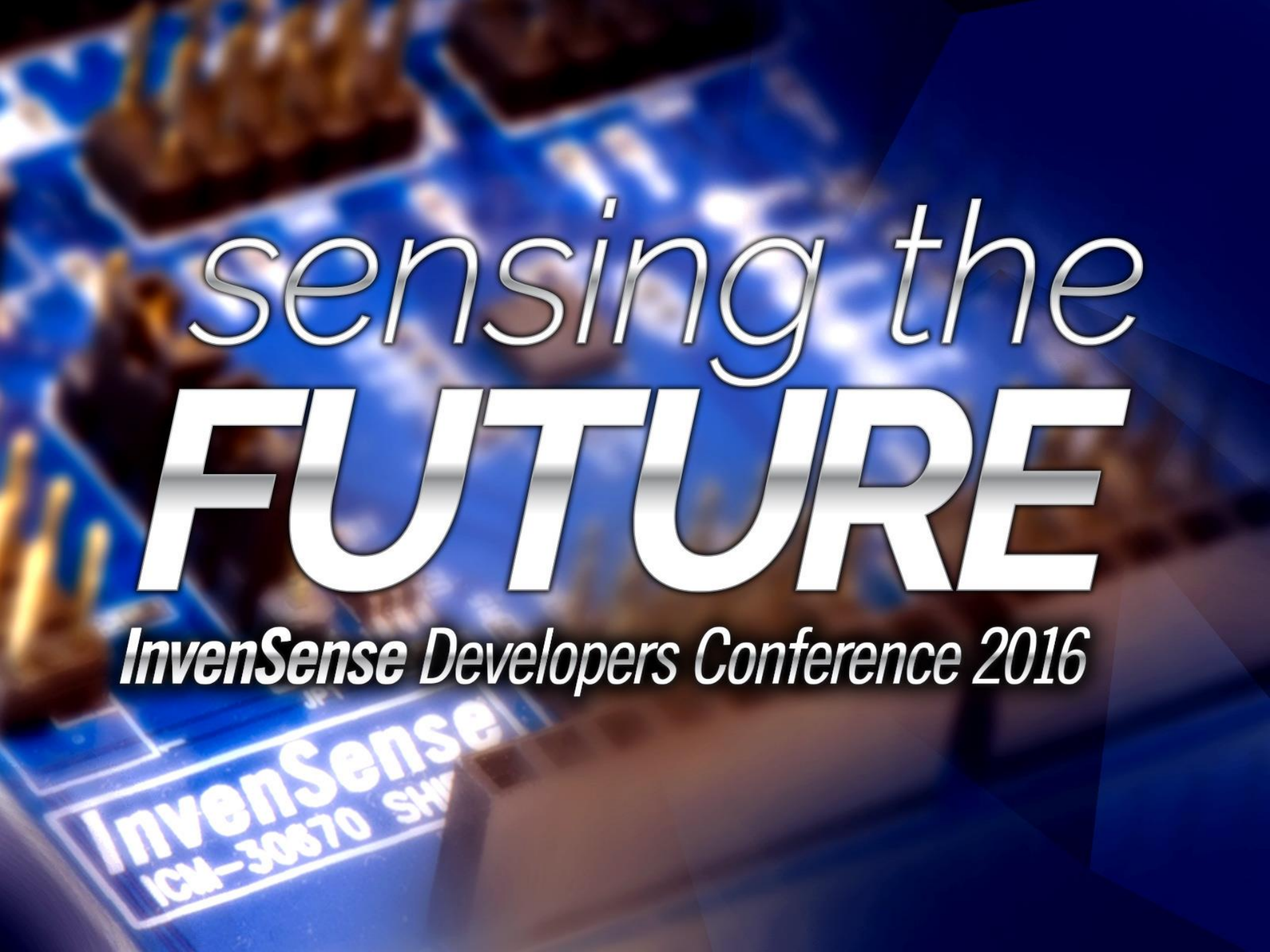

sensing the
FUTURE
InvenSense Developers Conference 2016
InvenSense
ICM-30670

SensorStudio

Deep Dive

Agenda

- SensorStudio Introduction
- Sensor Hub Introduction
- Understand SensorStudio Samples
- Design your Sensor fusion
- Design your Sensor driver
- Use your CustomSensor
- Make SensorStudio work with your Product

