# SensorStudio User Documentation



**InvenSense Inc.**
1745 Technology Drive, San Jose, CA 95110 U.S.A
+1(408) 988–7339
www.invensense.com

Document Number: AN-000078
Revision: 1.3
Date: 10/17/2016

# TABLE OF CONTENTS

## TABLE OF FIGURES

## ADDITIONAL USEFUL LINKS

**INVENSENSE WEBSITE:**
http://www.InvenSense.com/
**ARDUINO WEBSITE:**
https://www.arduino.cc/en/Guide/Environment
https://www.arduino.cc/en/Main/ArduinoBoardZero
https://www.arduino.cc/en/uploads/Main/Arduino-Zero-schematic.pdf
**J-LINK DEBUGGER WEBSITE:**
https://www.segger.com/jlink_base.html
https://www.segger.com/jlink-adapters-9pin-cortexm.html

## ADDITIONAL DOCUMENTS

ICM-30630 Datasheet
ICM-30630 System Hardware Design Guide
ICM-30630 Application Note: Using Eclipse IDE with J-Link Debugger
ICM-30630 eMD Software User Guide
ICM-30630 Command Protocol and Architecture
ICM-30630 Shield Hardware Guide

# 1 DOCUMENT INFORMATION

## 1.1. PURPOSE OF THE DOCUMENT

The purpose of the document is to get you started with the important concepts needed to work with InvenSense SensorStudio. The document consists of a Quick Start Guide (see 3) and a Reference Manual (see 4) sections.

## 1.2. REFERENCES

| ID | NAME | LINK |
|---|---|---|
| [2] | ICM-30670 Datasheet | http://www.invensense.com/products/icm-30670/ |
| [3] | InvenSense Developers Corner | http://www.invensense.com/developers/software-downloads |
| [4] | Arduino Zero board | https://www.arduino.cc/en/Main/ArduinoBoardZero |
| [5] | J-Link Debug Probes | https://www.segger.com/jlink-debug-probes.html |
| [6] | Eclipse IDE for C/C++ Developers | https://eclipse.org/downloads/ |
| [7] | Arduino IDE | https://www.arduino.cc/en/Main/Software |

**Table 1**

## 1.3. TERMINOLOGY

**FireFly Device**
InvenSense SoC Device, e.g. ICM-30670

**GenericSensorHub Device**
ST-Nucleo and ICM-20690-based device

**EMD**
Acronym for Embedded Software Packages

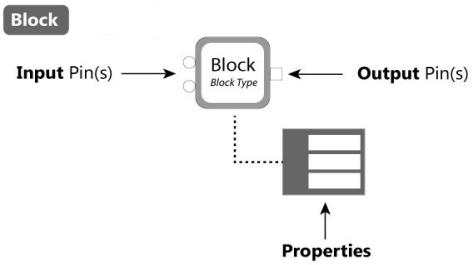
**SDK**
Software package containing items for development of new elements to be embedded on a SoC (FireFly or GenericSensorHub).

**Custom Sensor Designer**
Design the sensor code that you will be able to load on the MCU, using outputs of sensors integrated in the Sensor Framework. SensorStudio can also run this code on the desktop CPU.

**Auxiliary Sensor Designer**
Design the sensor driver code you will be able to load on MCU. SensorStudio cannot run this code on the desktop CPU.

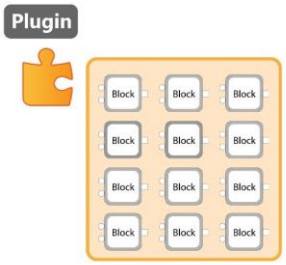
**Custom Sensor**
Custom Sensor is the user-defined code that has been loaded and running on the MCU.

**Block**
A block is a piece of software, processing inputs and generating outputs.
A block
- has one or multiple input pins
- exports one or multiple output pins
- exposes properties

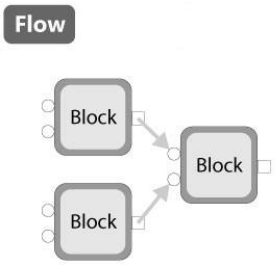


**Plugin**
It contains multiple blocks for a common usage. For instance, the IOFile plugin contains all blocks related to IO file operations like reading a file, writing a line, etc.

InvenSense SensorStudio comes with a wide range of plugins to access device sensors, perform math operations, signal processing, etc.




**Flow**
A Flow is composed of different blocks from different plugins connected to each other




## 1.4. SYSTEM REQUIREMENTS

In order for SensorStudio to operate properly, your computer needs meet or exceeed the following system requirements:

**Operating Systems:** Windows 10, Windows 8.1, Windows 8, Windows 7 (32bits or 64bits)

**Processors:** Intel i5 or equivalent

**Disk space:** 400 MB for Sensor Studio only

**RAM:** 2 GB

**Graphics:** No specific graphics card is required.

# 2 SCOPE

## 2.1. OVERVIEW

InvenSense SensorStudio is the software companion to InvenSense Sensor chipsets, designed to simplify the work involved in adding sensors and data fusion to your products.

With InvenSense SensorStudio connected to an evaluation board, you will be able to select and evaluate the Sensor embedded features you are interested in (orientation, activity tracking, gestures, etc). You will also be able to design and validate your own embedded features.

InvenSense SensorStudio simplifies configuring and extending Invensense SensorHub Framework. InvenSense already supports many partners' sensors, yet if your project requires an unsupported sensor, a complete API is provided to support the SensorHub Framework extension.

For a hands-on learning experience, please refer to the Quick Start Guide (see 3) section. The Reference Manual (see 4) sections is a summary of the SensorStudio capabilities. Also, the Help (see 4.7) is always there to give you detailed and contextual information.

# 3 QUICK START GUIDE

## 3.1. OVERVIEW

InvenSense SensorStudio is the software companion for the InvenSense Sensors chipsets, helping to support programming the SensorHub MPU and visualizing its outputs.
This section will guide you step-by-step through your first use of InvenSense SensorStudio.
You will learn how to:

- Visualize sensors outputs with InvenSense SensorStudio
- Program the SensorHub MPU and visualize outputs of your own algorithms

## 3.2. INSTALLING SENSORSTUDIO

You will need to create an account on InvenSense Developers Corner to download SensorStudio from the Software Downloads section.

- Once the download is complete, start the SensorStudio installer by double clicking on the **SensorStudio-win32-2.2.0.setup.exe** file.

- Follow the Setup wizard to install the application.
  You may choose to check the "uninstall previous version" if available, however this is not required.

- After the installation of SensorStudio application files, Setup Wizard will launch the installation of **Arduino driver**, **ST-Link driver**, **FTDI driver** & **Microsoft Visual Redistribuable** components.

Once the installation is complete, you can automatically launch SensorStudio.

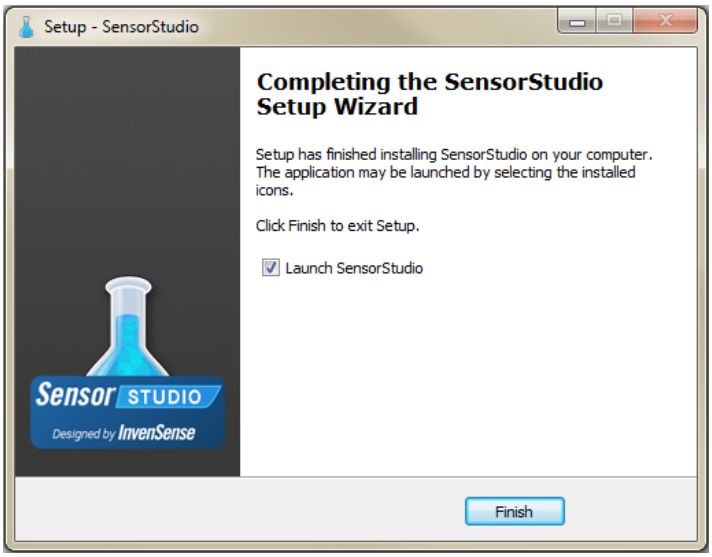



**Figure 1. Screenshot of InvenSense SensorStudio Setup Wizard complete**

## 3.3. CONNECTING THE EVALUATION BOARD

Refer to *Hardware quick start guide* document available from the Help menu for details on to setup your kit to be used with SensorStudio.

### 3.3.1 ICM-30670

InvenSense SensorStudio natively supports the connection with the ICM-30670 Arduino shield using an Arduino Zero.

Below is a picture of the hardware configuration we will use in this quick start guide:

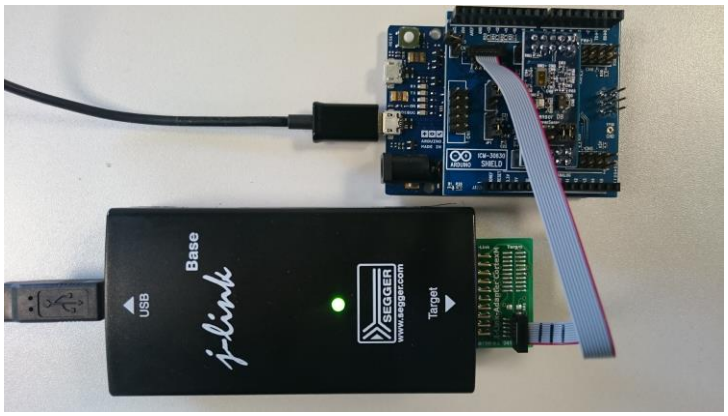



**Figure 2. Picture of ICM-30670 Development Kit and J-Link probe**

Details about the ICM-30670 Arduino shield configuration are provided in the "*ICM-30670 - eMD Software Guide*" and "*ICM-30670 - Using Eclipse IDE with J-Link*"

For this document, we are assuming all the prerequisite steps have been performed:

- The Arduino Zero, the InvenSense ICM-30670 Shield, and InvenSense Sensor DaughterBoard are connected to the PC through a USB port
- The J-Link Base debugger is connected to InvenSense ICM-30670 Shield using the debug ribbon cable
- The J-Link Base debugger is connected to the PC through a USB port

### 3.3.2 Generic Sensor Hub

InvenSense SensorStudio natively supports the connection with the GenericSensorHub reference kit, which includes an ICM-20690 sensor board, a Nucleo Carrier board and a STM32F411 Nucleo board.

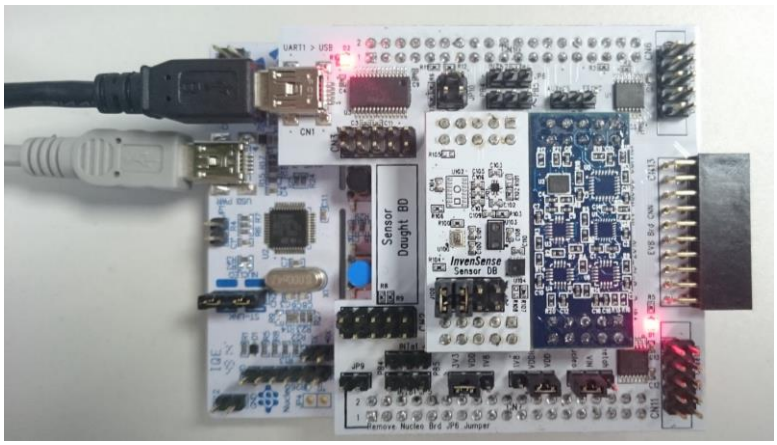



**Figure 3. Picture of Generic Sensor Hub development kit**

## 3.4. EVALUATING YOUR KIT

InvenSense SensorStudio installer may have created a shortcut on your Windows desktop and/or Windows start menu and/or Windows Toolbar.

Start InvenSense SensorStudio by double clicking on the  shortcut. If you elected not to add shortcuts during the install process, you can still locate an InvenSense SensorStudio shortcut where you installed the software (e.g. default install path is *C:\InvenSense\SensorStudio\2.2.0*).

At startup time, InvenSense SensorStudio will bring up the "*Welcome screen*".

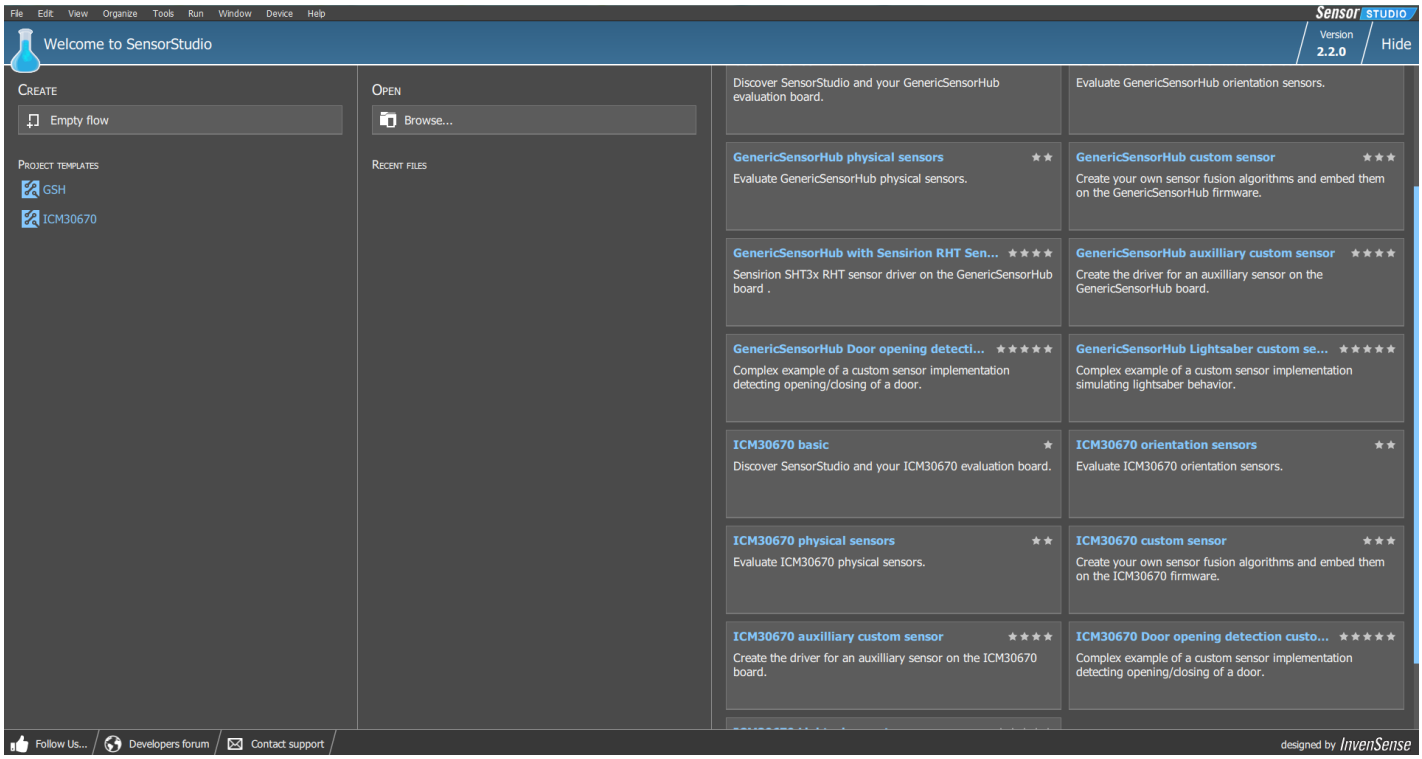



**Figure 4. Screenshot of Welcome Screen**

SensorStudio is checking if a new version is available for download from our Developers Corner. You will notice an "Update available" on SensorStudio Welcome screen when a new version exists. Clicking on this link will let you see the changelog and launch the new installer.

