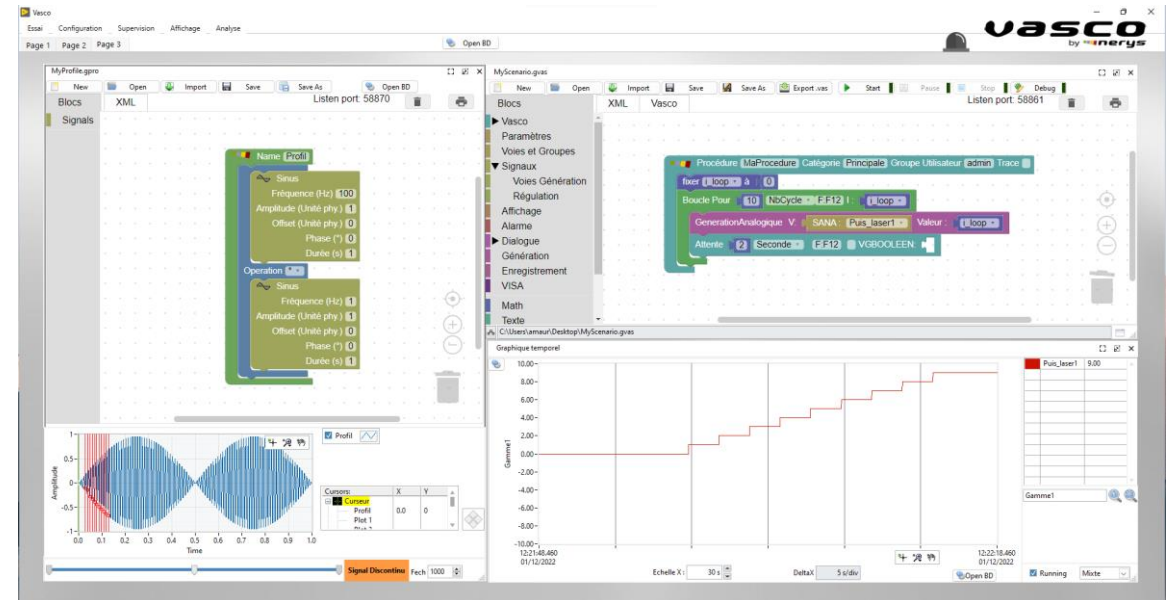


Nerys Group companies

- 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

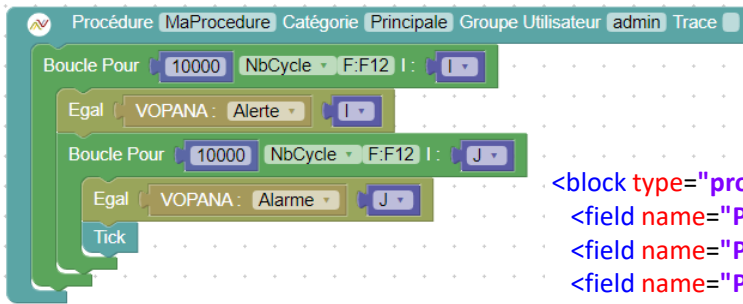
Générateurs code

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
```

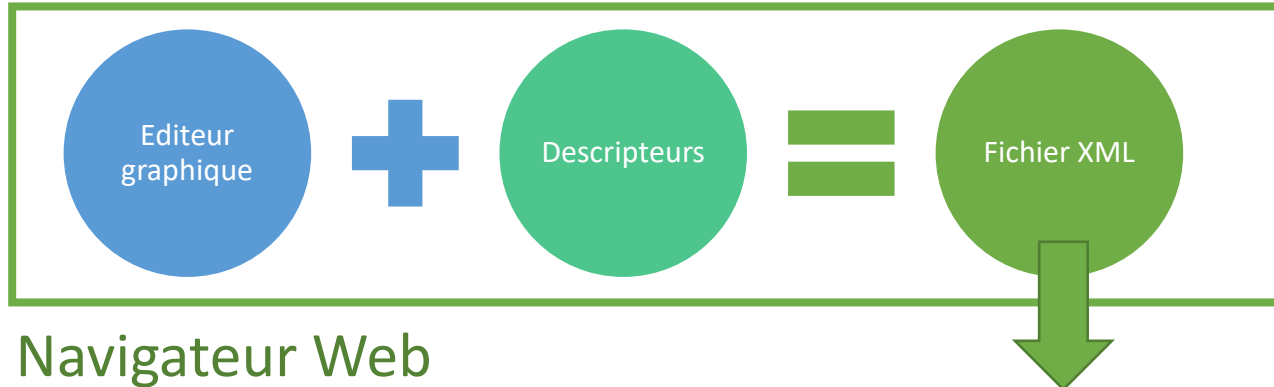