New ICM-20690 Development Kit

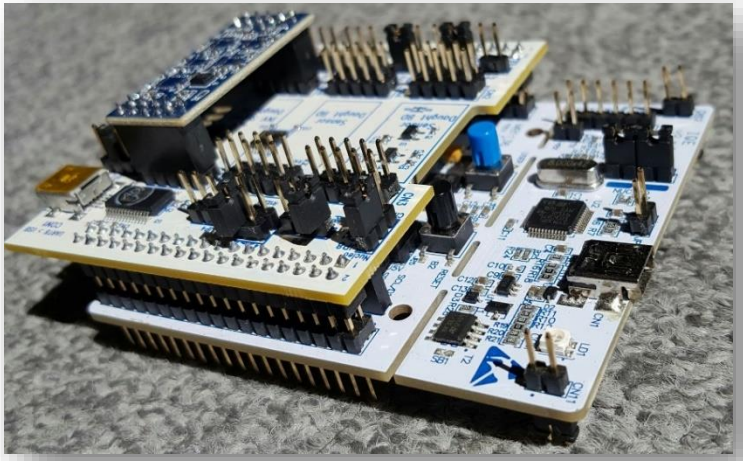
sensing the FUTURE

Download SensorStudio
from InvenSense Developers Corner

Get ICM20690 DK + Sensor Daughter Board
from InvenSense GenericSensorHub

Get a NUCLEO-F411RE board
from STMicroelectronics

Download Eclipse IDE for C/C++ Developers
from Eclipse.org downloads

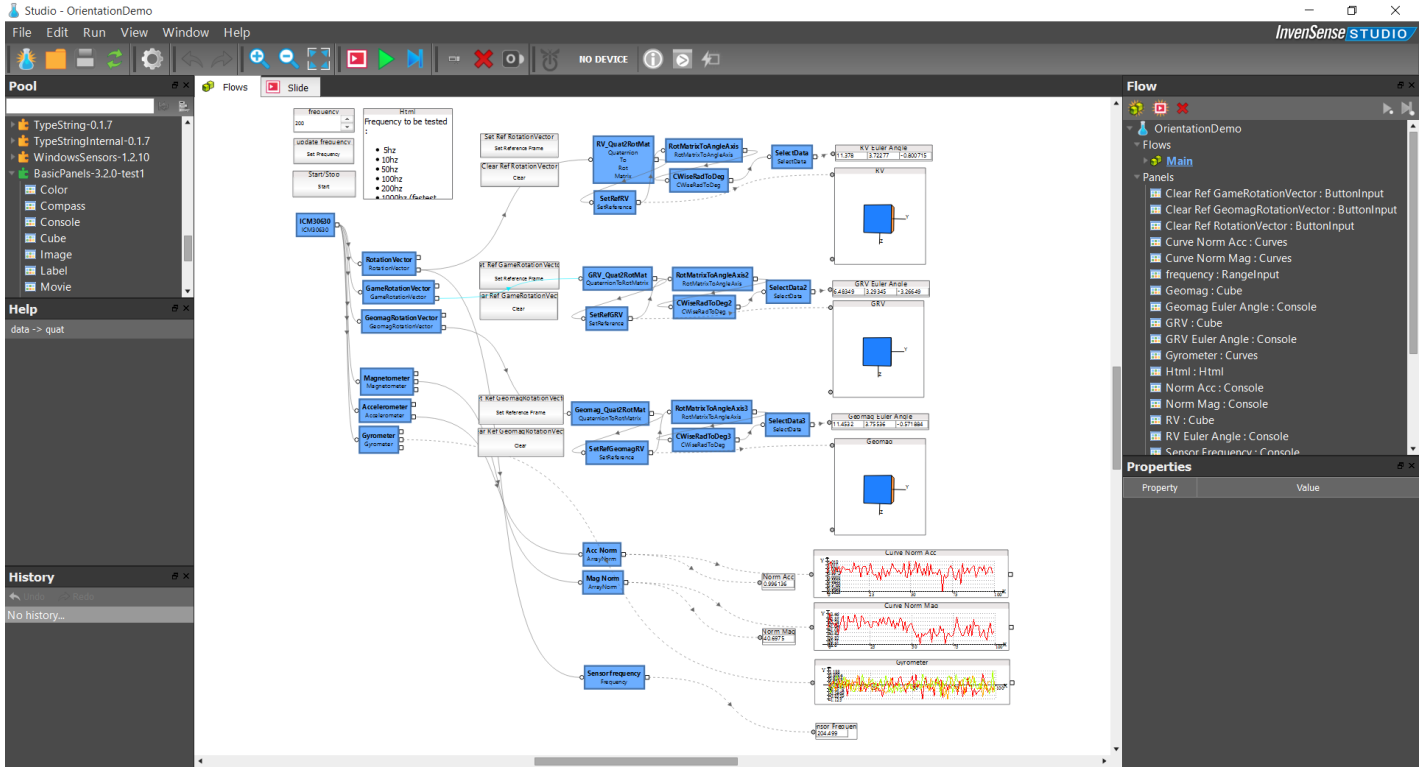


This presentation will use
SensorStudio ICM-20690 Development Kit

SensorStudio Introduction

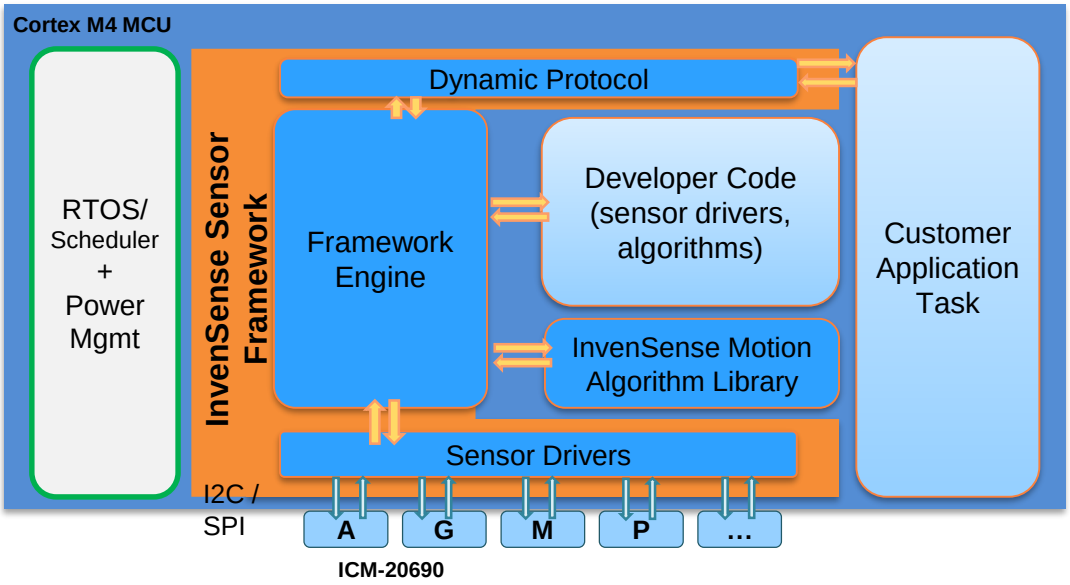
sensing the FUTURE

- Graphical User Interface
- View into GenericSensorHub
- Extend GSH with your sensors & algorithms!



GenricSensorHub Introduction

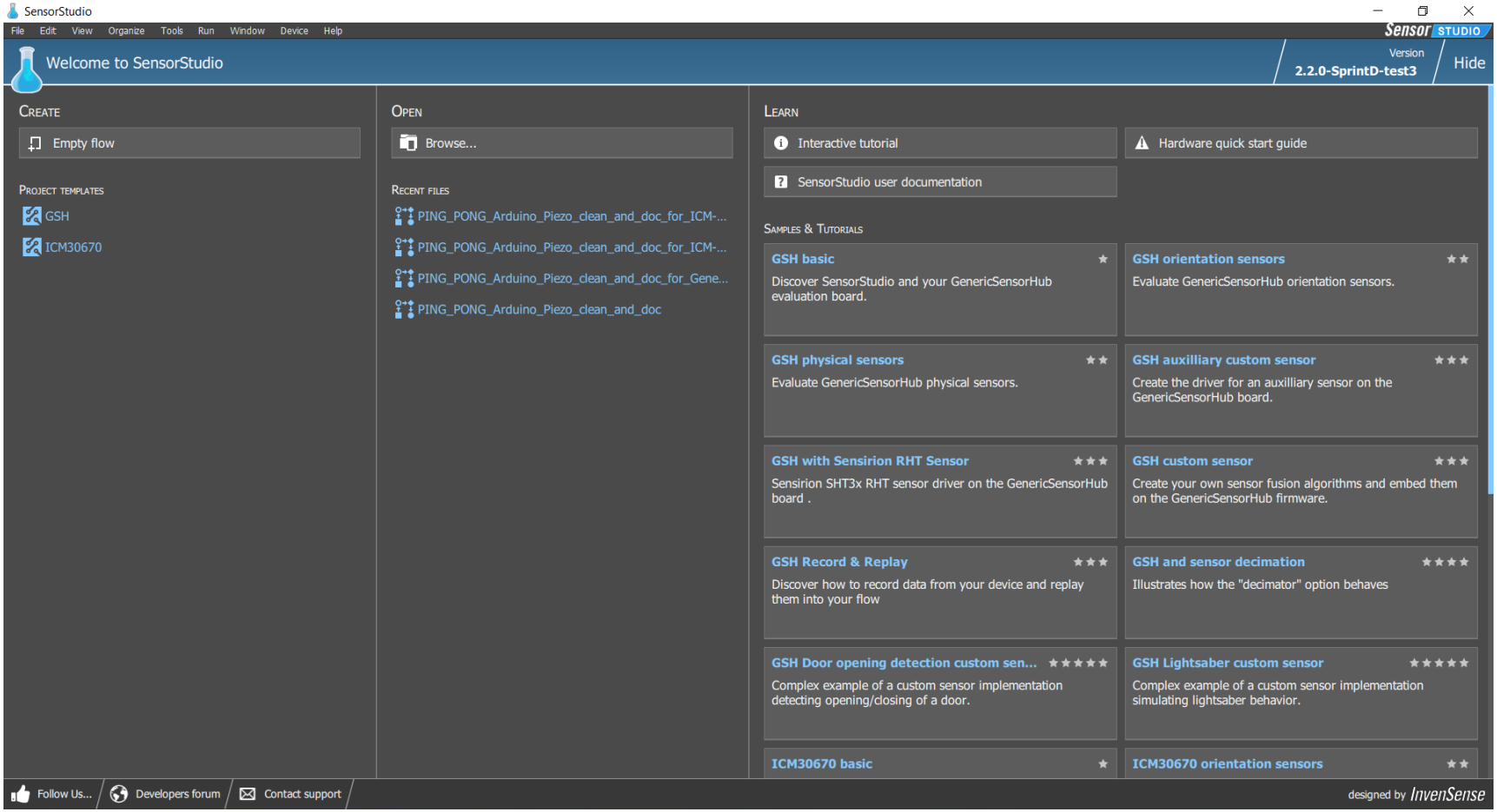
- GenricSensorHub and ICM-20690
- Includes InvenSense Motion Algorithms
- Sensor Framework designed for extensibility
  - Add new algorithms
  - Add new sensor drivers



Feature samples

sensing the FUTURE

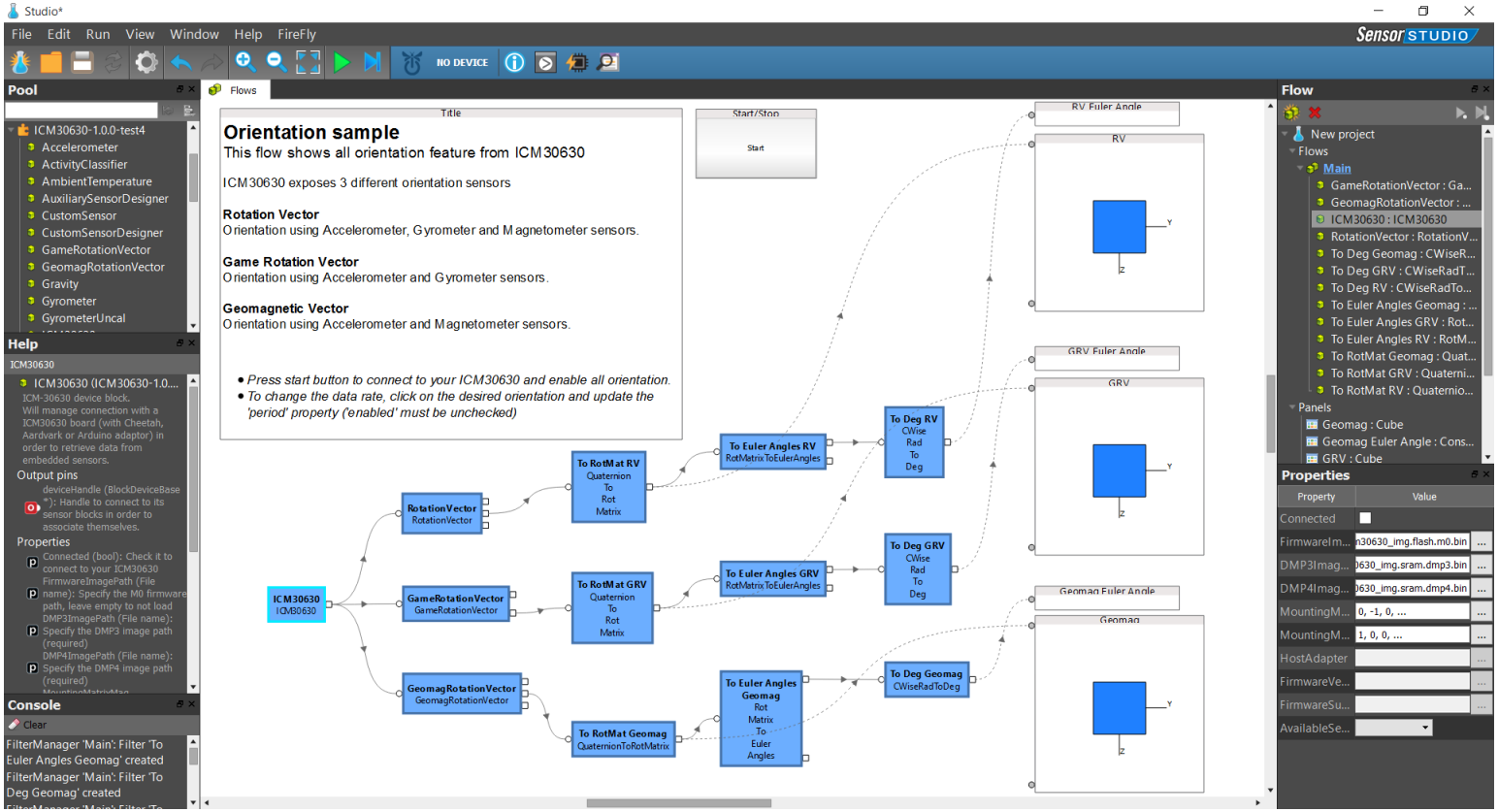
- Welcome screen
  - guide you to select the appropriate feature samples