Please select the orientation sensors sample from the Welcome screen ("GSH orientation sensors" or "ICM30670 orientation sensors" depending on the device you want to use). This will open up InvenSense SensorStudio main window.

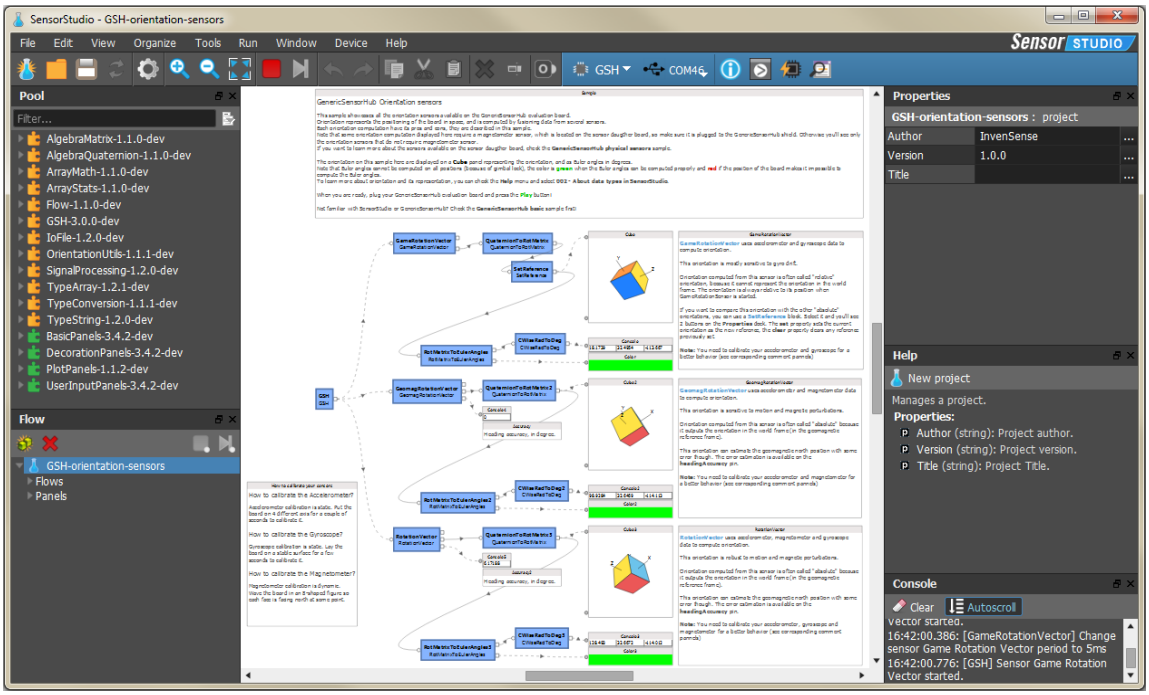
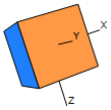



Figure 5. Screenshot of GSH Orientation sample

The orientation sensors sample can be started using the "Start" button on the SensorStudio toolbar  . When the sample is started, you will notice that the button has changed to a stop icon  .




You will see the various cubes rotate in space as you rotate your evaluation board.

Note that pressing the "Start" button will update the "Connected" property of your device block (GSH or ICM30670), will load and start the Sensor Framework in the evaluation board and start the SensorStudio flow. This allows the SensorStudio runtime to run on the desktop and process the outputs from the evaluation board. For more information on this, please refer to 3.7 Sensor Framework explained.

You can also "single click" on the device block (GSH or ICM30670) and observe that the "Connected" property is ticked:

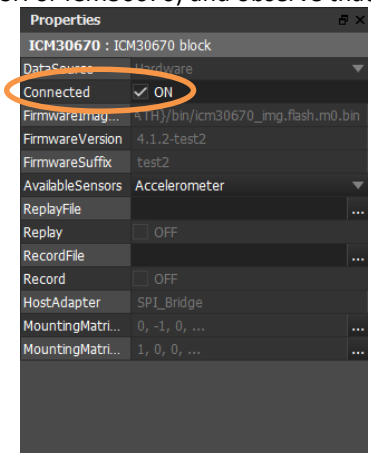



**Figure 6. Screenshot of block device properties**

You will also notice that the Help dock is now showing the contextual help related to the selected block.

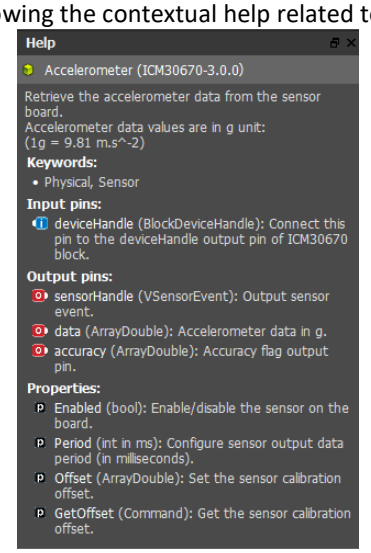



**Figure 7. Screenshot the Accelerometer sensor help dock**

You can “single click” on each block available in the flow to review the Help dock content and get more familiar with these elements.

You can find the list of features supported by your FireFly ICM-306XX SoC or GenericSensorHub through the “Available Sensors” property list:

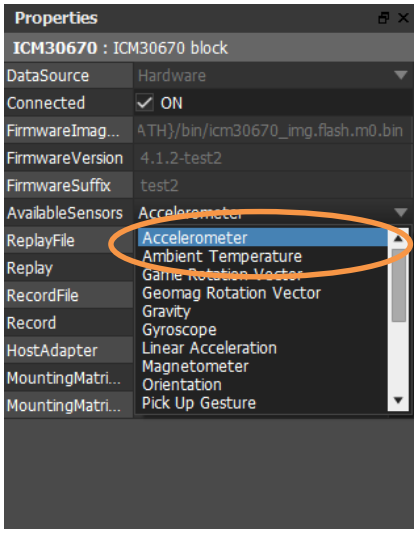



**Figure 8. Screenshot of ICM-30670 properties (AvailableSensors)**

You can always bring up the Welcome Screen again from:

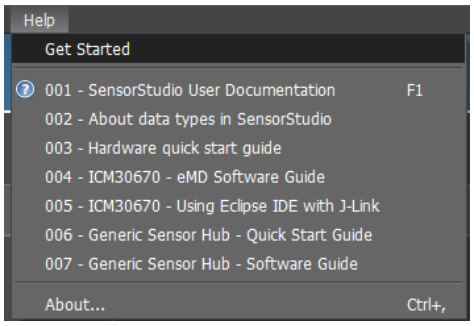



**Figure 9. Screenshot of Help menu/Welcome Item**

## 3.5. CREATE A FLOW

In this section, you will learn how to create your own flow and visualize the outputs from InvenSense sensor framework running on InvenSense SensorHub.
To start fresh in InvenSense SensorStudio, click on the "File" menu and select your device from the "New" submenu (GSH or ICM30670) to create a new flow in SensorStudio.

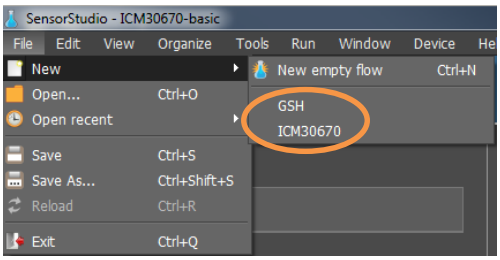



**Figure 10. Screenshot of new flow in File menu**

You will have a flow with only one block created, which corresponds to the device you selected (GSH or ICM30670).

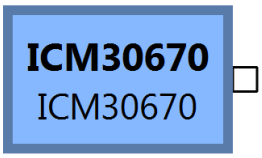



**Figure 11. Screenshot of ICM-30670 in the flow area**

Now let's add some sensors. Enter "accelerometer" in the "Pool" dock search field and notice the Accelerometer block on the list, as shown below.

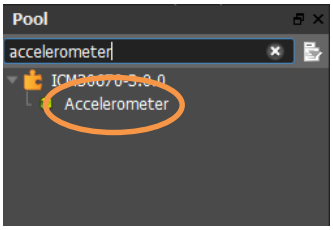



**Figure 12. Screenshot of search for Accelerometer block in pool dock**

For more information about the Pool dock, please refer to "4.9 Pool " section.
Click and hold on Accelerometer to drag and drop the Accelerometer block on the flow area

Repeat the "Click and hold" on to drag and drop the Gyroscope and GameRotationVector blocks on the flow area, to obtain the following

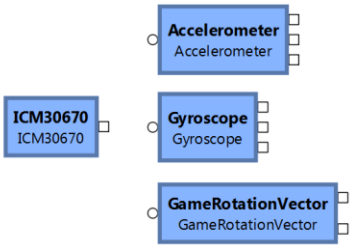



**Figure 13. Screenshot of Flow with ICM-30670, Accelerometer, Gyroscope & GameRotationVector blocks**

Click on the device block (GSH or ICM30670) output  , hold and move your mouse to the input of the Accelerometer block (until the mouse cursor becomes a "+" sign), to create a connection between the blocks. Repeat this operation between the GSH/ICM30670 & Gyroscope, and GSH/ICM30670 & GameRotationVector. You will obtain the following:

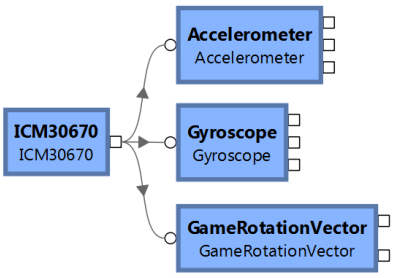



**Figure 14. Screenshot of flow connections between ICM-30670 device and its Sensor Framework features**

Enter "sliding" in the "Pool" dock search field and notice the Sliding panel from the list

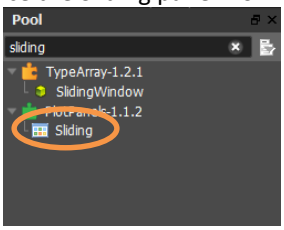



**Figure 15. Screenshot of Sliding panel in pool dock**

Click on the "Sliding" list item, hold, drag and drop two "Sliding" panels on the flow. Create connections between the Accelerometer 'data' output and the first Sliding panel, and between the Gyroscope 'data' output and second Sliding panel, as follow:

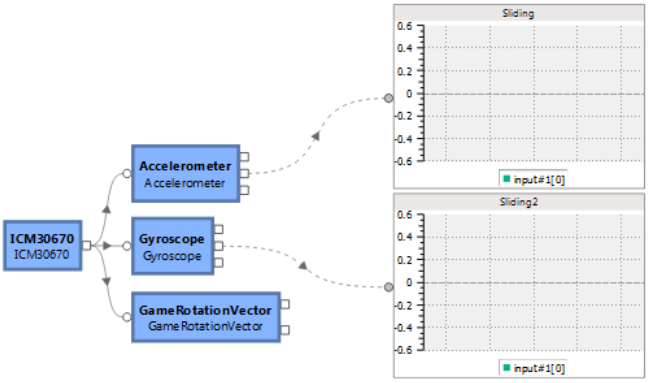
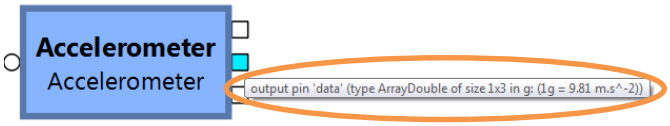


**Figure 16. Screenshot of Flow with Sliding panels for Accelerometer & Gyroscope**

You may have noticed when your mouse hovers the block outputs or inputs, there is a tooltip with the pin brief description




Now, click on the "Start" button on the SensorStudio toolbar  . The button should now be changed to a "Stop" icon  and the "Sliding" panels on the flow are now receiving outputs from the sensor framework running on the evaluation board.

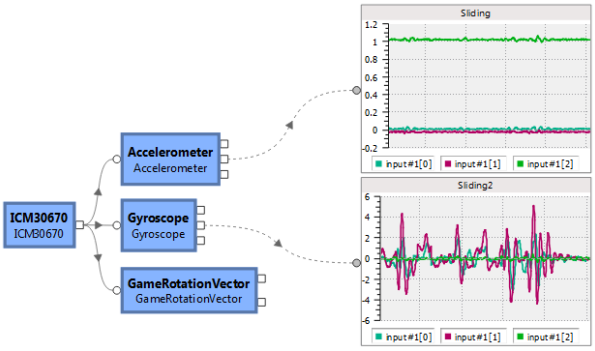



**Figure 17. Screenshot of running SensorHub Framework and SensorStudio flow**

Now we are going to display the output of the GameRotationVector block. Its output is a Quaternion type. The recommended panel to display an orientation is the "Cube" panel, but it expects a RotationMatrix as input. We will need to add a "QuaternionToRotMatrix" conversion block in between, as follow:

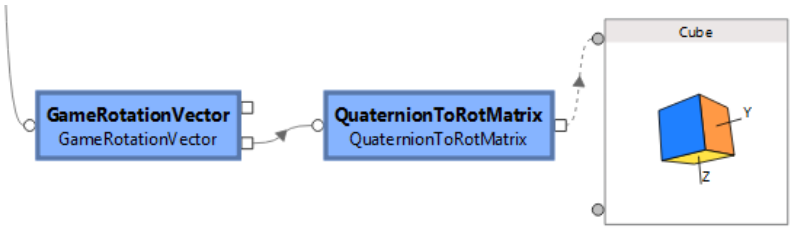



**Figure 18. Screenshot of GameRotationVector block & Cube Panel to Visualize Device Orientation**

Now you can save your flow using the "File" menu, "Save As" menu item (or Ctrl+Shit+S)

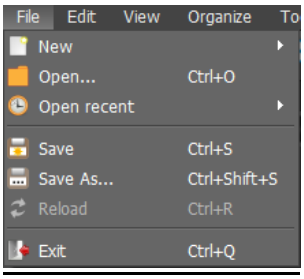



**Figure 19. Screenshot of File Menu Save As…**

## 3.6. DESIGN YOUR SENSOR ALGORITHM

In this section, you will learn how to add an algorithm, written in C code, to the InvenSense sensor framework running on the target.