```
<block type="procedure">  
  <field name="PROC_NAME">MaProcedure</field>  
  <field name="PROC_CAT">Principale</field>  
  <field name="PROC_GROUP">admin</field>  
  <field name="PROC_TRACE">FALSE</field>  
  <statement name="PROC">  
    <block type="bky_for_loop" id="Et:[fwd6Ato!|9c8x:%M]">  
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

Exemple de descripteur XML

Intégration avec LabVIEW

Principe : Utilisation du fichier XML comme source interprétée par LabVIEW

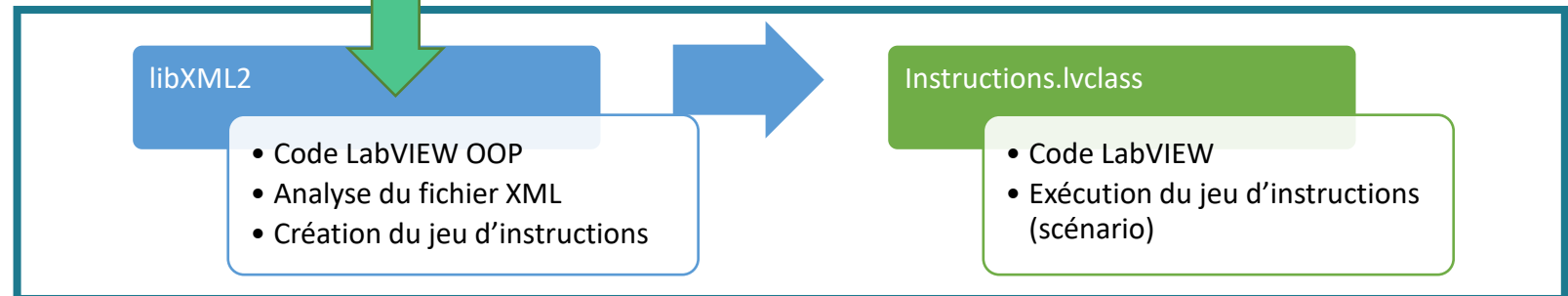


Structure XML :

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

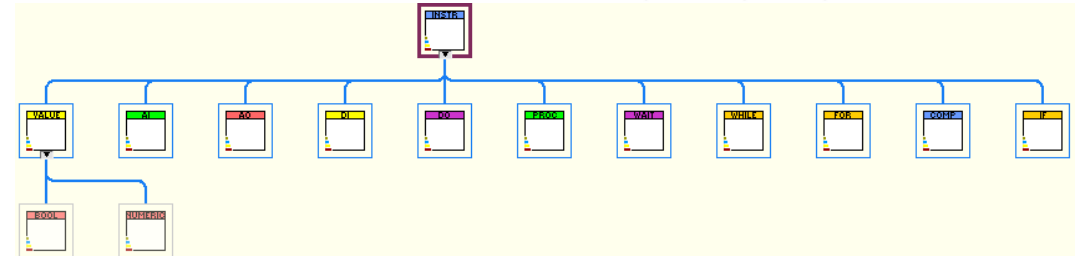
```
<block type="procedure" id="pz*o73e6+Am^zYpJvpZ2" x="38" y="38">
  <field name="PROC_NAME">MaProcedure</field>
  <field name="PROC_CAT">Principale</field>
  <field name="PROC_GROUP">admin</field>
  <field name="PROC_TRACE">FALSE</field>
  <statement name="PROC">
    <block type="bky_for_loop" id="Et:[fwd6Ato!|9c8x:%M">
      <field name="UNIT">NbCycle</field>
```