Orientation sample

sensing the FUTURE

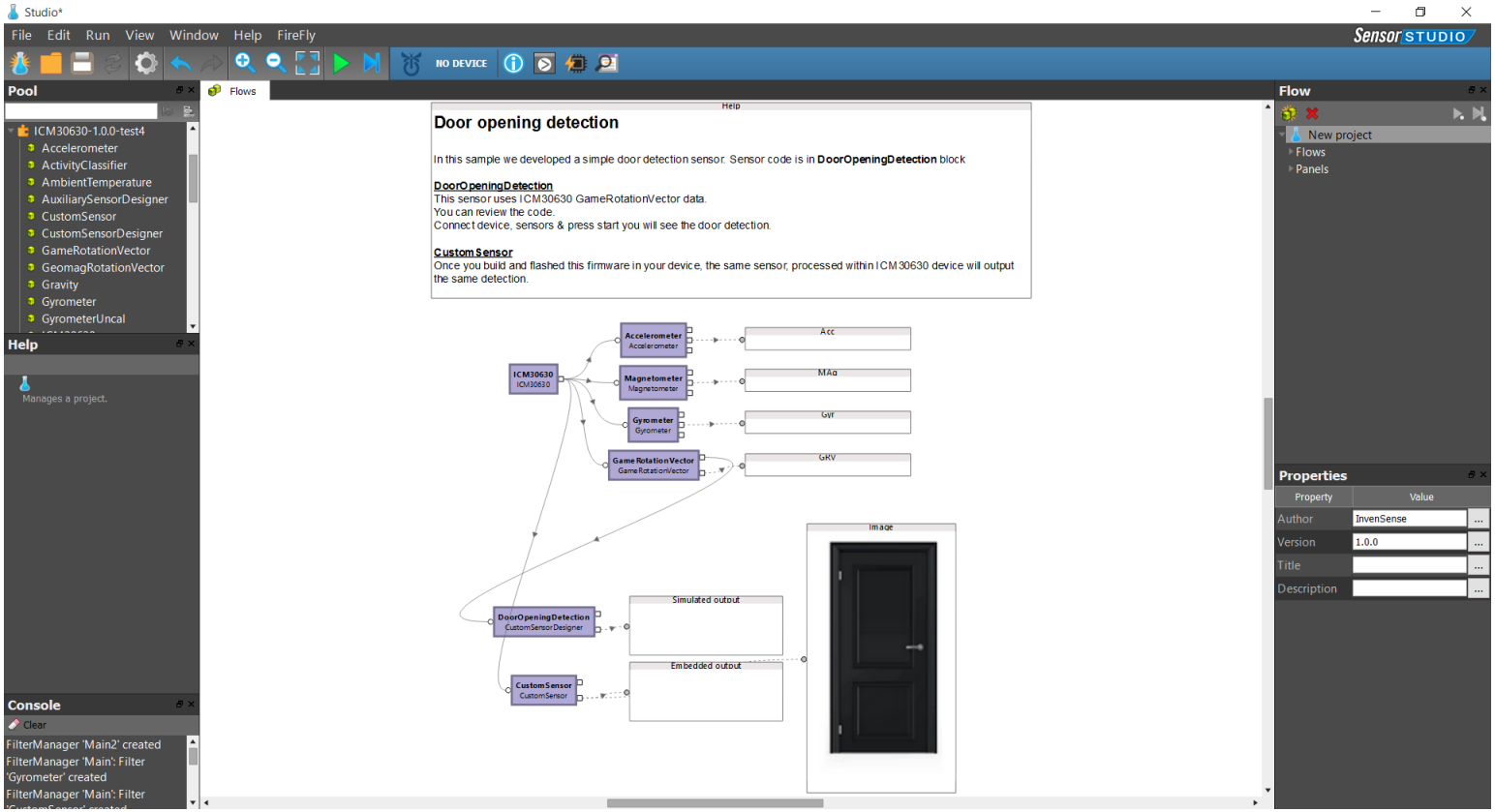
- Concrete use of RV, GRV, GeomagRV features
- SensorStudio runs conversion & visualization



Door Opening Detection ample

sensing the FUTURE

- Concrete use of GRV & CustomSensor features
- SensorStudio runs CustomSensor & visualization