The first step to create your algorithm is to add a CustomSensorDesigner block to the flow and connect it to the output of the sensors fusion supported by the embedded device.
For this example, we will use the output of the GameRotationVector block:

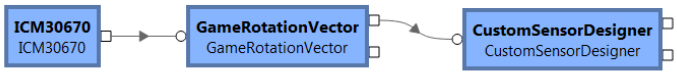



**Figure 20. Screenshot of Add CustomSensorDesigner to create your algorithm in C code**

Double click on the CustomSensorDesigner block to bring up the C editor for implementing your code. Note that this editor is also available when you have selected the CustomSensorDesigner block from the "ImplCode" property of the "Properties" dock.

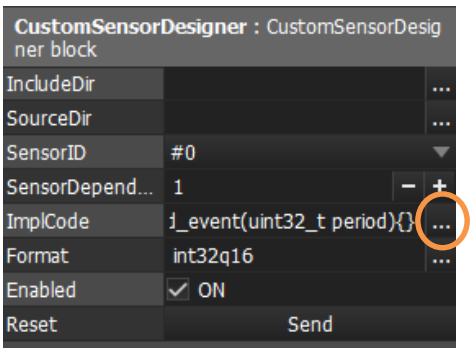



**Figure 21. Screenshot of CustomSensorDesign Properties**

You you will notice in the C editor that, for the block creation, a code skeleton has been provided by default, so the algorithm creation process is simplified. This code skeleton is using the accelerometer data, but we will only need little modifications to use the game rotation vector data.

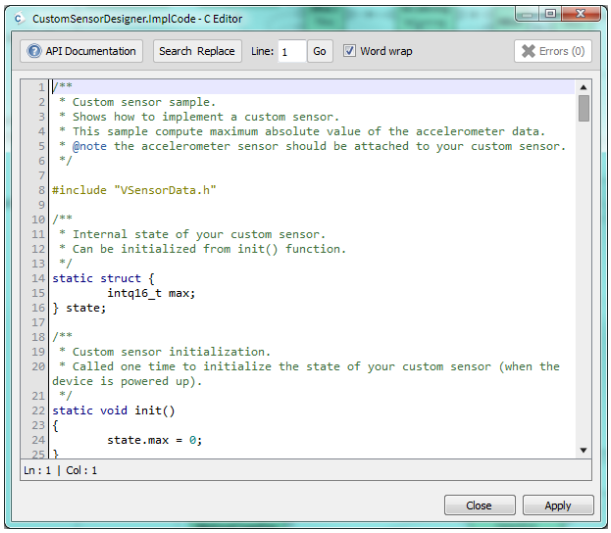



**Figure 22. Screenshot of CustomSensorDesign C Editor**

The skeleton code serves as a reminder of the APIs a CustomSensor code has to implement in order to integrate smoothly in the InvenSense sensor framework.
For this example to be educational, we will keep the code very simple and just try to use the yaw angle, detect a 30° horizontal rotation of the board, and report a Door Opened/Closed message.

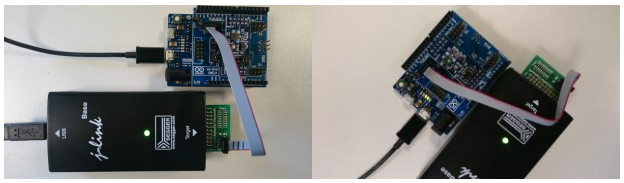


**Figure 23. Picture of ICM-30670 Arduino horizontal rotation**

We will start with the `init()` function. The beginning of the provided skeleton code follows, and we replaced the skeleton code with the following, including declarations and `init()` code:

```
#include "VSensorData.h"

/**
 * Internal state of your custom sensor.
 * Can be initialized from init() function.
 */
static struct {
   intq16_t max;
} state;

/**
 * Custom sensor initialization.
 * Called one time to initialize the state of
your custom sensor (when the device is powered
up).
 */
static void init()
{
   state.max = 0;
}
```

```
#include "VSensorData.h"
#include <math.h> // for acos

/**
* Door opening detection threshold in degree
* Change this for tuning detection sensitivity
*/
#define
DOOR_OPENING_DETECTION_THRESHOLD_ANGLE_RISE
(30)
#define
DOOR_OPENING_DETECTION_THRESHOLD_ANGLE_DROP
(15)

/**
 * Internal state of your custom sensor.
 * Can be initialized from init() function.
 */
static struct {
   int32_t door_open_detected;
   float angle;
} state;

/**
 * Custom sensor initialization.
 * Called one time to initialize the state of
your custom sensor (when the device is powered
up).
 */
static void init()
{
   state.door_open_detected = false;
   state.angle = 0;
}
```

**Figure 24. Original Skeleton Code Versus DoorOpeningDetection Code for `init()`**

You can leave the `subscribe_event()`, `unsubscribe_event()`, `period_event()` and `game_rotation_vector_period_event()` functions unmodified as we don't want to customize their behavior in this simple code. You will, however, need to rename `accelerometer_period_event()` to `game_rotation_vector_period_event()`.

Modify and rename the `accelerometer_vector_data_event()` function as follow:

```
static void accelerometer_data_event(uint32_t
timestamp, void* data, uint16_t len)

{
   /* unused parameter */
   (void)len;

   /* convert data to accelerometer data event
*/
   assert(len ==
sizeof(VSensorDataAccelerometer));
   VSensorDataAccelerometer* acc =
(VSensorDataAccelerometer*) data;

   /* compute maximum */
   state.max = acc->x;
   if(acc->y > state.max)
      state.max = acc->y;
   if(acc->z > state.max)
      state.max = acc->z;









   /* send the maximum value as the data event
of your custom sensor */
   notify(timestamp, &state, sizeof(state));
}
```

```
static void
game_rotation_vector_data_event(uint32_t
timestamp, void* data, uint16_t len)
{
   /* unused parameter */
   (void)len;

   /* convert data to quaternion data event */
   assert(len ==
sizeof(VSensorDataQuaternion));
   VSensorDataQuaternion* q =
(VSensorDataQuaternion*) data;


   state.angle = 2 * acos(intq30_to_float(q-
>w)) * 180 / 3.14159;

   // Test if angle is > threshold
   if ((state.angle >
DOOR_OPENING_DETECTION_THRESHOLD_ANGLE_RISE) &&
       (state.door_open_detected == 0))
   {
      state.door_open_detected = 1;
   }
   else if ((state.door_open_detected == true)
&&
       (state.angle <
DOOR_OPENING_DETECTION_THRESHOLD_ANGLE_DROP))
   {
      state.door_open_detected = 0;
   }

   /* send data event */
   notify(timestamp, &(state), sizeof(state));
}
```

**Figure 25. Original Skeleton Code versus DoorOpeningDetection code for data_event()**

The complete list of data types and functions available for you to use in your CustomSensor code are listed in the 'Extension Framework Documentation', please refer to "3.7 Sensor " or click on the "API Documentation" button of your CustomSensorDesigner "C Editor".

Now that we are done with coding, we need to define the data type format of the CustomSensorDesigner output for the flow (that should be consistent with the `notify()` calls from you code).

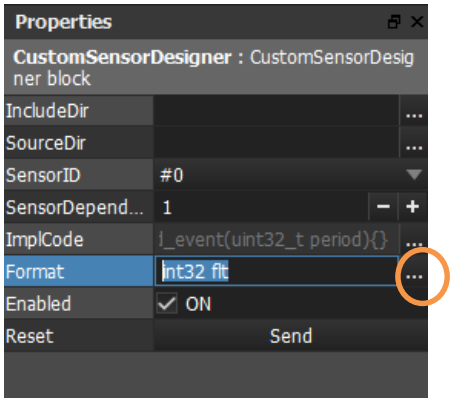



**Figure 26. Screenshot of CustomSensorDesigner Properties Format**

The complete list of data type format available for use in the CustomSensorDesigner output can be obtained by selecting the Format field. The Help window will then show all the options available:

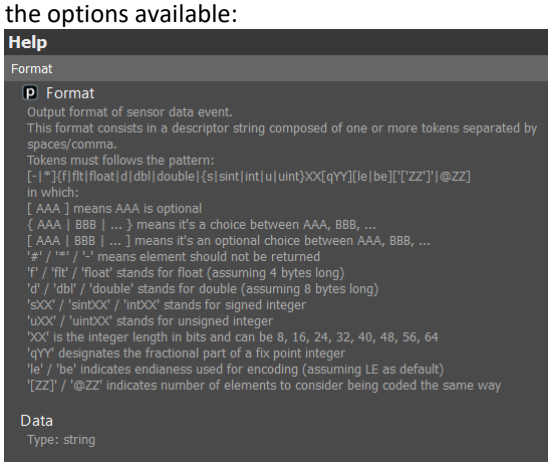



**Figure 27. Screenshot of CustomSensorDesigner Properties Format help**

**Warning:** beware of padding! When defining a structure in your custom sensor passed to the `notify` function, it will be sent to the host as one. Depending on the structure field, the compiler can add padding between the members that you need to take into account when decoding your data.
For example, if you define the following structure:

```
struct data {
    uint8_t data1;
    // 3 bytes of padding here
    uint32_t data2;
};
```

The compiler may add padding between .data1 and .data2. To properly decode this structure into studio, you need to use the following decoding string: ***u8 *u8@3 u32*** (the ***u8@3** indicates to the decoder to read 3 bytes and ignore them).

More on padding: https://en.wikipedia.org/wiki/Data_structure_alignment#Data_structure_padding

As the code of the CustomSensorDesigner is emulated on your computer (using the GameRotationVector data coming from the embedded device), you can already add a Console panel to display the open door state and the angle of the door, and a Color panel to visualize the open door state:

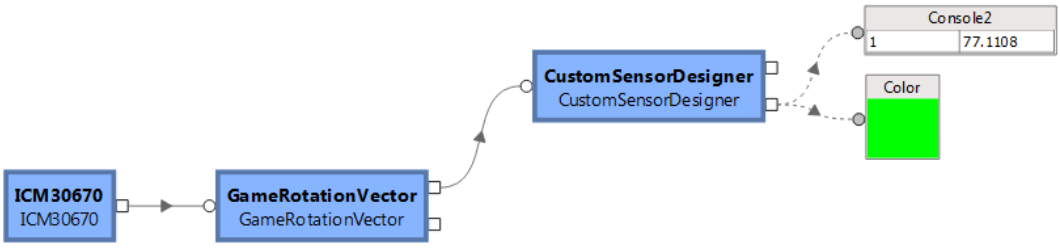



**Figure 28. Screenshot of DoorOpeningDetection flow (emulated)**

The Color panel will change color, when preforming a yaw angle (flat rotation) of the board of more than 30°, as per `game_rotation_vector_data_event()` function implementation seen before.

Everything is now ready to build the code and run it on your embedded device; and this is just one click away:


**Figure 29. Screenshot of Device toolbar Build button**

This will bring up the build configuration dialog:

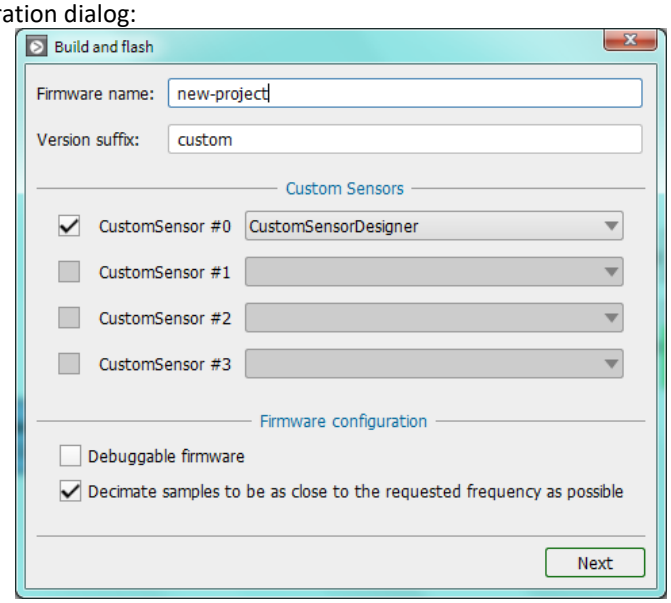



**Figure 30. Screenshot of "Build and flash" configuration**

Make sure your CustomSensorDesigner is enabled. Press the "Next" button will start the build process.

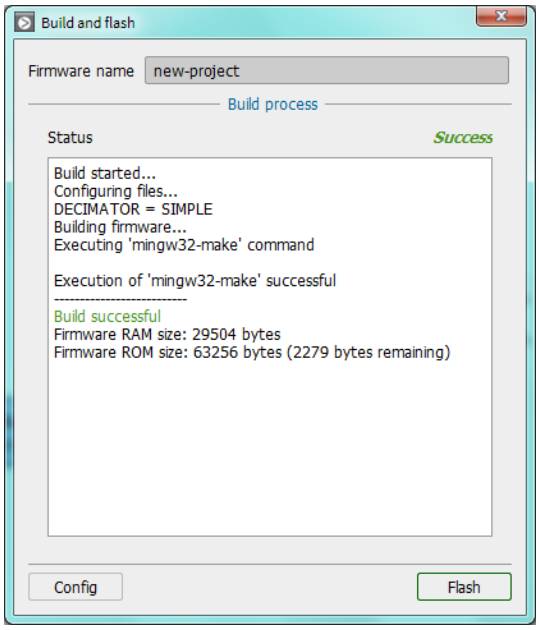



**Figure 31. Screenshot of "Build and flash" window successful build**

You will get GCC compiler warnings and errors details (if any) listed in the status area.

