Ignition HIL simulation implementation guide

Ferdie Reijnen, Jeroen Verbakel, Joanna van de Mortel-Fronczak, and Jacobus Rooda

May 2, 2019

This document describes the process of generating and implementing a HIL model in the Ignition SCADA software. Section 1 describes the process of generating the compiled java code for the HIL simulation. Section 2 describes setting up Ignition. Section 3 described how the HIL simulation can be started. Generation of PLC code is not described in this document. information related to PLC code generation can be found here: `http://cif.se.wtb.tue.nl/tools/cif2plc/`. The files mentioned in this document can be downloaded from: `https://github.com/ffhreijnen/PrinsesMarijkeComplex`.

1 Generating the Java model

For generating the Java code, we use CIF3, which is available for download here: `http://cif.se.wtb.tue.nl/download.html`.

For generation of the Java file, the *X4_Generate_java.tooldef2* is used. It requires a *Sim.cif* file and a *HardwareMapSim.cif* file as input. These files can be found under *Simulation/Sim.cif* and *HIL_code_generation/HardwareMapSim.cif*, respectively.

The tooldef2 file generates three files, *c2j.cif*, *c2j.java*, and *c2j.py*. *c2j.cif* is an intermediate cif file that is used for code generation, *c2j.java* is the generated code, and *c2j.py* is used for communication between the PLC and the java model.

To compile the java code, *X6_Compile_Java.tooldef2* is used. For this, a Java compiler is necessary, the location of the *javac.exe* has to be set in the *Config.tooldef2*. As input, the generated *c2j.java* is used. The tooldef generates a *myPackage* folder with the compiled code.

2 Configuring Ignition

Configuring ignition consists of two steps. Setting up the tags for PLC communication and implementing the java code. A trial version of the Ignition software can be downloaded here: `https://inductiveautomation.com/downloads/ignition`.

2.1 PLC tags

For Ignition to communicate with the PLC, so called tags have to be configured. More information about tags is available here: `https://docs.inductiveautomation.com/display/DOC79/Tags`. The following steps are performed in the Ignition Designer.

In the tag browser, one tag is created for enabling and stopping the simulation. Right click *Tags*, *New Tag*, and *OPC Tag*. Name it "SimulationRunning", with data type Boolean.

Secondly, for every PLC input and output a tag has to be created (This can also be generated during code generation, but here we assume that this generation is not available). The tags to create are the tags referenced in the *c2j.py* file, as generated by CIF. The name is the name as in this file, the type is as defined in CIF. For PLC connection the OPC server should be chosen correctly. More information about creating tags is available here: `https://docs.inductiveautomation.com/display/DOC79/Browsing+and+Creating+OPC+Tags`, specific information for Siemens is available here: `https://docs.inductiveautomation.com/display/DOC79/Siemens#Siemens-ConfiguringSiemensAddressi`

2.2 Java code implementation

In the *Project Browser*, select *Scripts*, then *Script Library [project]*, then *New Script*, and name it *myFuns*. Here we define function, used for the communication between the generated Java file and the sensor and actuator tags.

Listing 1: *MyFuns* python file.

```
from system import tag

def valueOf(tagName):
"""Function that returns the value of a PLC tag."""
Actuator = tag.read(tagName)
if not Actuator.quality.isGood():
  print("ERROR: Quality of " + tagName + " is bad.")
  return Actuator.value

def writeTo(tagName, newValue):
"""Function that write a value to a PLC tag."""
if tag.exists(tagName):
  curValue = readTag(tagName)
  if curValue != newValue:
    tag.write(tagName, newValue)
  else:
    print("ERROR: Tag " + tagName + " Does not exist")
```

Secondly, we define a script for the communication between the Java file and the sensor and actuator tags, named *c2jInputs*. The contents of this script are automatically generated during code generation. This is the generated *c2j.py* code. Copy the content of *c2j.py* to *c2jInputs*.

Thirdly, we define a script that runs the model, named *CifHandler*. The java model file is named *c2j*.

Listing 2: *CifHandler* python file.

```
## imports
import sys
import threading
from project.myFuns import *
import project.c2jInputs as myIO

# Enter the location of the HIL java folder (myPackage) below
sys.path.append(" ")
```

```
from myPackage import c2j

def runModel():
  """Function that executes the Java model."""
  model = CifHandler()
  print "async started"
  try:
    model.execInfinite(30)
  except SystemExit:
    print "async finished"

## Java model handler class
class CifHandler(c2j):

def updateInputs(self):
  """This function is called from the Java model before
  the model is executed to update the inputs."""
  myIO.readTags(self)

def preExec(self):
  """This function is called from the Java model before
  the model is executed. here, we check if the simulation
  termination is requested."""
  runValue = system.tag.read("SimulationRunning")
  if not runValue.value or not runValue.quality.isGood():
    raise SystemExit('Thread terminated')

def postExec(self):
  """This function is called from the Java model after
  one model cycle is executed. Here, we write all the
  simulation sensor values to the PLC tags."""
  myIO.writeTags(self)
```

To start the simulation a button is created as follows. In the *project browser*, click *Windows* and then *Main Window*. In the newly created window, create a buttons (click *components* and select *Button*). Double click the button, under *Event Handlers*, *action*, *actionPerformed*, select *Script Editor*. This button is used to start the simulation as follows.

Listing 3: Script of button 1.

```
system.util.invokeAsynchronous(project.CifHandler.runModel)
```

3 Starting the HIL simulation

The following steps have to be performed to start the HIL simulation.

1. Download the PLC program to the PLC.

2. Start the PLC in run mode.
3. Establish a connection between Ignition and the PLC via the Ignition Control Panel.
4. Verify whether the Ignition tags are able to obtain the data from the PLC.
5. Set the *SimulationRunning* Boolean to True.
6. Click the start simulation button.
7. To end te simulation, set the SimulationRunning Boolean to False.