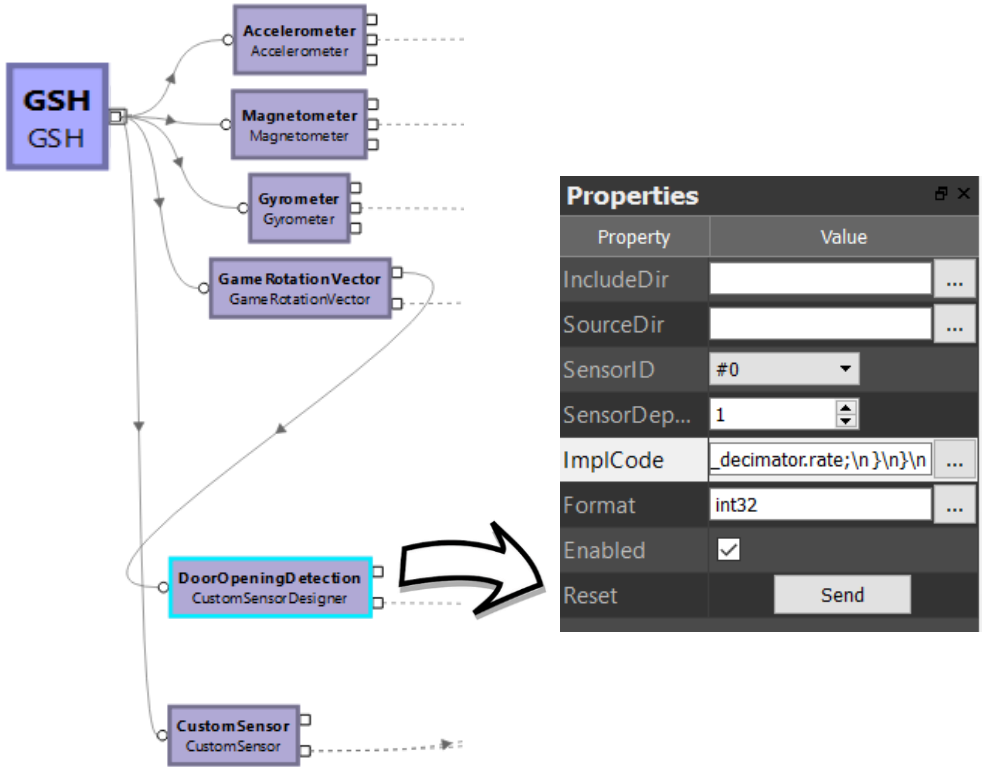


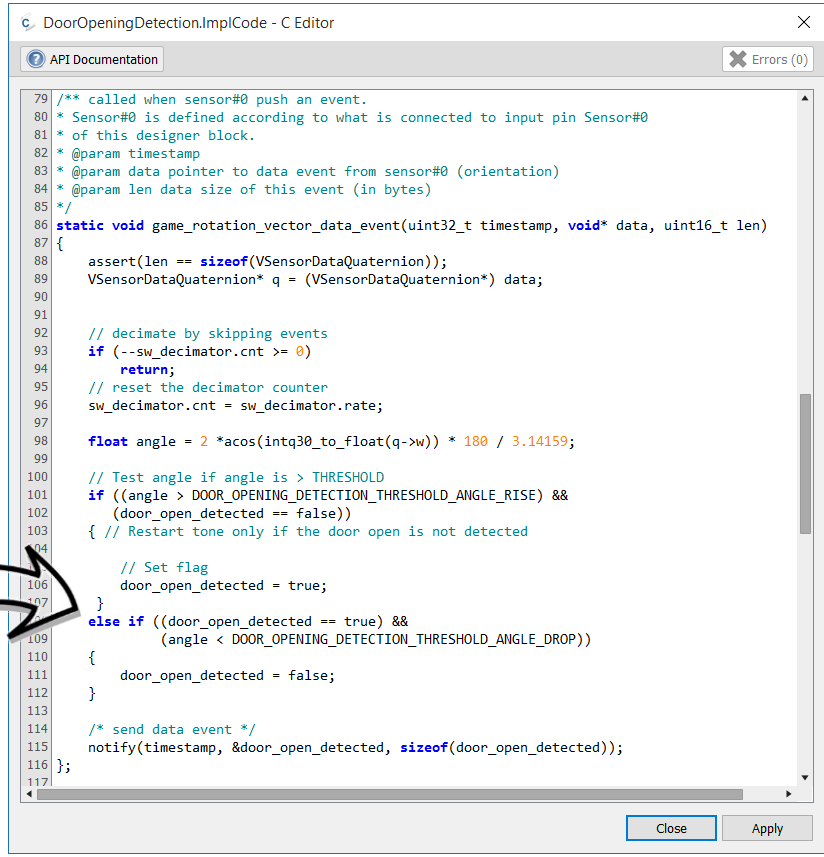
Design your sensor fusion – c

itor

sensing the FUTURE

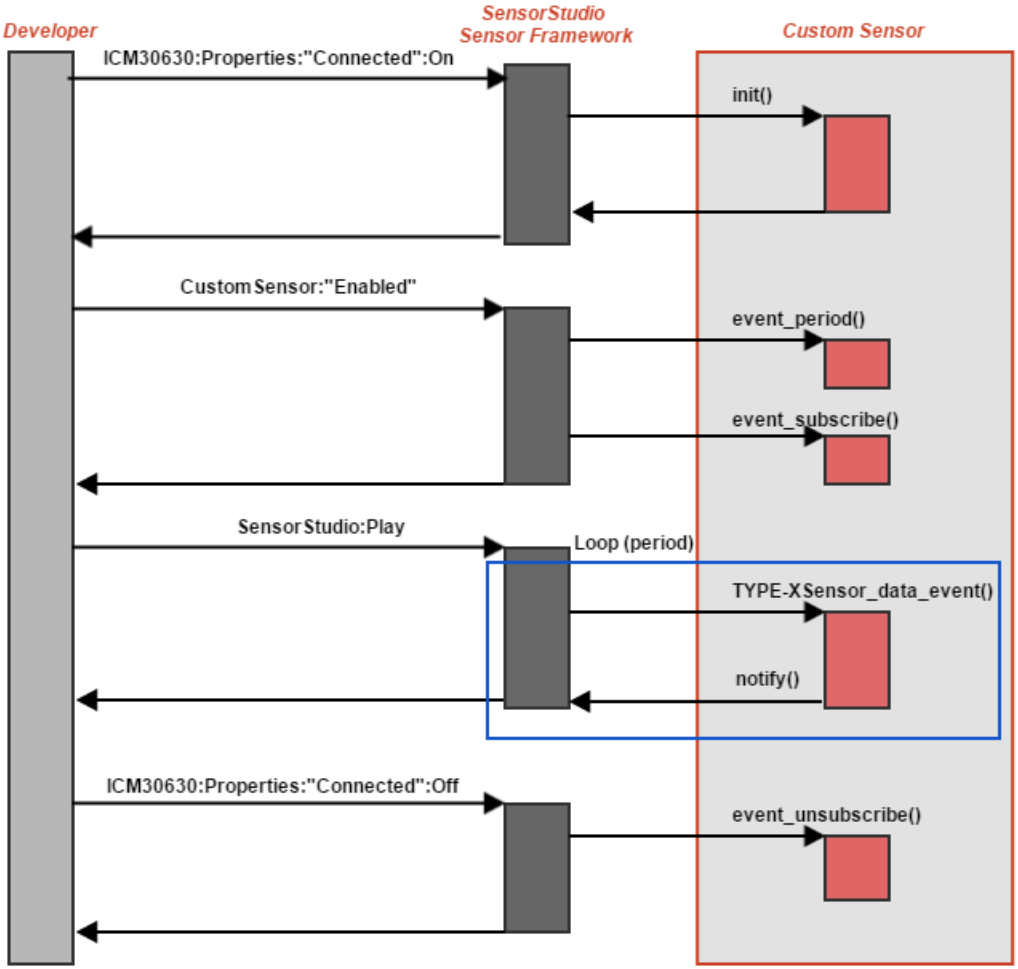
- CustomSensorDesigner skeleton code as a guide
- CustomSensorDesigner code runs on Desktop

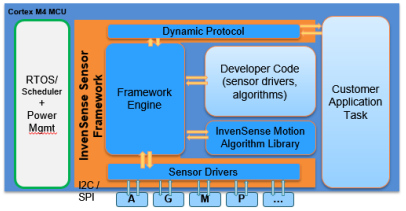




Sensor Framework integration

- **notify** & **subscribe** pattern to consume/produce sensor data

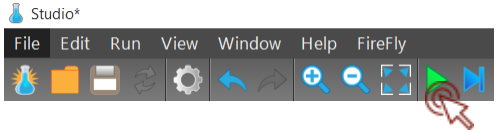


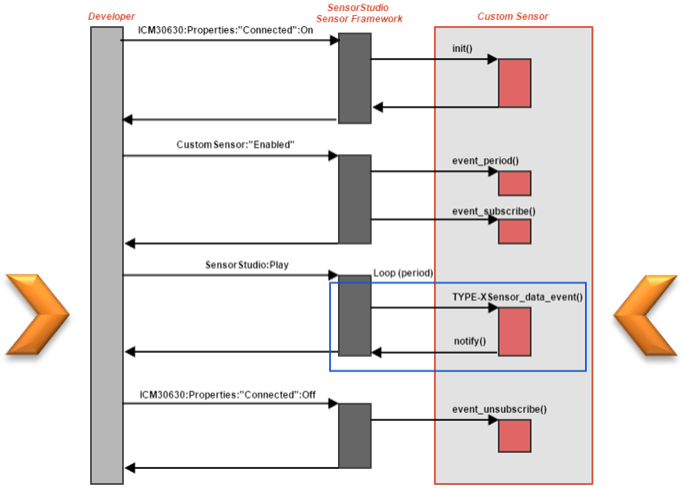


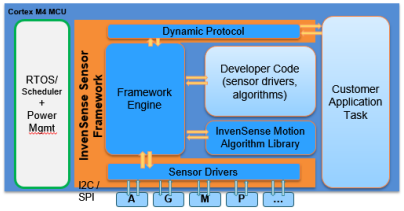
Sensor Framework Integration

- *TYPE-XSensor*_**data_event()**
  - Called upon receiving a new sensor data of a given type
  - You will have as many *TYPE-XSensor*_**data_event**() functions to implement as you have subscribed sensors
  - This is the place to store and process your data
  - Replace *TYPE-XSensor*_ by other sensor data type
  - When you are done with your processing, use the **notify()** function to pass on your produced data.