Following a successful build, you can press the "Flash" button; this brings up the Flash window as follow:

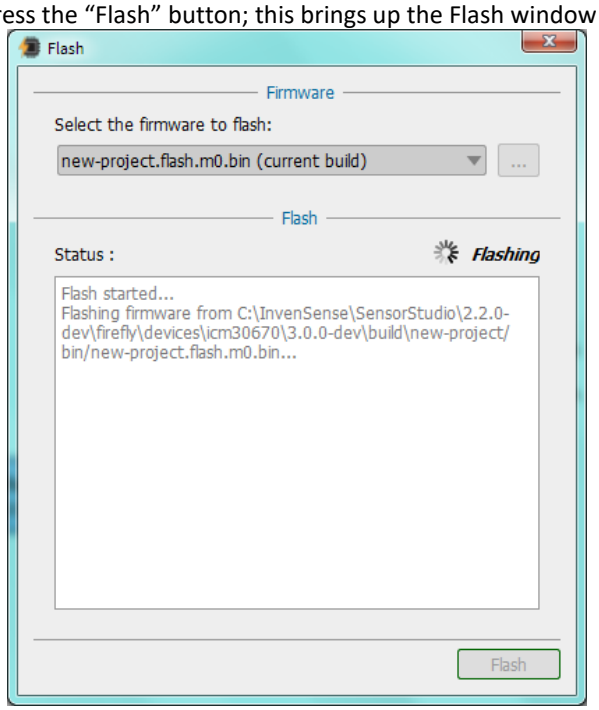



**Figure 32. Screenshot of FireFly "Flash" window**

Upon successful flashing, you can now close the "Flash" window. Via the properties of your device block, you can verify the name of the firmware is no longer the default name but now corresponding to the name you previously entered in Build Config windows:

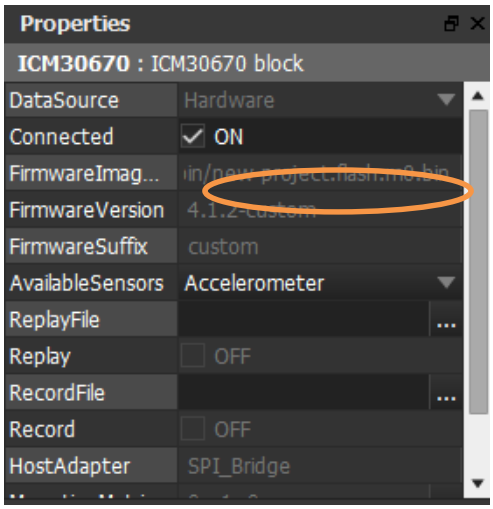



**Figure 33. Screenshot of firmware image path**

You should also see your Custom Sensor among the AvailableSensors:

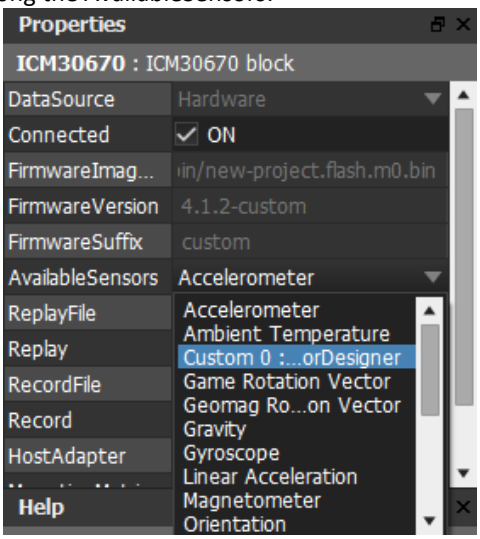



**Figure 34. Screenshot of the AvailableSensors of a device block**

Now that your CustomSensor code has been loaded and is running in the embedded device, you can add a CustomSensor block to retrieve its data, and visualize its output via a Color panel. This shows that the same DoorOpenDetection code can be running in the embedded device ("CustomSensor block") and in the desktop CPU, at the same time.

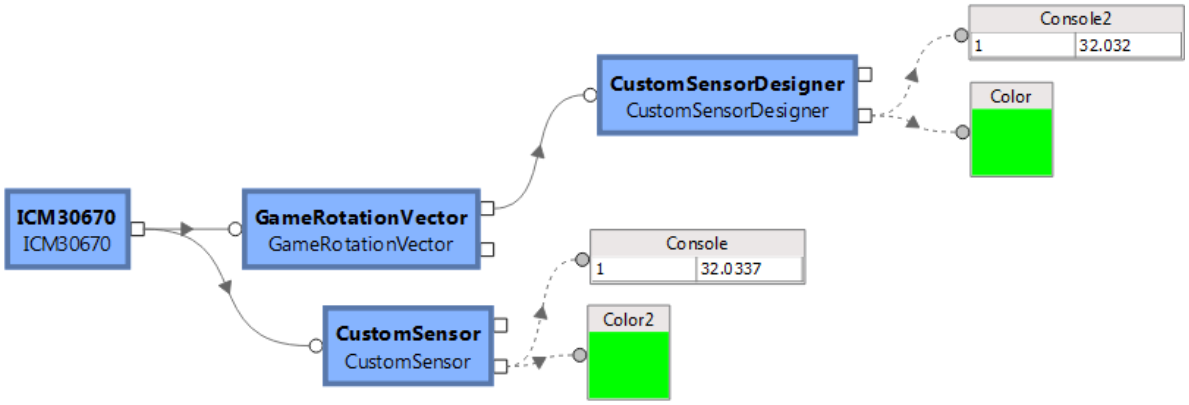



**Figure 35. Screenshot of DoorOpeningDetection flow (emulated on top, embedded in the bottom)**

When you built your firmware, you assigned an ID to your CustomSensorDesigner block. In the CustomSensor block, you can choose which ID to retrieve via the SensorID property. Note that the Format will automatically be fetched and filled from the embedded device when built via SensorStudio.

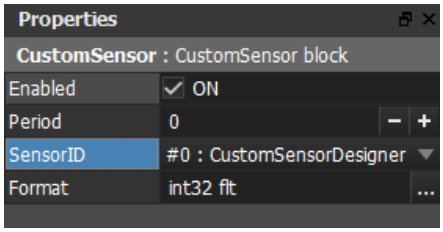
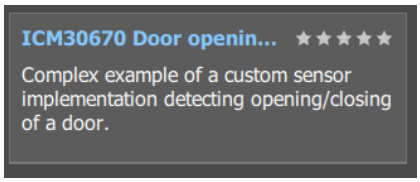



**Figure 36. Screenshot of SensorID property of a CustomSensor block**

The final project is available for ICM-30670 and GSH from the SensorStudio Welcome Screen



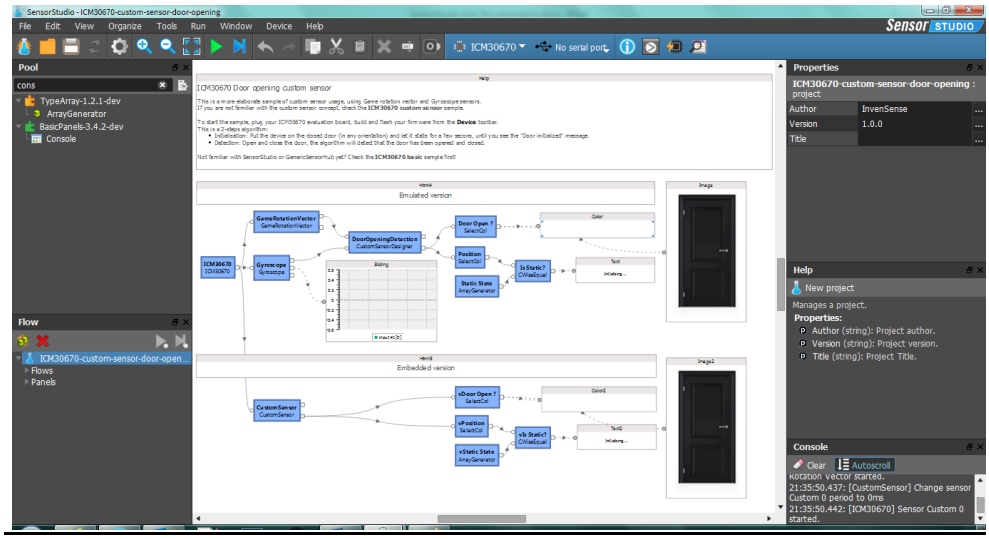



**Figure 37. Screenshot of Door Opening Detection sample**

Additional information and visual elements have been added to the complete flow. The algorithm is also more complex in order to support multiple orientations of the device.

You will notice the “image” panel is being used to display the “Opened Door” and “Closed Door” image files, depending on the output of the CustomSensor ID #0. Left click on the image panel and click on the “…” button of its “Properties” window:

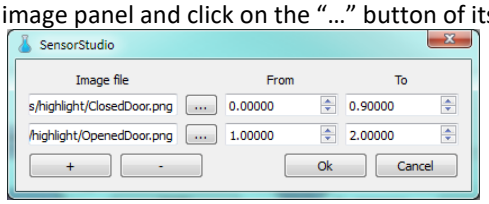



**Figure 38. Screenshot of Image Panel**

Again, you will notice that both the “CustomSensorDesigner” output (running on the computer) and the “CustomSensor” ID #0 output (running on the embedded device) are being displayed. This can be useful to speed up the algorithm development time, as running embedded always requires additional build and flash steps.

## 3.7. SENSOR FRAMEWORK EXPLAINED

The Sensor Framework includes a communication mechanism to **notify** about and **subscribe** to produced data.
Creating your CustomSensor code, with CustomSensorDesigner block, involves understanding the basic operation of InvenSense Sensor Framework.

- Your CustomSensor `init()` function is called once to initialize the state of the custom sensor (after the embedded device is powered up, when sensor framework is started: ICM-30670 or GSH “Connected” property)
- Your CustomSensor `period_event()` function is called just before your custom sensor is started. This allows you to configure the frequency of your subscribed sensors (like accelerometer). SensorStudio implements this by default; you may change this to implement decimation on all sensors at once.
- Your CustomSensor `subscribe_event()` function is called by Sensor Framework when enabling the sensor. It lets you register your sensor dependencies to the Sensor framework. SensorStudio implements this by default (all visual connections are already taken into account).
- Your CustomSensor `game_rotation_vector_data_event()` function is called upon receiving a new sensor data of a given type. You will have as many **`XYZ_data_event()`** functions to implement as you have subscribed sensors. This is the place to store and process your data (just replace **`game_rotation_vector_`** by other sensor data type). When you are done with your processing, use the `notify()` function to pass on your produced data.
- Your CustomSensor `game_rotation_vector_period_event()` function is called when the period of the given subscribed sensor is changed. You may implement this function if you have subscribed sensors running at difference frequencies. SensorStudio implements this by default; you may change this to implement decimation on the given sensor.
- Your CustomSensor `unsubscribe_event()` function is called by Sensor Framework when disabling the sensor. It lets you unregister your sensor dependencies to the Sensor framework. SensorStudio implements this by default (all visual connections are already taken).

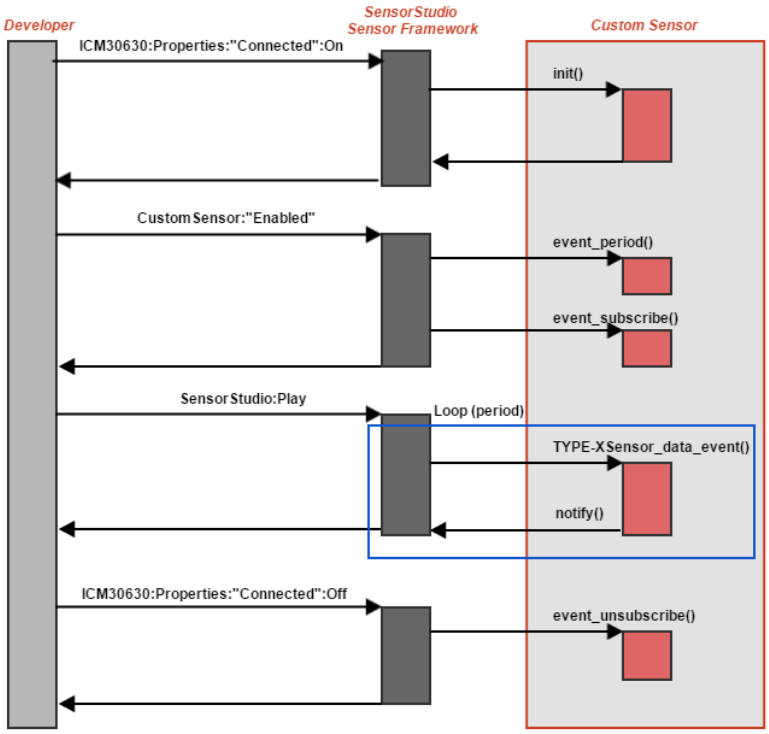



**Figure 39. Sensor Framework overall operation diagram**

The complete list of APIs and detailed documentation of the Sensor Framework is provided in the "Doc" folder of you SensorStudio install, look for the 'Extension Framework Documentation' link, or directly from the C Editor of the CustomSensorDesigner block.
You may need to pay some special attention to the input frequencies of your CustomSensors, and how you deal with incoming data from the other Sensors, depending on your application needs.

We have seen before in this section, the Sensor Framework relies on a communication mechanism, based on subscribe & notify messages. In case several CustomSensors subscribe to a given Sensor and specify different frequencies, the Sensor Framework will ensure this given Sensor is programmed to produce data at the faster data rate.

For example, two CustomSensors could request Accelerometer Sensor data at different frequencies:
- CustomSensor A at 10Hz
- CustomSensor B at 100Hz

The Sensor Framework would adjust the accelerometer to 100Hz and it would be up to the CustomSensor A to decimate or cope with incoming data at a faster rate.

The DoorOpeningDetection code sample has been designed to provide a working exemple of decimation, please report to the `period_event()` and `game_rotation_vector_period_event()` functions.

You should also note the Sensor Framework ensures the Sensor data output is decimated so the Application processor is getting the specified frequency. You might want to double check the Sensor frequency and how you are implementing the `period_event()`, in case, you may experience different behaviors between your CustomSensor running on the embedded device ("CustomSensor block") and your CustomSensor running in the desktop CPU.

## 3.8. DESIGN YOUR SENSOR DRIVER

InvenSense Sensor Framework comes with pre-integrated sensors: magnetometer, proximity, temperature, pressure. These sensors are built-in and always available.
You may also add your own new sensor type to the Sensor Framework, using the AuxiliarySensorDesigner block. This section of the document will guide you in doing so.