LabVIEW



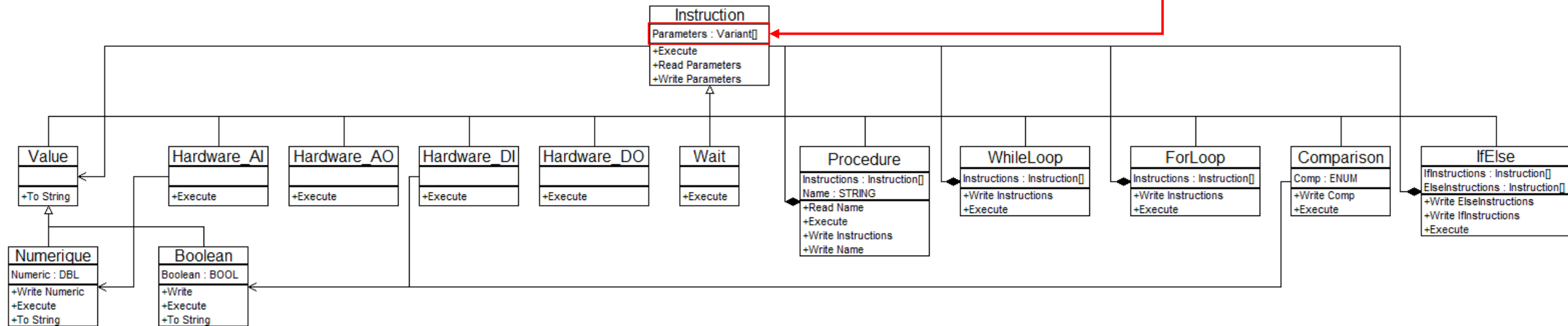
LabVIEW : Gestion des instructions

- Architecture Objets
 - Dispatch dynamique
- Exécution récursive
 - Une seule primitive : Execute.vi



NB : Parameters : Instruction[]