Click « Play »



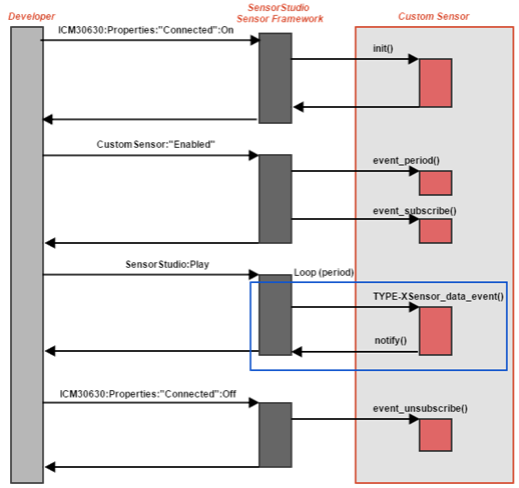




DoorOpening sample - data_event()

sensing the FUTURE

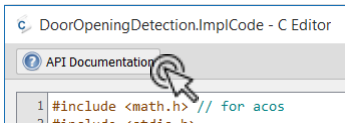
```
static void game_rotation_vector_data_event(uint32_t timestamp, void* data, uint16_t len)
{
  assert(len == sizeof(VSensorDataQuaternion));
  VSensorDataQuaternion* q = (VSensorDataQuaternion*) data;
  float angle = 2 *acos(intq30_to_float(q->w)) * 180 / 3.14159;

  // Test angle if angle is > THRESHOLD
  if((angle > DOOR_OPENING_DETECTION_THRESHOLD_ANGLE_RISE) &&
    (door_open_detected == false))
  {
    // Set flag
    door_open_detected = true;
  }
  else if ((door_open_detected == true) &&
           (angle < DOOR_OPENING_DETECTION_THRESHOLD_ANGLE_DROP))
  {
    door_open_detected = false;
  }

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



Sensor Framework Doc

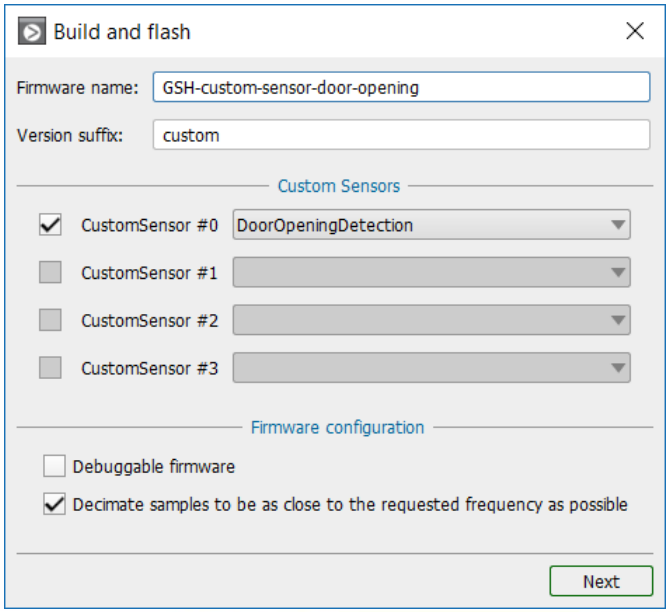
- C Editor shortcut to Sensor Framework API
- Complete list of
  - Sensor data
  - Sensor functions

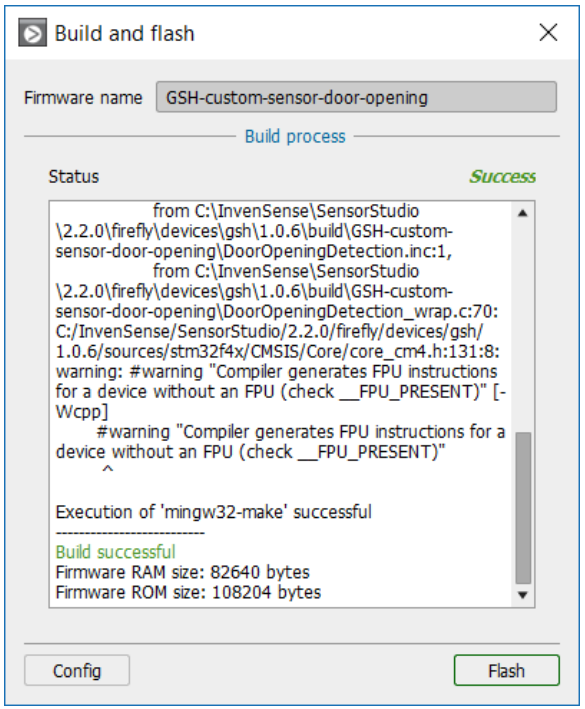
available



Design your sensor fusion

- Select your features, assign CustomSensor #IDs
- Cross-build your code for Nucleo
- Flash your Firmware



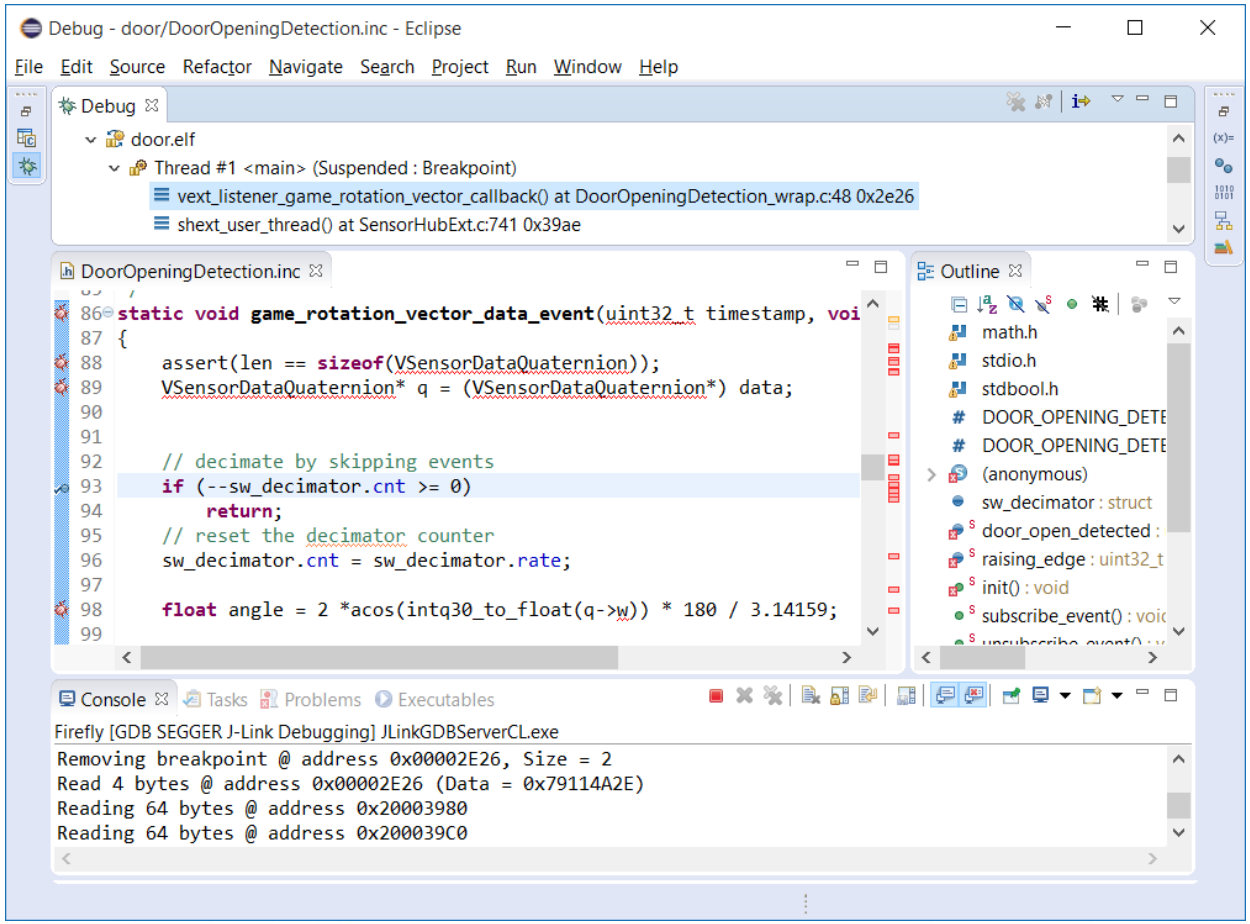


Running your sensor fusion

- CustomSensor runs embedded
  - Firmware loaded, GenericSensorHub Connected
  - GRV & CustomSensor Enabled

Debug your code in Eclipse

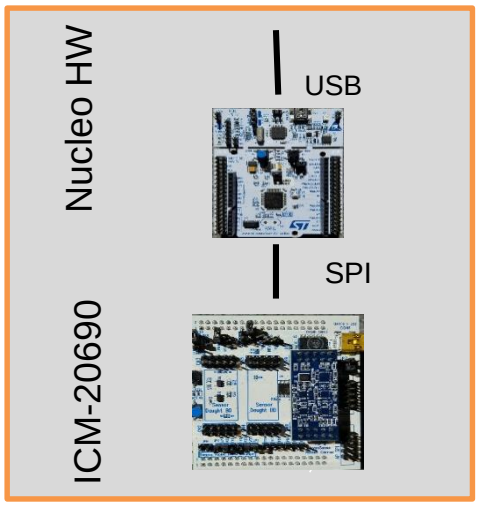
- Launch Eclipse project
- Set your breakpoint & Step in your code



Migrating to a Full Prototype

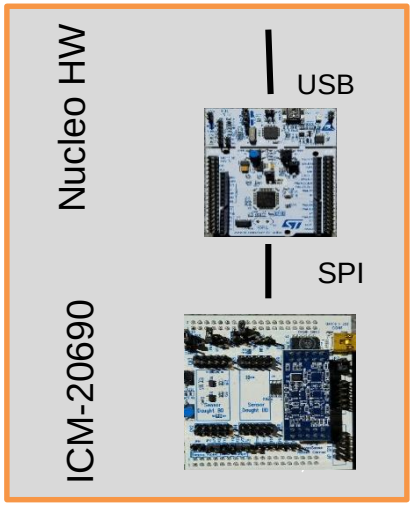
sensing the FUTURE

Simple and fast migration from Algo/SW development