Before getting into the creation details, it is important to note the AuxiliarySensorDesigner is somewhat similar but different from the CustomSensorDesigner block discussed in sections 3.6 & 3.7. Both blocks support creating code that integrates the InvenSense Sensor Framework, which you can access through a CustomSensor block, the difference between those blocks lies in:

- CustomSensorDesigner block is software that depends & process other sensors outputs to create its own outputs
- CustomSensorDesigner block is software that does not access hardware directly (code can be run within SensorStudio desktop executable)
- AuxiliarySensorDesigner block is software that doesn't depends & process other sensors outputs
- **AuxiliarySensorDesigner block is software that access hardware directly to create its own outputs (code cannot be run within SensorStudio desktop executable)**

Now let us dive in the creation details; just look up for the AuxiliarySensorDesigner block from the Pool dock:


**Figure 40. Screenshot of AuxiliarySensorDesigner in Pool window**

Drag and drop the AuxiliarySensorDesigner block onto the flow area (left click & hold on the AuxiliarySensorDesigner block from the Pool dock; move to the flow area and release the left click button to place the AuxiliarySensorDesigner block close by the device block). Add a corresponding CustomSensor block and create the block connections on the flow area as follow:

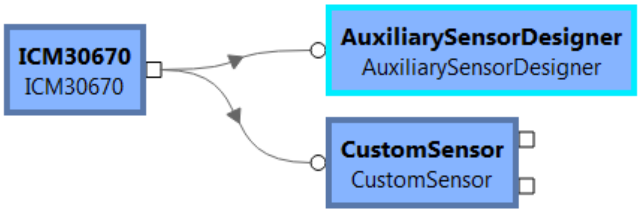



**Figure 41. Screenshot of AuxiliarySensorDesigner flow**

Default code for reading a 3rd-party Magnetometer is provided when creating a new AuxiliarySensorDesigner, just to help you and serve as a working sample. You can access the AuxiliarySensorDesigner code by double clicking on the blox, or through the ImplCode "…" button of the block Properties dock:

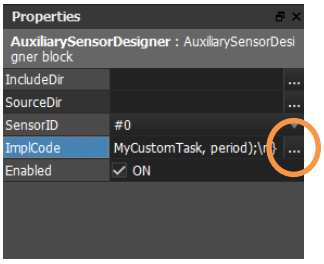



**Figure 42. Screenshot of AuxiliarySensorDesigner Properties**

This will bring up the C Editor window:

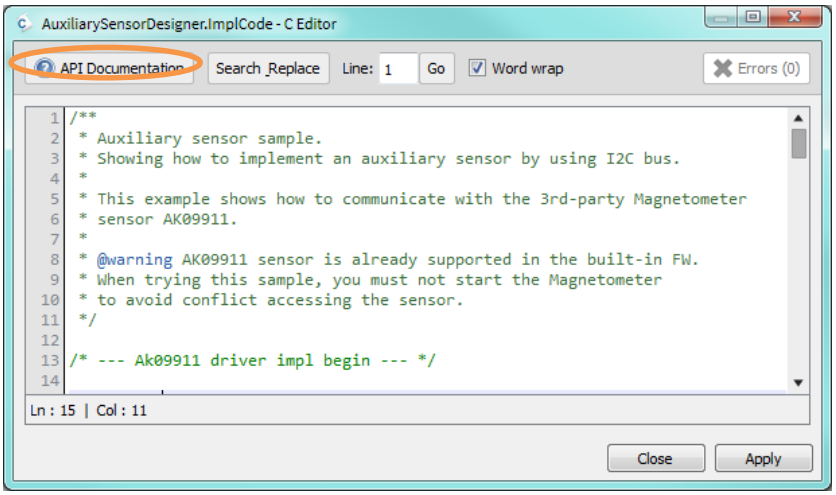



**Figure 43. Screenshot of AuxiliarySensorDesigner Code Sample**

The complete list of APIs and detailed documentation of the Sensor Framework is provided directly from the C Editor "API Documentation". We will now review the APIs you need to implement your own sensor driver.

Scroll down the C Editor windows and locate the `write_reg_hook()` function:

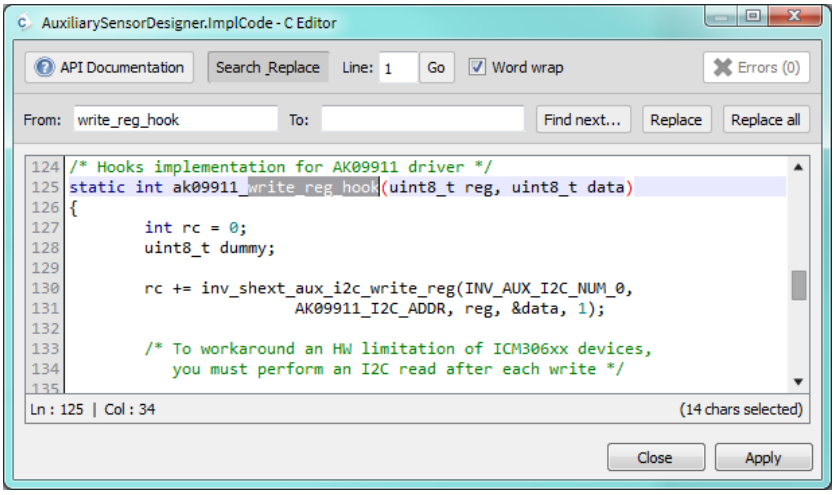



**Figure 44. Screenshot of AuxiliarySensorDesigner code sample I2C read/write**

`inv_shext_aux_i2c_write_reg()` is used to write register over Auxiliary I2C bus and `inv_shext_aux_i2c_read_reg()` is used read register over Auxiliary I2C bus. These are the basis for your access to your hardware.

Now, locate the `MyCustomTaskCode()` function. This task will be your own task, responsible for polling data from your sensor at a specified Period.

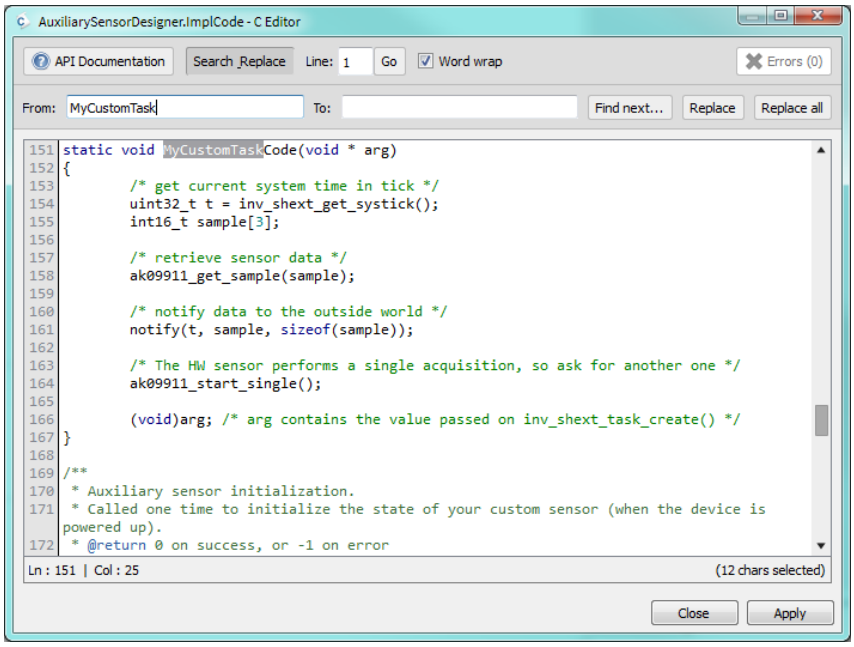



**Figure 45. Screenshot of AuxiliarySensorDesigner code sample `MyCustomTaskCode()`**

Therefore `MyCustomTaskCode()` implementation :

- Gets current system time in tick using `inv_shext_get_systick()`, for later timestamping purpose.
- Retrieves new samples from your sensor (see `ak09911_get_sample()` implementation).
- It then uses the `notify()` function to share the produced data with other CustomSensor code, that might be using it.
- Finally it triggers a new acquisition, so next time the `MyCustomTaskCode()` is called, the sensor data will be ready to read.

During the Sensor Framework init(), you will initialize your task **MyCustomTask** using `inv_shext_task_create()`

```
/**
 * Auxiliary sensor initialization.
 * Called one time to initialize the state of your custom sensor (when the device is powered up)
 * @return 0 on success, or -1 on error
 *         If an error is returned, the sensor will be automatically
 *         unregistered from the system and become unavailable
 */
static int init()
{
    /* initialize I2C hardware feature */
    if(inv_shext_aux_i2c_init(INV_AUX_I2C_NUM_0,
            INV_SHEXT_AUX_I2C_CLK_400KHZ) != 0)
        return -1;

    /* initialize AK09911 sensor (should fail if sensor is not connected) */
    if(ak09911_init() != 0)
        return -1;

    /* initialize task object that is used in this sample */
    inv_shext_task_create(&MyCustomTask, MyCustomTaskCode,
            0 /* optional pointer passed to MyCustomTaskCode() */);

    return 0;
}
```

**Figure 46. Screenshot of AuxiliarySensorDesigner Code Sample `init()`**

On `subcribe_event()`, you will need to start your task **MyCustomTask** using `inv_shext_task_start()`

```
/**
 * Called when someone (host or another sensor) subscribes to this sensor.
 *
 * It is only called once for the first client.
 * It can be used to reset states or subscribe to other sensors.
 */
static void subscribe_event()
{
    /* Start HW sensor and ask for a data acquisition */
    ak09911_start_single();

    /* In this example we simply start the task that was created in init() function. */
    inv_shext_task_start(&MyCustomTask, 10 /* ms */); /* start with some delay */
}
```

**Figure 47. Screenshot of AuxiliarySensorDesigner Code Sample `subscribe_event()`**

And on `period_event()`,you will need to define the period at which your task **MyCustomTaskCode** will be called using `inv_shext_task_set_period()`

```
/**
 * Called when the requested period (in ms) for this sensor has changed.
 *
 * The requested period corresponds to the minimum of all configured period
 * for each clients of the sensor (eg: if two clients subscribe to this sensor,
 * one requesting a period of 20ms (50 Hz),
 * and the other requesting a period of 10ms (100Hz),
 * the request value for this will be 10).
 *
 * It can be used to configure algorithm or set requested period for other
 * sensors.
 */
static void period_event(uint32_t period)
{
    /* In this example we update the period of our task to read
       data from the HW sensor at the requested rate */

    /* AK09911 does not support polling above 130 Hz */
    if (period < 8)
        period = 8;

    inv_shext_task_set_period(&MyCustomTask, period);
}
```

**Figure 48. Screenshot of AuxiliarySensorDesigner Code Sample `period_event()`**

# 4 SENSORSTUDIO REFERENCE MANUAL

## 4.1. MAIN SCREEN

InvenSense SensorStudio is a graphical User Interface supporting development of sensor data fusion with multiple supported InvenSense chipsets.

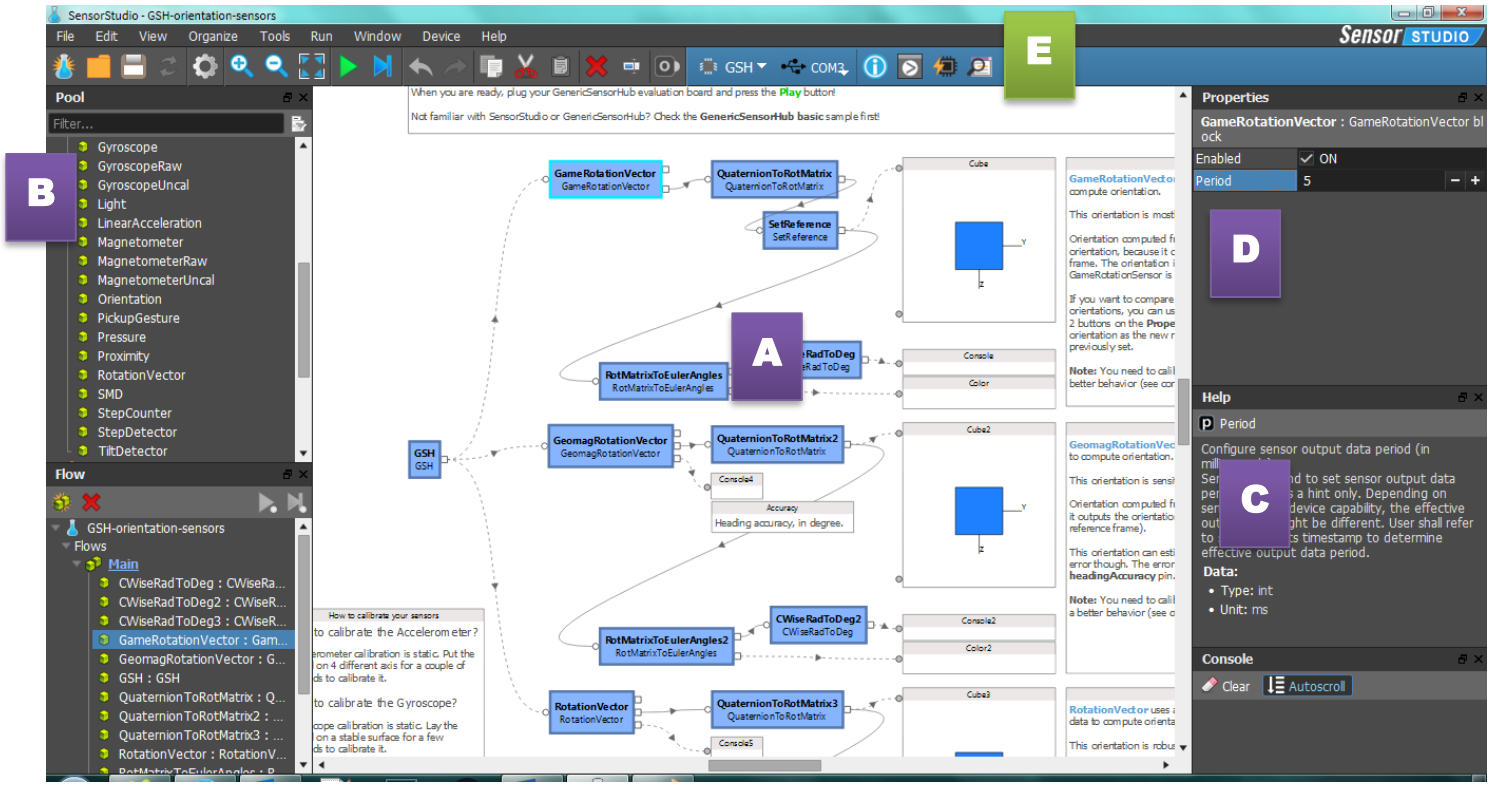



**Figure 49. Screenshot of InvenSense SensorStudio with orientation project file opened**

In the center you find the ***Flow*** section, **A**. It is the core of ***InvenSense SensorStudio*** where you can design your feature by adding and linking different ***blocks*** together and by so composing flows. Here we see the ***flow*** graph from the orientation sample. It starts with Sensors ***blocks***, providing sensors fusion orientation ouputs. Then different ***blocks*** are plugged to deliver at the end an AG, AM and AGM orientation with the Cube panels**.**

We can also see some visualization objects within our scene: they are called ***Panels***.

From **B**, you have access to a pool of ***plugins***, each plugin containing several ***blocks*** / ***panels*** which you can drag in the flow section.

Each block is explained in the Help section, **C**.

You can view and edit blocks and panels properties, once selected, in the Properties section **D**

In the command menu bar, there is a device toolbar, **E**, which allows ***InvenSense SensorStudio*** desktop to control an embedded device.

## 4.2. DEVICE

You may use the device toolbar buttons to manage your SoC device:


The "*Device Information*" button brings up windows with the device name and firmware revision:


The device toolbar will display the selected device. Since SensorStudio 2.2.0, you can design flow for ICM-30670 FireFly or Generic Sensor Hub (ICM20690-based) devices.