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

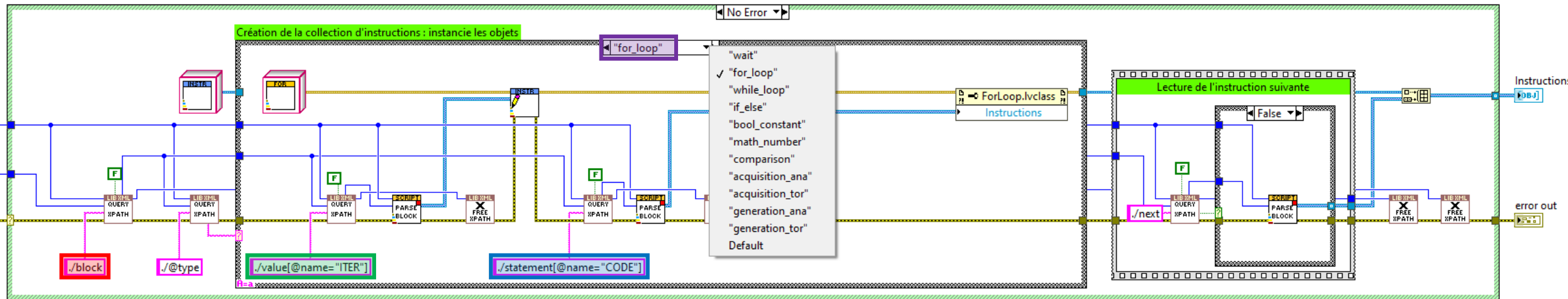


LabVIEW : Analyse du XML

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

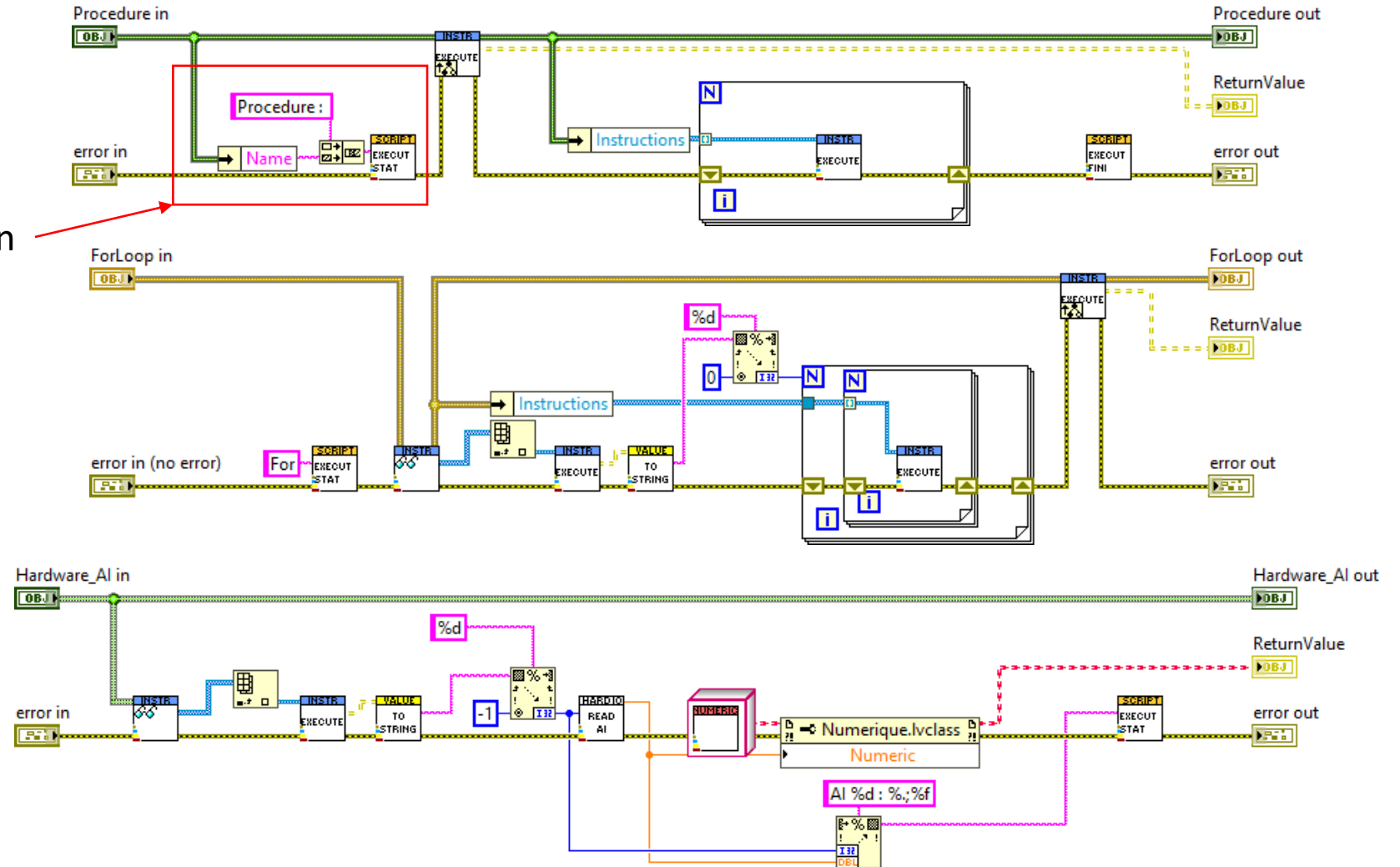
Dispatch dynamic :

Procedure.lvclass:Execute.vi

Trace d'exécution

ForLoop.lvclass:Execute.vi

Hardware_AI.lvclass:Execute.vi



Pour aller plus loin



————— Nerys Group companies —————

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



Nerys Group companies



Utilisation de blockly avec une application LabVIEW

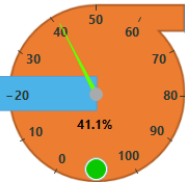
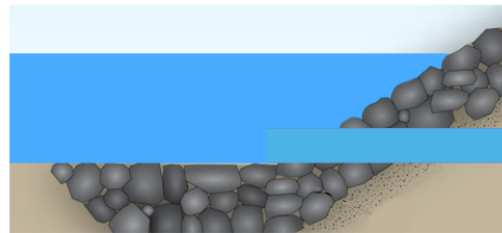
LabVIEW User Group