--> proto/product development



A - Tested and validated project is integrated onto Nucleo via SensorStudio

B - Create new Application code to include rest of the platform needs
i.e . connectivity shield



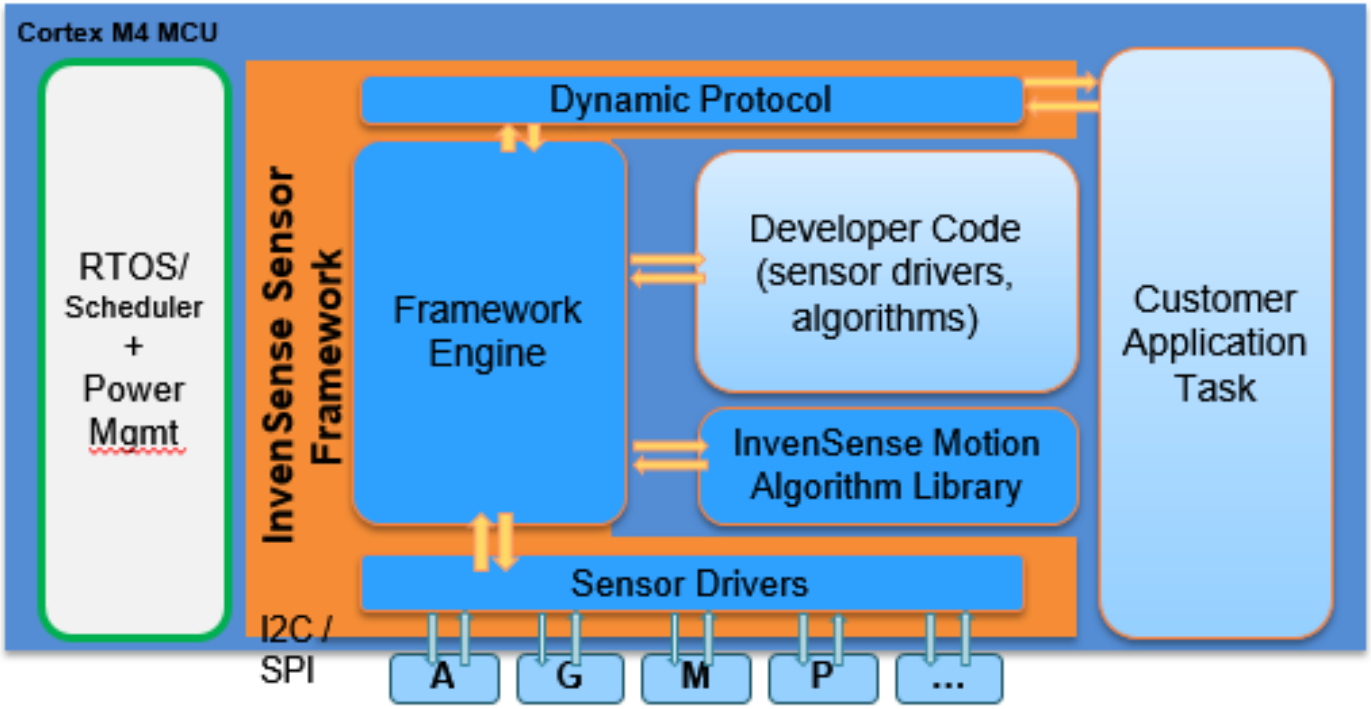
SW Development Environment for GenericSensorHub

APP/HW product development Environment

Build your app in your IDE

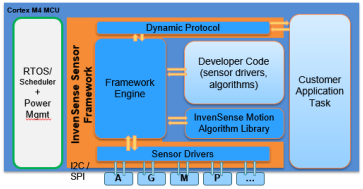
sensing the FUTURE

- Curstomer Application Task to build your app
- Extend the sample code provided
  - get all outputs from GenericSensorHub
- Add your display, wireless, peripherals, …

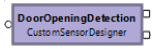


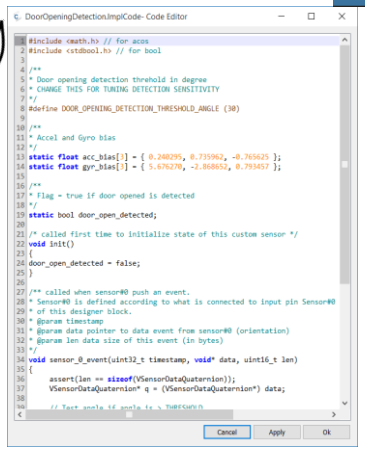
Design your sensor fusion - Summary

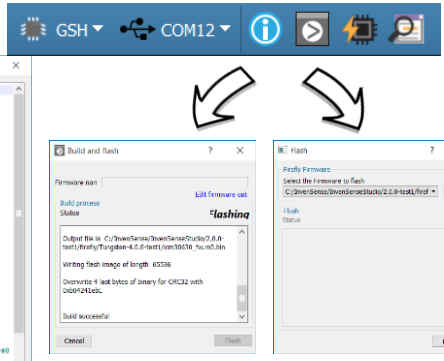
sensing the FUTURE



- Add a CustomSensorDesigner block
- Open CustomSensorDesigner C Editor
- Implement the Sensor Framework interface
  - Skeleton code provided to simplify programming tasks
  - Sample codes are included too
- Select the features you want to use
- Build your code for GSH MCU
- Flash the Nucleo
- Add a CustomSensor block
- Select the CustomSensor ID
- Observe GSH outputs
- Debug with Eclipse
- Build your app



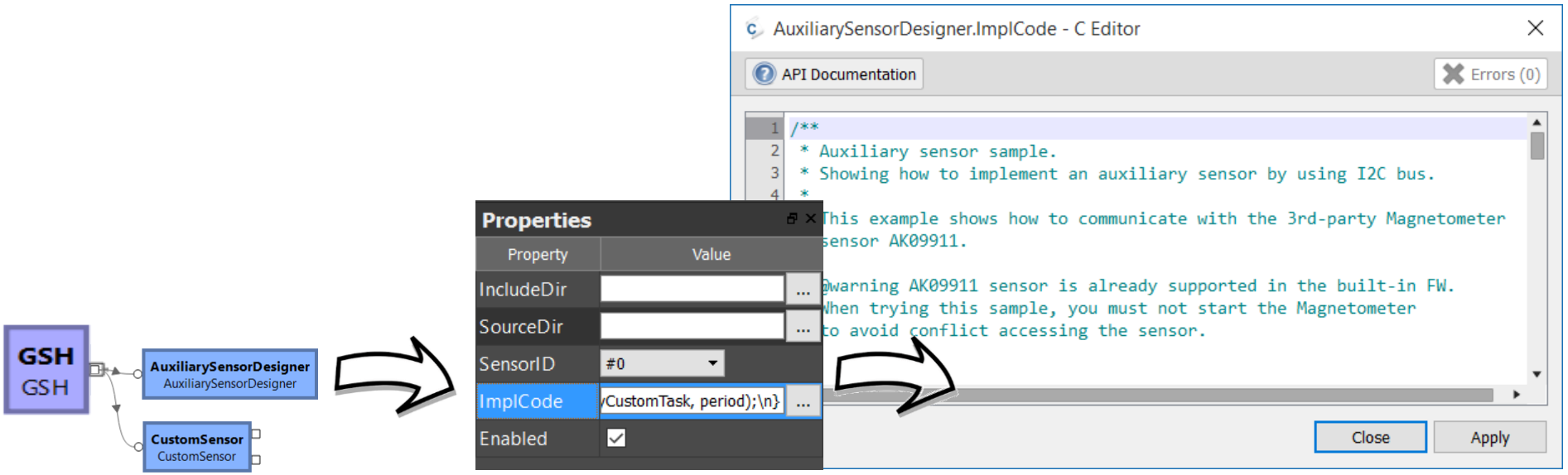




Design your sensor driver – C Editor

sensing the FUTURE

- AuxiliarySensorDesigner skeleton code as a guide
- AuxiliarySensorDesigner runs on HW (only)



Sensor Framework Doc

- C Editor shortcut to Sensor Framework API
- Complete list of functions
  - auxiliary I2C bus
  - system timer
  - tasks

AuxiliarySensorDesigner.ImplCode - C Editor

API Documentation