When a device is selected, the corresponding blocks are available from the Pool dock.

## 4.3. BUILD CONFIGURATION

The "*Build firmware*" button brings up the "*Build and flash*" window:

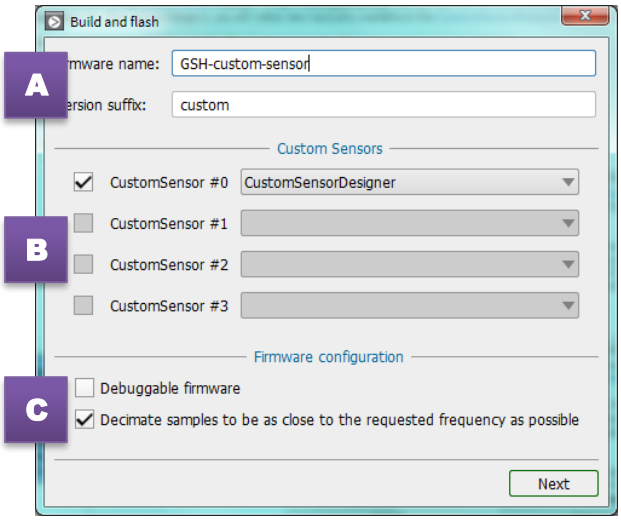




**Figure 50. Screenshot of "Build and flash" configuration**

| | |
|---|---|
| A | ▪ Firmware name is automatically set based on the project name. You specify your own value. The firmware name is the name of the firmware image built.<br>▪ You can set a version suffix to track your firmware integration. Firmware version is displayed in the Information menu from the device toolbar and from the device block properties, once the firmware is connected. |
| B | ▪ The Custom Sensors area is automatically filled with all the Custom Sensor Designer blocks name present in your project. For each custom sensor id available, you can choose which sensor designer you want to embed in your firmware. |
| C | ▪ The "Debuggable firmware" option is required if you want to be able to debug your custom sensor code with Eclipse & JLink (see the debug documentation)<br>▪ The "Decimate samples to be as close to the requested frequency as possible" option enables or disable the decimator module present in firmware that drops or not data based on their timestamp, before sending them to sensor studio. Refer to "*ICM-30670 - eMD Software Guide*" and "*Generic Sensor Hub - Software Guide*" for more about decimation. |

This will configure the InvenSense sensor framework accordingly with the selected features/sensors.
Press the “Next” button when the configuration is done to start the build process.

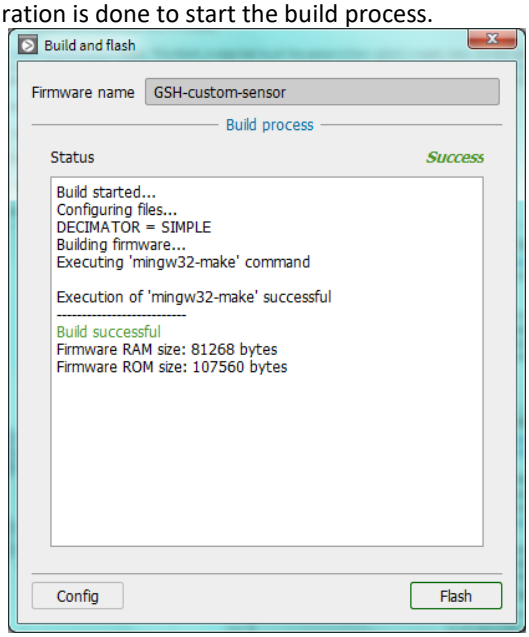
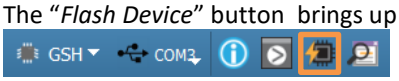



**Figure 51. Screenshot of “Build and flash” window successful build**

You will get GCC compiler warnings and errors details (if any) listed in the status area.

## 4.4. FLASHING FIRMWARE

The “*Flash Device*” button brings up the Flash window:



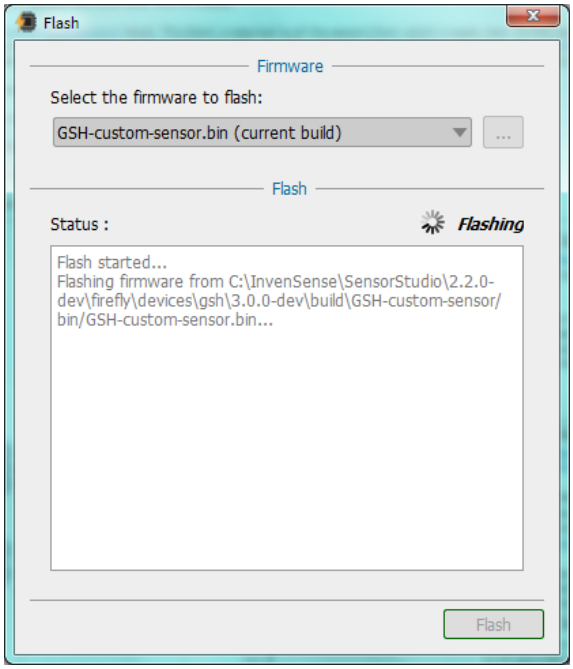




**Figure 52. Screenshot of “Flash” window**

All the actions of the device toolbar are also listed in the Device menu:

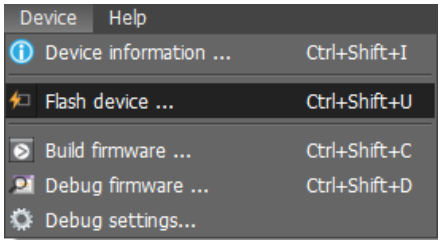



**Figure 53. Screenshot of Device menu**

## 4.5. DEBUGGING FIRMWARE

**Prerequisites:**

- **ICM-30670:** See the document "*005 - ICM-30670 – Usign Eclipse IDE with J-Link*" for more information. This document will explain what you need to install to debug correctly your ICM-30670 board.
- **Generic Sensor Hub:** See the document "*006 - Generic Sensor Hub - Quick Start Guide*" to know how to install *Eclipse* and required plugins.

The "*Debug firmware*" button launchs *Eclipse* to debug your firmware:


**Warning:** If your firmware is not built yet, a warning will indicate you to build it first. In this case, it's important to select a "*debuggable firmware*" to avoid code optimization and allow the permission to debug your firmware:

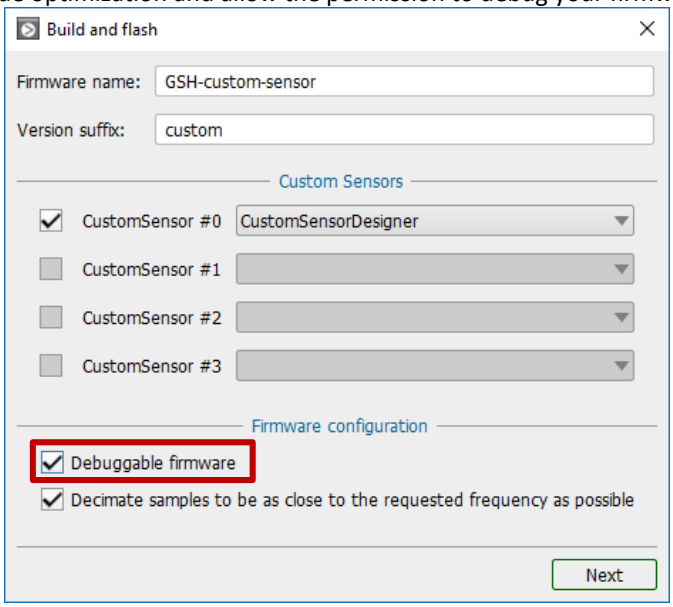



**Figure 54. Screenshot of build configuration dialog**

After your firmware is built, you can click on "*Debug firmware*" button to launch *Eclipse*. Then from *Eclipse*, you will find a project with the same name as your opened *SensorStudio* project. Select your project from "Project explorer" view, click on the right-arrow of "Debug" button from the main toolbar, and then click on the "Debug configurations…" from the drop-down menu:

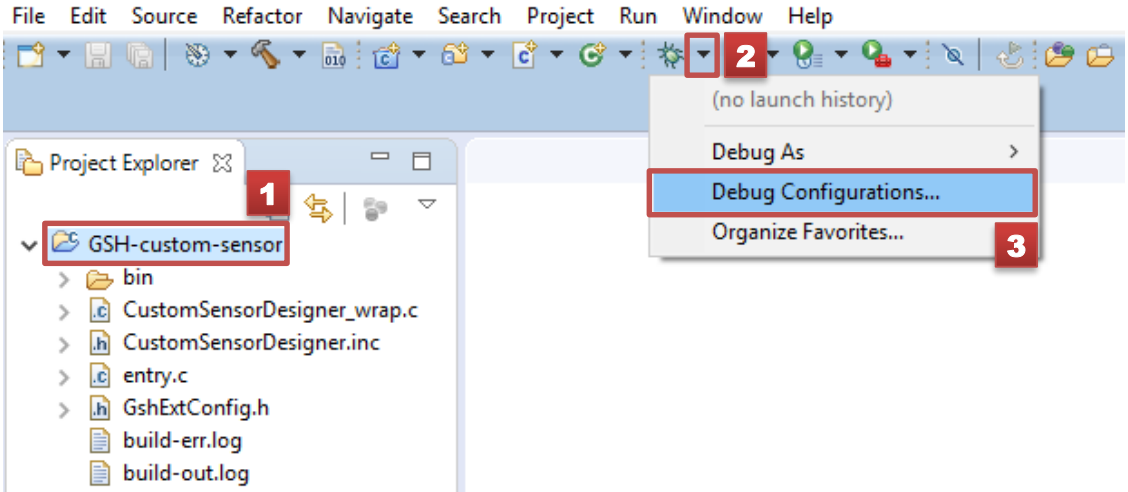



**Figure 55. Screenshot of debug menu from Eclipse**

That will open the debug configuration dialog to choose the dedicated debug launcher for your board and project. According to your board, you will find the debug launcher at different locations:

- **ICM-30670**: under "*GDB SEGGER J-Link Debugging*" category with name "*Firefly-<project-name>*"
- **Generic Sensor Hub**: under "*GDB OpenOCD Debugging*" category with name "*Debug-<project-name>-(no-load)*"

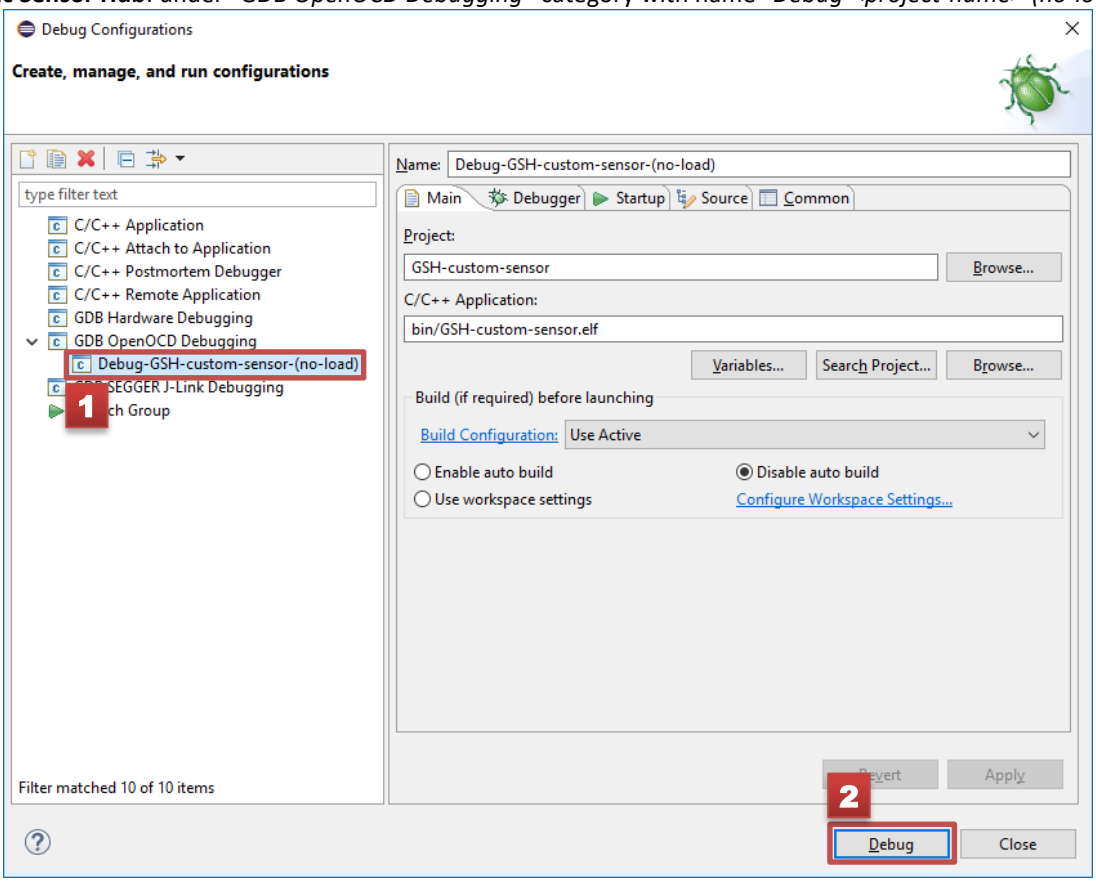



**Figure 56. Screenshot of Debug configuration from Eclipse**

Then you will begin the debugging of your firmware. Eclipse will ask you to switch to "debug" perspective, you can accept. And now can play with breakpoints, watchpoints and debug execution. More information is available from *Eclipse* website: http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Ftasks%2Fcdt_o_debug.htm

**Tip:** If you want debug again the same project, you don't need to access again to this "Debug configurations…" dialog. The debug launcher of your project will automatically appear in the "Debug" menu:

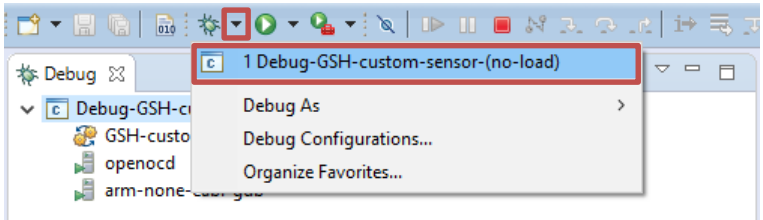



**Figure 57. Screenshot of favourites debug launchers from Eclipse**

## 4.6. FLOW AREA

You may use the View toolbar buttons  to "Zoom In" or Zoom Out" and adjust the flow area zoom level to your convenience.

You may use the "Fit in view" button to let SensorStudio find the zoom level and centering of the view so you can see all your design elements at once.