Francophone



Process à Piloter



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
```

Script Path

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

Procedure

RemplissageMouton

Start

Démonstration

#DQMH #Blockly #LabVIEW #Mouton

Vasco by nerys

Essai Configuration Supervision Affichage Analyse

Page 1 Page 2 Page 3

Open BD

MyProfile.gpro

Blocs XML Listen port: 58870

Signaux

Name Profil

Sinus

Fréquence (Hz) 100

Amplitude (Unité phy) 1

Offset (Unité phy) 0

Phase (°) 0

Durée (s) 1

Operation

Sinus

Fréquence (Hz) 1

Amplitude (Unité phy) 1

Offset (Unité phy) 0

Phase (°) 0

Durée (s) 1

Graphique temporel

Gamme1

Puis_laser1 9.00

Gamme1

Signal Discontinu Fch 1000

Running Mixte

Démonstration de Blockly dans un logiciel Industriel
#Vasco #NERYS #PlacementDeProduit



— Nerys Group companies —

The background of the slide is a photograph of a complex industrial machine, likely a robotic assembly line. The machine is primarily black and blue, with various mechanical components, cables, and sensors visible. A central cylindrical component is prominent. The lighting is focused on the machine, creating highlights and shadows. The image is partially obscured by a white diagonal shape on the left and a dark blue diagonal shape on the right.

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 vertical sidebar lists properties: Input, Field, Type, and Colour. The main workspace displays a block definition for 'block_type'. The visual representation of the block is a green rectangle containing three input fields: a 'dummy input' with the text 'Mon super block !', a 'value input' labeled 'INPUT1' with the text 'Ma 1ère entrée' and type 'Boolean', and another 'value input' labeled 'INPUT1' with the text 'Ma 2nd entrée' and type 'String'. Below the visual representation, there are controls for 'automatic inputs', 'top+bottom connections', 'tooltip', 'help url', 'top type', 'bottom type', and 'colour' (set to 135°). On the right, the 'Block Definition' and 'Generator stub' are shown in JavaScript code. The 'Block Definition' code defines the 'init' function for the block, which appends the dummy input and the two value inputs. The 'Generator stub' code defines the 'generateCode' function for the block, which uses 'valueToCode' to generate code for the value inputs.

```
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");
  }
};

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

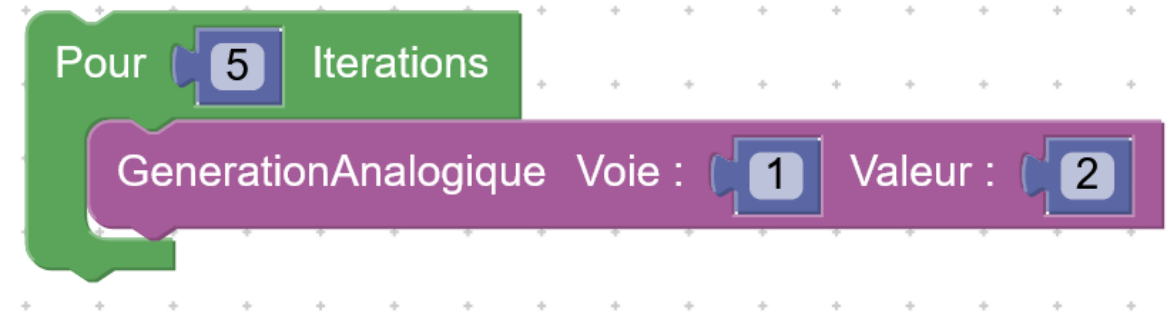
.\Blockly\Specifique\addons-blockly\Blocks\LUGE.js

```
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_=(()=>{  
  
      });  
  }  
};
```



.\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
```