```
1 /**
2  * Auxiliary sensor sample.
3  * Showing how to implement an auxiliary sensor by using I2C bus.
```

InvenSense Generic Sensor Hub firmware 1.1.0

Documentation of the Generic Sensor Hub firmware running on Nucleo Board

Main Page | Related Pages | Modules | Classes | Files | Search

- Extension framework
  - Auxiliary bus APIs
    - INV_AUX_I2C_NUM_0
    - inv_shext_aux_i2c_num
    - inv_shext_aux_i2c_clo
    - inv_shext_aux_i2c_init
    - inv_shext_aux_i2c_rea
    - inv_shext_aux_i2c_writ
  - Auxialiary GPIO APIs
    - inv_shext_gpio_struct_t
    - INV_GPIO_PIN5
    - inv_shext_gpio_port_t
    - inv_shext_gpio_mode_t
    - inv_shext_gpio_clear
    - inv_shext_gpio_get_sta
    - inv_shext_gpio_init
    - inv_shext_gpio_set_sta
    - inv_shext_gpio_struct_i
    - inv_shext_gpio_toggle
  - Semaphore APIs
  - Task APIs
    - inv_shext_task
    - inv_shext_task_t

```
int inv_shext_aux_i2c_read_reg ( inv_shext_aux_i2c_num_t aux_i2c,
                                 uint8_t                 addr,
                                 uint8_t                 reg,
                                 uint8_t *               data,
                                 uint16_t                len
                               )
```

Read register over Auxiliary I2C bus.

Reading a register consists in an I2C Write of 1 bytes (register address) followed by an I2C read of N bytes.

Warning

It is no possible to read more than 16 bytes per request

Parameters

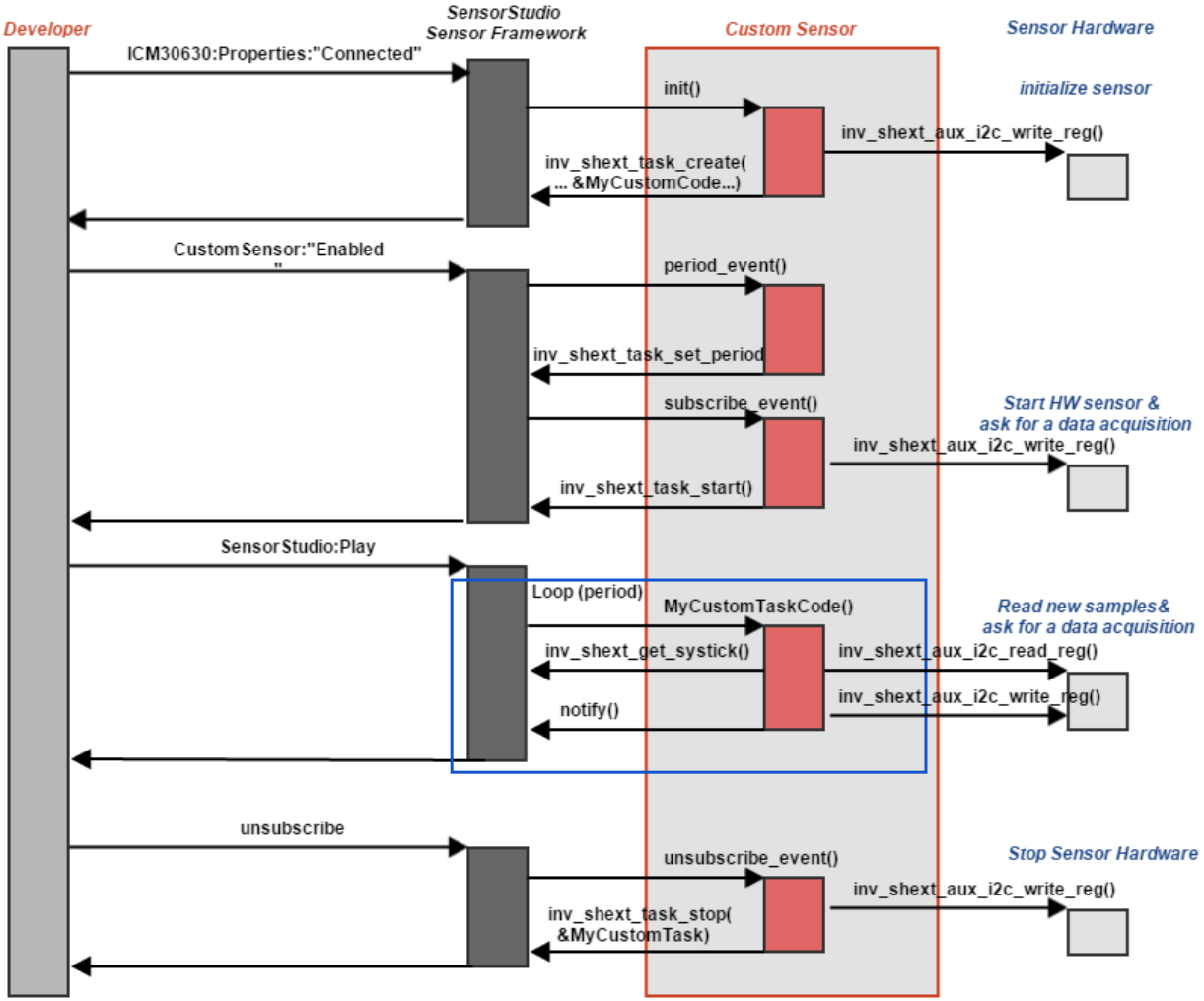
[in]	aux_i2c	I2C auxiliary bus base adress register to use
[in]	addr	I2C peripheral 7bits address
[in]	reg	register to read
[out]	data	pointer to data buffer
[in]	len	number of data to read (must be <= 16)

Generated on Mon Oct 17 2016 14:49:40 for Generic Sensor Hub firmware by doxygen 1.8.6

Sensor Framework integration

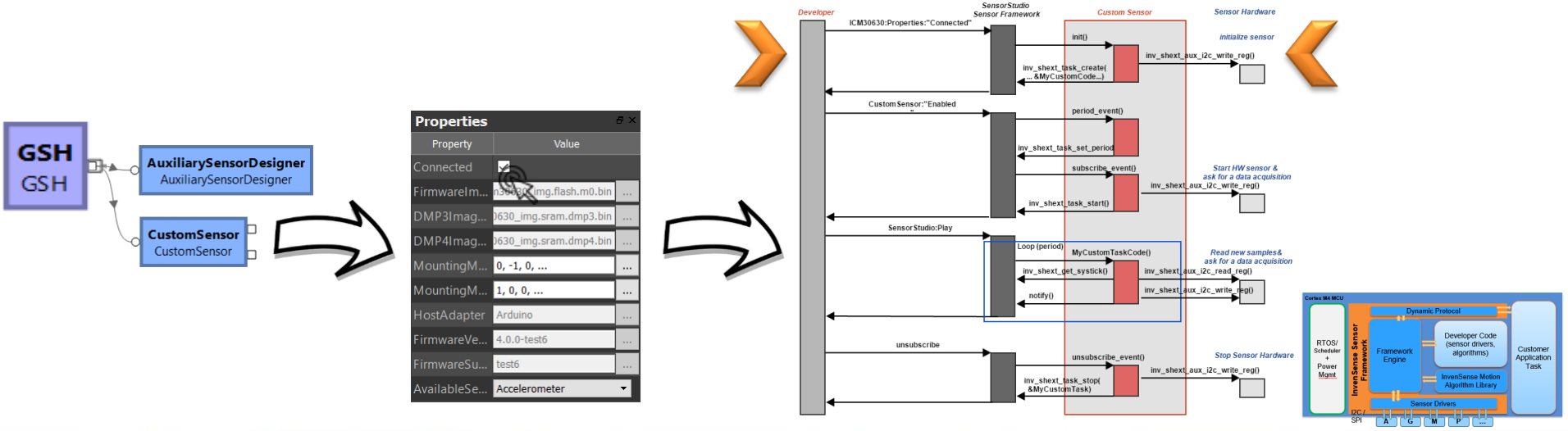
sensing the FUTURE

- **Task & notify** pattern to produce sensor data



Sensor Framework integration

- **init()**
  - Called once, Initialize the CustomSensor state (after the GSH device is powered up, when Sensor Framework is started: GSH "Connected" property)



Sensor Driver - Define a task

- Responsible for polling data from your sensor at the specified Period

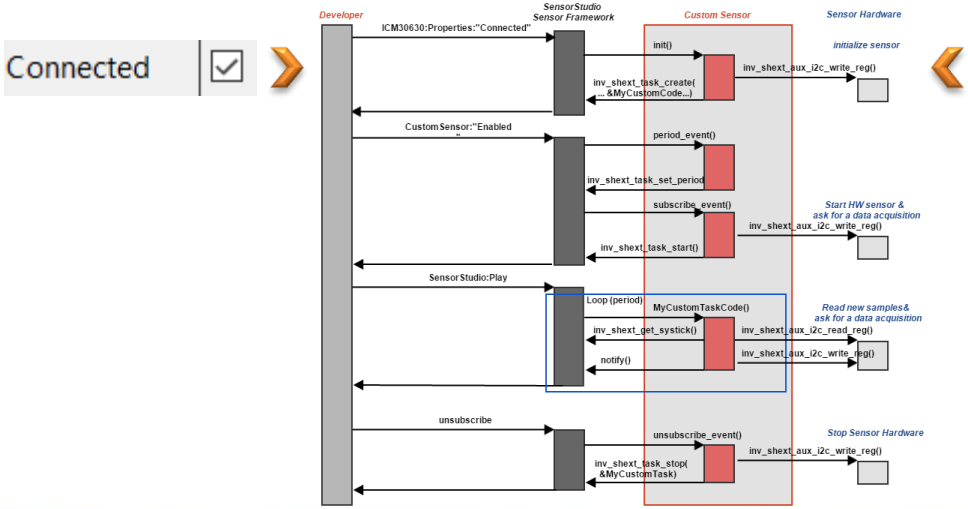
```
/**
* Code for the custom task that will retrieve data from the AK09911
*/
static void MyCustomTaskCode(void * arg)
{
/* get current system time in us */
uint32_t t = inv_shext_get_systick();
int16_t sample[3];