The "Zoom In", "Zoom Out" and "Fit in view" actions are also available from the View menu:

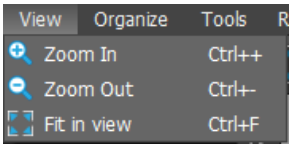



**Figure 58. Screenshot of InvenSense SensorStudio View Menu**

Left click and hold on the view area allows you to pan, e.g. to move around the content of the flow area as you wish when working on your project. Just release the click when the position suits your need.

## 4.7. HELP DOCK

The help dock displays context-specific information on a selected block, pin, or connection in the flow area, e.g. click on a device block (ICM30670 or GSH).

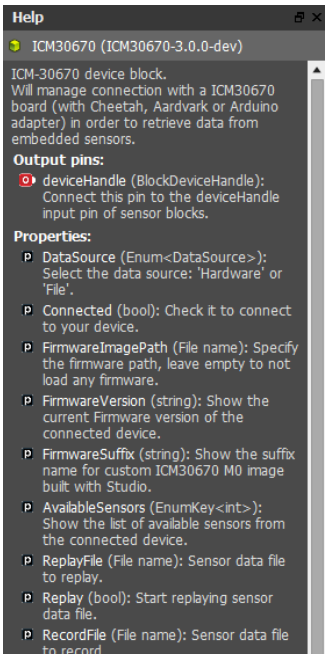



**Figure 59. Screenshot of ICM-30670 Help dock**

The Help dock context can show further details of the options available for any property of the Properties dock, e.g. click on the "Format" property of the CustomSensorDesign Properties dock:

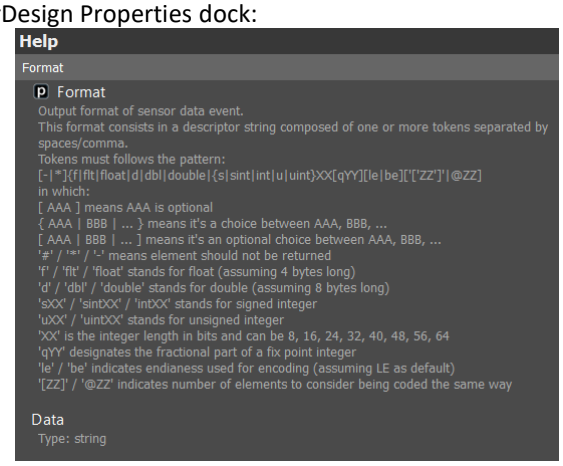



**Figure 60. Screenshot of CustomSensorDesign Properties Format help**

The Help context is highlighted in light blue color in the flow area and in the Properties dock when applicable.

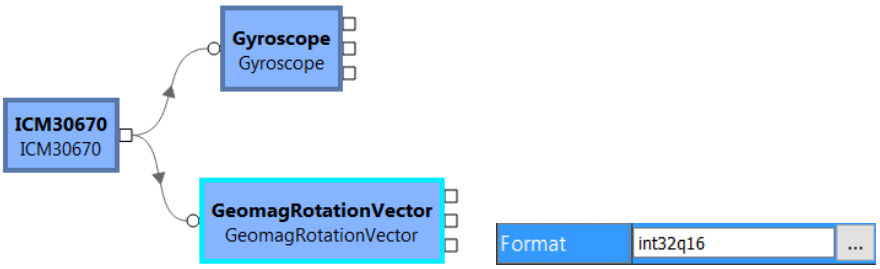



**Figure 61. Screenshot of selected items**

## 4.8. PROPERTIES DOCK

The Properties dock is also contextual, meaning it is related to the selection in the flow area, e.g. click on ICM-30670 device block.

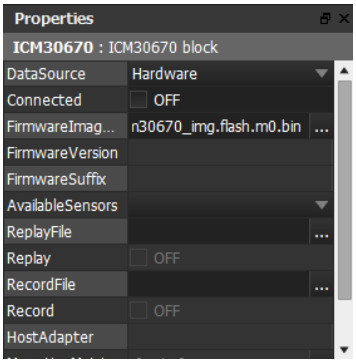



**Figure 62. Screenshot of ICM-30670 Properties dock**

Use the Help dock to find out more details about any given property.

## 4.9. POOL DOCK

InvenSense SensorStudio pool dock lists all software components available.

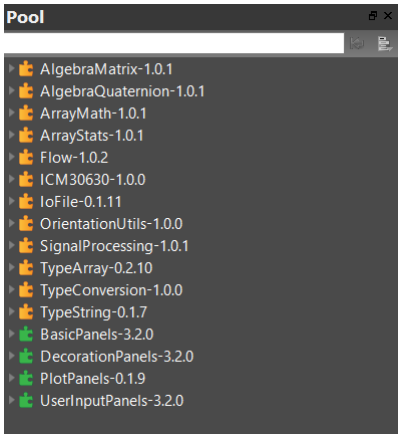



**Figure 63. Screenshot of Pool dock**

The software components are called SensorStudio plugins and there are two types of plug-ins:

Yellow items are groups of processing elements designed to compute various things, from basic matrix, arrays operations, to more complex high-level functionalities. By default, these processing elements run within SensorStudio desktop CPU.

Green items are groups of visualization elements, called Panels, designed to display various things, from basic text, to more advanced graphics.

The “filter” field allows you to enter part of the block you might be looking for, and perform a search from the blocks & panels Pool dock:

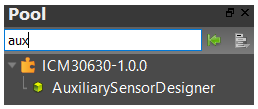



**Figure 64. Screenshot of AuxiliarySensorDesigner in Pool dock**

## 4.10. FLOW DOCK

InvenSense SensorStudio Flow dock provides you a list view of the elements present in your project (while the Flow area is a graphical view).


**Figure 65. Screenshot of Flow dock**

In the Flow dock of the Orientation sample above, we can find the various blocks (ICM-30670 device; GameRotationVector, GeomagRotationVector & RotationVector sensors; and various computing blocks to transform the quaternion outputs into Euler angles).

The Start button  will start the Flows.

SensorStudio runtime will traverse the graph, in order, from the device outputs and progressing, executing each computing block one at a time, within one Flow. Each Flow is executed in an independent thread. The graph traversal is repeated based on the MinPeriod property value of the corresponding Flow.
Depending on your project complexity, you may have chose to create different Flows running within different threads, using the Create Flow button  .

You should notice the InvenSense SensorStudio "Start toolbar" is now showing the running icon  , while it was previously showing the "Start" icon  .

Consistently, the Start button in the Flow dock has transformed into a Stop button  .

These actions are also available from the Run menu and supports keyboard shortcuts as follow

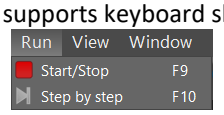


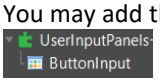

**Figure 66. Screenshot of Flow menu**

The "*Step-By-Step*" button will start the Flows execution one step at a time. In this case, the Flows are executed once, and the graphs are traversed from the device outputs, following the graph's ordering, processing one block at a time, until all blocks have been executed once, then stop. You may execute another Flow step using the "*Step-by-Step*" button again.

## 4.11. LUA SCRIPT

LUA scripting is an efficient way to automate several manual actions in a single click push button interface.

You may add this push button to your Flow searching the Pool dock for the ButtonInput panel:




Once dragged in the flow, you can then click on the "…" button of its Script property

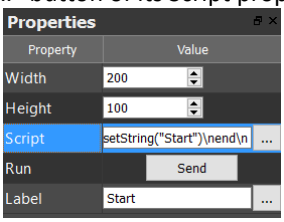



**Figure 67. Screenshot of ButtonInput Properties**

This will bring up the Lua Editor window with some sample code as follow, to help you create your script

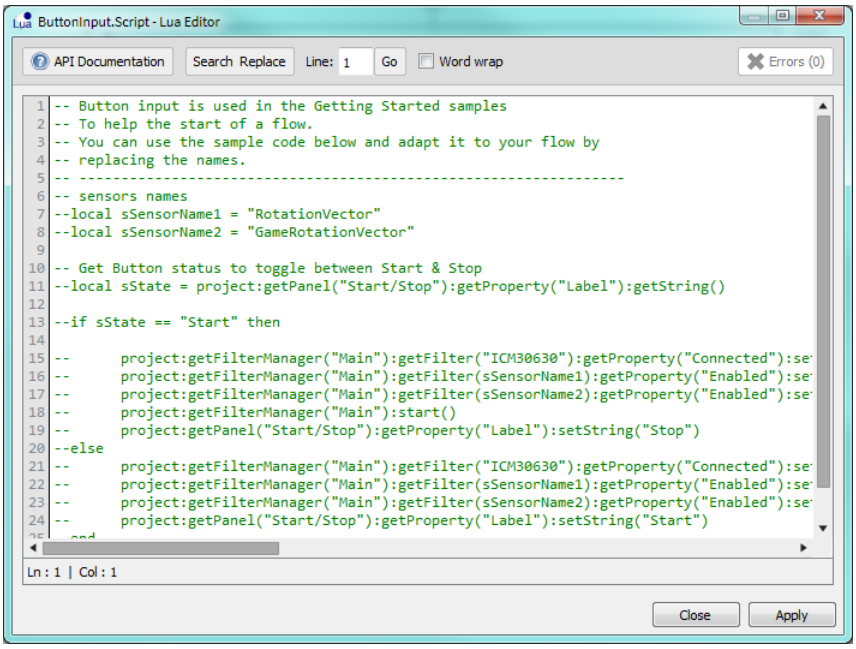



**Figure 68. Screenshot of Lua Script Default Code Sample**

## 4.12. RECORD & REPLAY

It is possible to **record** to a csv file, all data received from a device. Such file can be analysed and processed by external tools.
It is also possible to **replay** a previously recorded file. It helps designing a flow with repeatable real data.

### 4.12.1 How to use the Record feature?

- Make sure the *DataSource* property from the Device block is set to *Hardware*
- Specify the path of file that will hold recorded the data in the RecordFile property
- Start your flow by clicking on the **Start** button
- Check the *Record* property when you are ready to record data

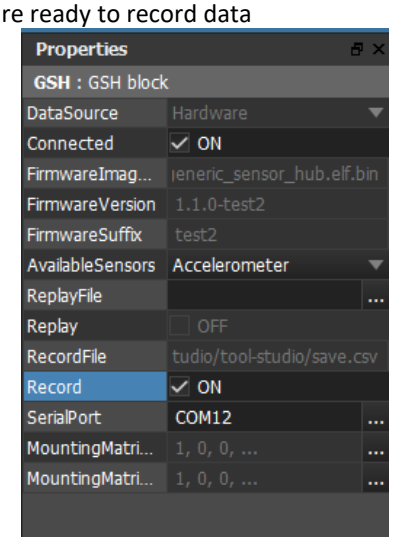



**Figure 69. Record properties**

**Warning:** when checking the *Record* checkbox, the record file will be overwritten.

You can change the sensor configuration or enable/disable sensors while recording.
Feel free to open the recorded file in a text editor to see how it looks.

And once you're happy with the data you've recorded, you can replay them into your flow!

### 4.12.2 How to use the Replay feature?

- Change the *DataSource* property from the Device block from *Hardware* to *File* (make sure the *Connected* property is unchecked to be able to change the data source)
- Specify the path of a the previously recorded file in the *ReplayFile* property
- Start your flow by clicking on the **Start** button (if not already)
- Check the *Replay* property to start replaying your file into your flow

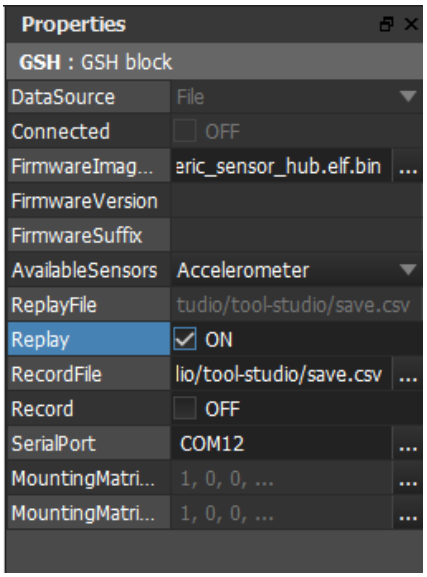



**Figure 70. Replay properties**

**Warning:** only the data from the **enabled** Sensor blocks present in the flow will be replayed (even if they are present in the recorded file)

# 5 GOING BEYOND

This section introduces how to go beyond SensorStudio with your ICM-30670 development kit and the standard GSH development kit.

## 5.1. ICM-30670 - DEVELOPPING YOUR SOLUTION ON THE ARDUINO ZERO

When using the ICM-30670 with SensorStudio, SensorStudio automatically flashes the Arduino Zero with a bridge application to encapsulate the protocol to communicate with SensorStudio. ICM-30670 eMD is then driven by SensorStudio via this bridge, and therefore is not standalone. You can make your ICM-30670 standalone by developing your own Arduino Zero application via the Arduino IDE.
Documentation on setting up your Arduino development environment and developping applications is covered in the ICM-30670 eMD Software guide (available from the Help menu in SensorStudio). Section "*2.4.3. Package installation"* of such documentation refers to the InvenSense ICM-30670 eMD Developer Kit. This kit is installed along SensorStudio, and is available under C:\InvenSense\SensorStudio\2.2.0\firefly\devices\icm30670\3.0.3.

## 5.2. PORTING GSH TO ANOTHER PLATFORM AND USING IT WITH SENSORSTUDIO

Although partially supported, it is still possible to support from SensorStudio, a GSH device running on a different platform than the official one consisting in the Nucelo F411-RE and InvenSense Nucleo Carrier board.
Please read the entire sections before experimenting.

Main limitations are:

- Flashing firmware only works for ST Nucleo board (see § 5.2.3)
- Serial communication between the board and SensorStudio is not configurable (see § 5.2.2)
- Only Eclipse can be launch from SensorStudio for firmware debugging
- Templates location for building custom firmware is not dynamically configurable (see § 5.2.5)

### 5.2.1 Porting GSH firmware to another platform

The default firmware sources for the standard development kit is available in:

- *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/sources/*

Porting the GSH firmware to another board/MCU is not covered by this document. Refer to *Generic Sensor Hub - Software Guide* document for details on how to port GSH firmware to another platform.

### 5.2.2 Retrieving sensor data from a GSH port into SensorStudio

SensorStudio communicates with a GSH device using a serial interface and a proprietary protocol.
You can select the COM port to use for communication from the Device Toolbar as highlighted on Figure 71.


**Figure 71 Screenshot of COM port selection from Device Toolbar**

- **Important:** SensorStudio will attempt to flash the default firmware image (that works only for the official development kit). To prevent this, you must clear the *FirmwareImagePath* property before starting the flow or checking the *Connected* property. See § available in:
  - *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/doc/*

Flashing a custom firmware image from SensorStudio

**Notes:**

- Serial interface parameters are not yet configurable from the device driver. The device driver currently expects the following parameters:
  - Baudrate: 2000000
  - Hardware flow control enable
- If you're using the InvenSense Nucleo carrier board (connected to another MCU), you can reuse the on-board FTDI converter for communicating with SensorStudio.
- The custom firmware must properly implement the GSH protocol as described in *Dynamic Protoco Description* document available in:
  - *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/doc/*

### 5.2.3 Flashing a custom firmware image from SensorStudio

Flashing a firmware image is currently done using OpenOCD. OpenOCD commands used for flashing are not yet configurable, and only the ST Nucleo F411-RE is supported. Hence, it is not possible to flash a custom firmware image for another board directly from SensorStudio.
Flashing the firmware to your board must be done by an external tool.
To prevent SensorStudio from attempting to flash a firmware image, you must clear the *FirmwareImagePath* property before starting the flow or checking the *Connected* property.

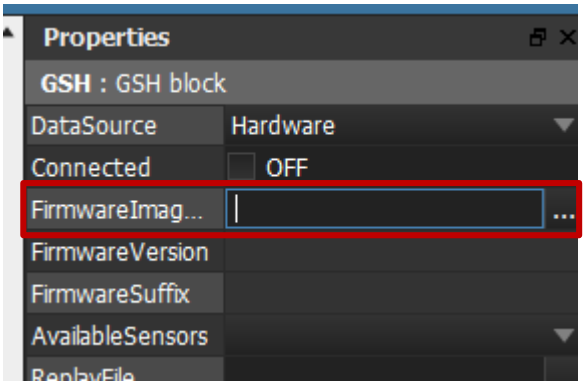



**Figure 72 Screenshot of FirmwareImageProperty from GSH device block**

**Notes:**

- The OpenOCD tool used for flashing the ST Nucleo is located under:
  - *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/openocd/bin/openocd.exe*
- The command arguments are:

```
- -f "board/st_nucleo_f4.cfg"
  -c "gdb_port pipe"
  -c "adapter_khz 1800" // to avoid warning: "Info : Unable to match requested speed
  2000 kHz, using 1800 kHz"
  -c "reset_config connect_assert_srst"
  -c "init"
  -c "reset halt"
  -c "flash erase_sector 0 0 last"
  // if .elf image type
  -c "flash write_image <FilePath>"
  // or if not and .elf image type
  -c "flash write_image <FilePath> 0x08000000 bin"
  -c "reset run"
  -c "shutdown"
```

### 5.2.4 Retrieving custom sensor code

To generate custom sensor code, you must click on the "Build Firmware" button from the Device Toolbar and then on "Next" button after selecting the appropriate options.

Clicking on the "Next" button will actually generate the C files and Makefile required for building. The build process will start immediately and generate a firmware for Nucleo.
Refer to § for tweaking the standard buildsystem for directly generating a complete firmware for your MCU/board from SensorStudio.
After clicking on the "Next" button, you can then retrieve the generated C code in:

- *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/build/<project-name>*

You must include those files into your buildsystem to generate a complete firmware including your custom sensors. The generated Makefile will also contain some important definitions to be included in your buildsystem too.

- **Important:** After the build process is complete, do not click on the 'Flash' button or it will attempt to flash and connect to your device. Close the window instead and flash your firmware by an external tool (see § available in:
  - *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/doc/*

Flashing a custom firmware image from SensorStudio).

### 5.2.5 Building and debugging a GSH port from SensorStudio

SensorStudio relies on an external Makefile and Eclipse project template to compile the GSH firmware and generated eclipse project and debug configuration.
By updating those templates, it is possible to build GSH firmware for another MCU/board from SensorStudio.
Templates used from SensorStudio are available in:

- *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/config/linaro-cm4-nucleo/eclipse/template*

Templates consist in "*.in*" files containing strings between @. Those strings (aka variables) are automatically replaced by SensorStudio just before the build phase.
Project files are generated into:

- *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/build/<project-name>*

The *GNU make* utility used to process the Makefile is located under:

- *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/tools/mingw32-make.exe*

Refer to **Table 2 Project template files**, **Table 3 Template variables description** and **Table 4 Build environnent** for more information about each template files, variables and environment information:

**Table 2 Project template files**

| File | Description |
|---|---|
| *Makefile.in* | Entry Makefile for building the firmware.<br>In will include the main Makefile for building GSH. |
| *GshExtConfig.h.in* | Extra configuration header files for the firmware<br>Included in GshConfig.h if GSH_EXTERNAL_CONFIG_FILE preprocessor macro is defined |
| *.cproject.in*<br>*.project.in* | Template file for Eclipse project |
| *Debug-_PROJECT_NAME_-(no-load).launch.in* | Template file for Eclipse debug configuration |
| *.settings/org.eclipse.cdt.core.prefs* | This file is directly generated from SensorStudio and does not derived from any template. It contains some environment information for Eclipse. |

**Table 3 Template variables description**

| Variable | Description |
|---|---|
| `_PROJECT_NAME_` | The project name as defined by the name of the SensorStudio flow or the "*Firmware name*" field in the "*Build*" window. |
| `_PROJECT_OUTPUT_DIR_` | Where resulting binary is generated. Currently expanded to *bin* |
| `_PROJECT_VERSION_` | Expanded to value set in "Version suffix" field from the in the "*Build*" window. |
| `_PROJECT_DEBUG_` | Expanded to "yes" or "no" depending if "*Debuggable firmware*" option is checked in the "*Build*" window. |
| `_PROJECT_EXTRA_FLAGS_` | Currently expanded to an empty string |
| `_PROJECT_EXTRA_IDIRS_` | Extra include directories to search for header passed to the compiler as *-I* flag. |

| | |
|---|---|
| | Currently expanded to few extra paths pointing to internal GSH sources as well as the current project sources.<br>Path specified in *CustomSensorDesigner* block *IncludeDir* and *SourceDir* properties are also added. |
| `_PROJECT_EXTRA_DEPS_` | Extra dependencies to check for in order to trigger a full re-build of the firmware.<br>Expanded to generated Makefile and FW configuration file. |
| `_PROJECT_EXTRA_CSRCS_` | Extra sources to compile and link.<br>Expanded to generated custom sensors sources and *entry.c* file.<br>Path specified in *CustomSensorDesigner* block *SourceDir* are also added (if .c suffix). |
| `_PROJECT_EXTRA_DEFS_` | Extra preprocessor definitions passed to the compiler as *-D* flag.<br>Expanded to *-DGSH_EXTERNAL_CONFIG_FILE=\"GshExtConfig.h\"* in order to indicate the extra configuration header file to be included when building the firmware. |
| `_PROJECT_EXTRA_LIBS_` | Extra static libraries to link against, passed as *-l* flag to the linker.<br>Expanded to the value specified in *CustomSensorDesigner* block *SourceDir* property (if *.lib* or *.a* suffix) |
| `_PROJECT_EXTRA_LDIRS_` | Extra directories to look for static libraries, passed as *-L* flag to the linker.<br>Expanded to few extra paths pointing to internal GSH sources as well as the current project sources.<br>Path specified in *CustomSensorDesigner* block *IncludeDir* and *SourceDir* properties are also added. |
| `GSH_FRONTEND_DECIMATOR` | Expanded to *SIMPLE* or *NONE* according to firmware option "*Decimate samples to be as close to the requested frequency as possible*" |

The table below describes the environment variables exported by SensorStudio upon Makefile invocation or added to Eclipse *.settings/org.eclipse.cdt.core.prefs* file.

**Table 4 Build environnent**

| Environment variable | Value | Description |
|---|---|---|
| `SHELL` | cmd | To properly parse Makefile if bash.exe is present in user PATH |
| `INVN_GSH_SDK_PATH` | *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/* | Absolute path to GSH SDK |
| `INVN_TOOLCHAIN_PATH` | *<sensorstudio-install-dir>/firefly/toolchain/linaro/bin* | Path to GNU toolchain used in Makefile to compile the firmware |
| `INVN_OPENOCD_PATH` | *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/openocd/bin/openocd.exe* | Path to OpenOCD executable for flash/debug from Eclipse. |
| `INVN_MAKE_EXE` | *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/tools/mingw32-make.exe* | Path to GNU make executable used to process Makefiles. |

### 5.2.6 Overriding default environment

**Warning:** Overriding the default SensorStudio environment is beyond the nominal usage of SensorStudio and was not fully validated and can have unwanted side effects.
If environment variables described in **Table 4 Build environnent** exists when *SensorStudio.exe* is run, they will take precedence over the default value.

**Table 5 Default envirnoment**

| Environment variable | Default value | Description |
|---|---|---|
| `INVN_DEVICE_GSH_MAKE_PATH` | *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/tools/mingw32-make.exe* | Specify location to GNU make executable to use.<br>Also override `INVN_MAKE_EXE` (see **Table 4 Build environnent**) |
| `INVN_DEVICE_GSH_TOOLCHAIN_PATH` | *<sensorstudio-install-dir>/firefly/toolchain/linaro/bin* | Override `INVN_TOOLCHAIN_PATH` value (see **Table 4 Build environnent**). |
| `INVN_DEVICE_GSH_OPENOCD_PATH` | *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>/openocd/bin/openocd.exe* | Specify location to OpenOCD executable to use.<br>Also override `INVN_OPENOCD_PATH` (see **Table 4 Build environnent**) |

| | | |
|---|---|---|
| INVN_DEVICE_GSH_SDK_DIR | *<sensorstudio-install-dir>/firefly/devices/gsh/<gsh-plugin-version>* | Specify location of entire GSH SDK.<br>If you plan on modify the SDK and GSH sources, this variable can be used to indicate the new SDK and let the existing one untouched. |

# 6 TROUBLESHOOT

## 6.1. ERROR WHEN FLASHING GENERIC SENSOR HUB FIRMWARE

If you get an error when flashing Generic Sensor Hub firmware, just try again a second time.
If the error persists, try to physically disconnect/reconnect your device from the computer and flash your firmware again.

## 6.2. LAGS IN CURVES WHEN MANY SENSORS ARE ENABLED WHEN USING GENERIC SENSOR HUB

If you observe lags in curves display when using Generic Sensor Hub and enabling severals sensors at a high rate, you can workaround this issue by reducing the latency of the FTDI device when using InvenSense Nucleo carrier board.
To do so:

- Go to the "Device Manager" of your computer
- In *"Ports (COM & LPT)"* group, select the "*USB Serial Port*" corresponding to the carrier board
- Right click and open the *"Properties"* window
- On the *"Port Settings"* tab, click *"Advanced"* button
- Change the *"Latency Timer (ms)"* option to **1**

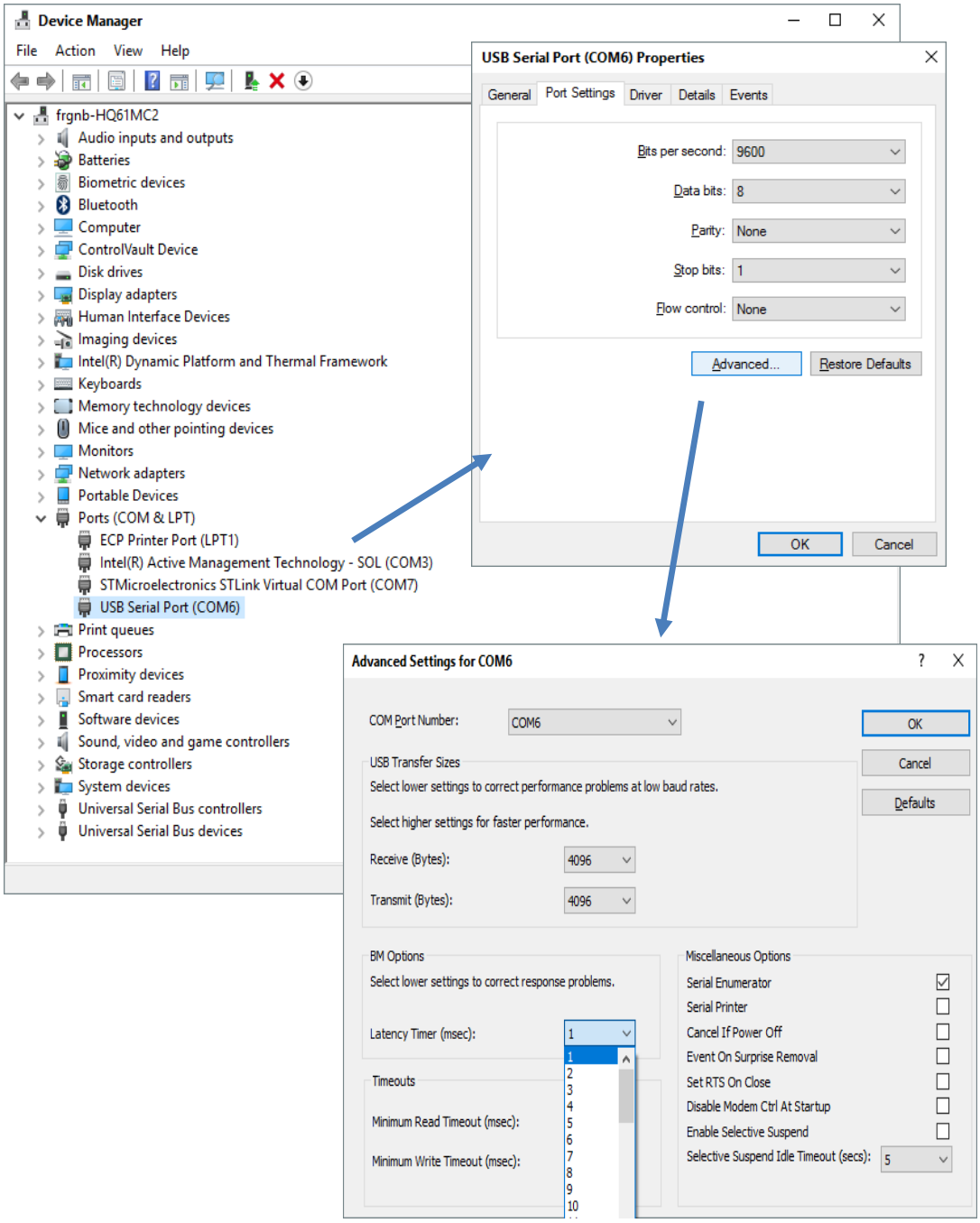



**Figure 73. Reducing the latency of the FTDI device**

# 7 REVISION HISTORY

| REVISION DATE | REVISION | DESCRIPTION |
|---|---|---|
| 10/28/2015 | 1.0 | Initial Release |
| 02/22/2016 | 1.1 | Update for SensorStudio 2.1.0<br>Updated sections *4.2 FireFly* & *3.3 Connecting FireFly* |
| 05/19/2016 | 1.2 | Additional updates for SensorStudio 2.1.0 |
| 10/11/2016 | 1.3 | Update for SensorStudio 2.2.0 |