ak09911_get_sample(sample); /* retrieve sensor data */

notify(t, sample, sizeof(sample)); /* notify data to the outside world */

/* The HW sensor performs a single acquisition, so ask for another one */
ak09911_start_single();

(void)arg; /* arg contains the value passed on inv_shext_task_create() */
}
```

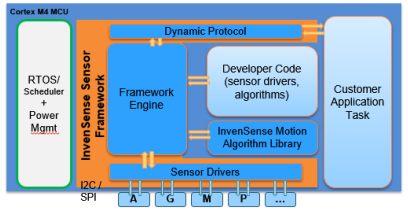
Note : the notify is here, you don't have to implement a data_event(), this is the difference with an algorithm. Sensor Driver is only producing data while an algorithm consumes data from depending sensors & produces data.

AuxiliarySensorDesigner - init()

- **Create your task & initialize I2C at init()**

```
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;
}
```

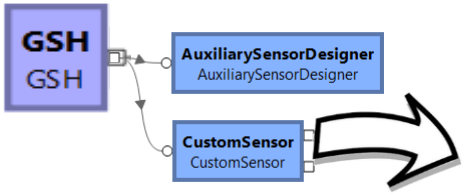


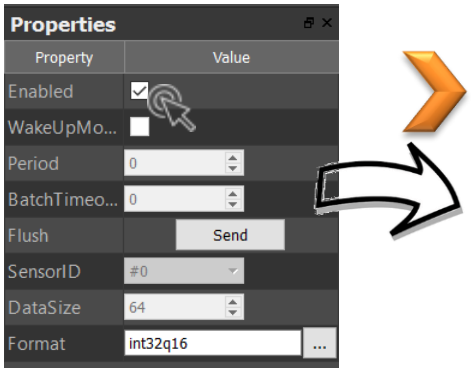


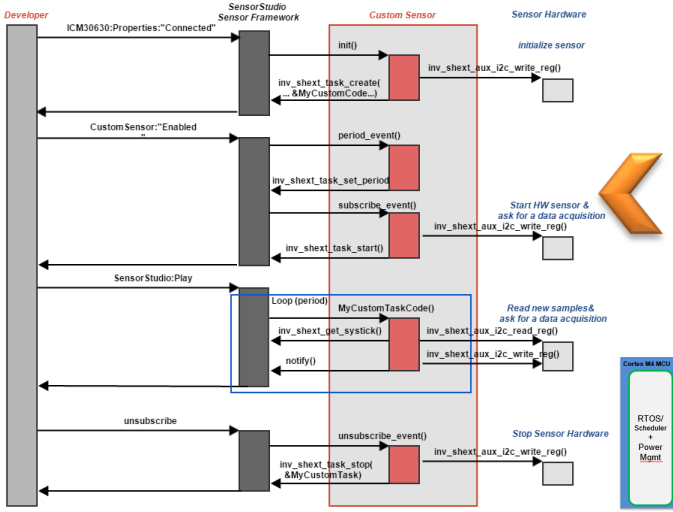
Sensor Framework integration

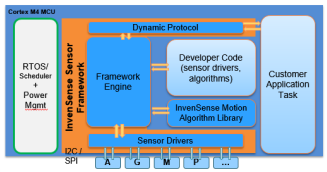
- **period_event()**
  - Called just before your custom sensor is started.
  - Allows to configure the frequency of your CustomTask
- **subscribe_event()**
  - Allows to start your CustomTask

Click CustomSensor:
«Enabled»









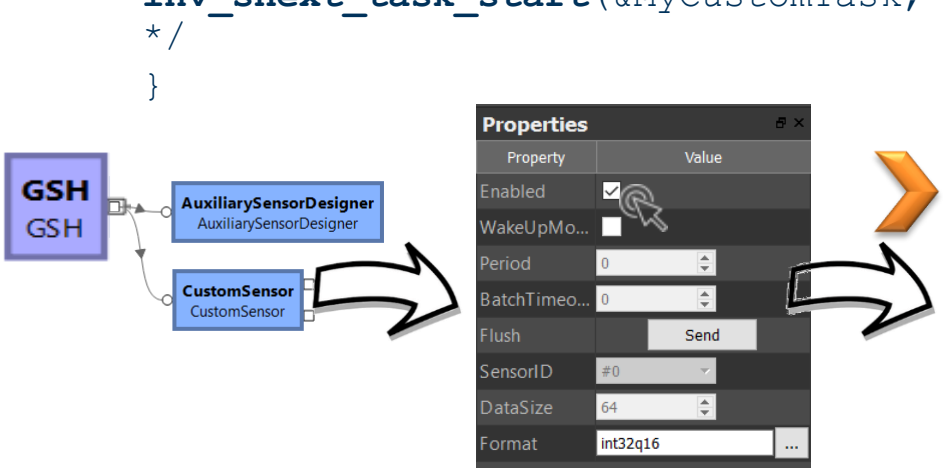
AuxiliarySensorDesigner – period_event()

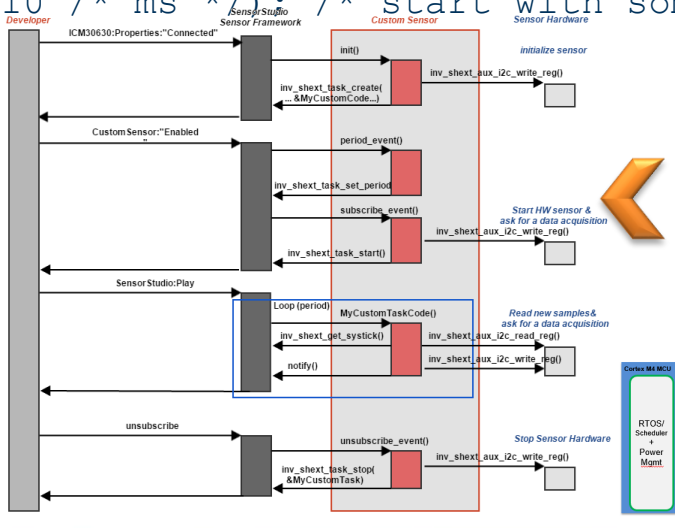
- **Set your task period at period_event()**

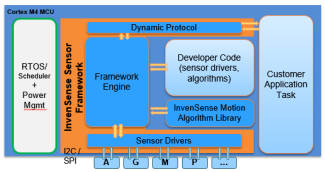
```
static void period_event(uint32_t period) {
/* In this example we update the period of our task */
inv_shext_task_set_period(&MyCustomTask, period);
}
```

- **Start your task at subscribe_event()**

```
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
*/
}
```

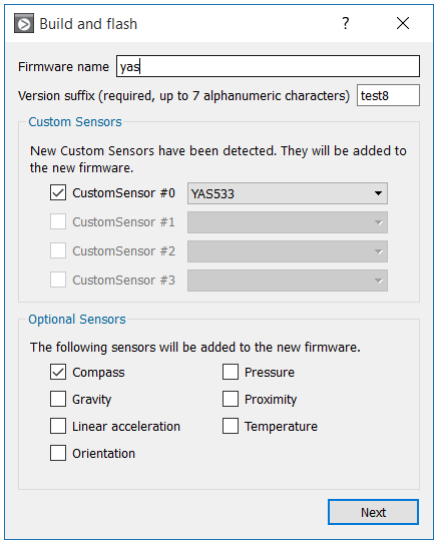


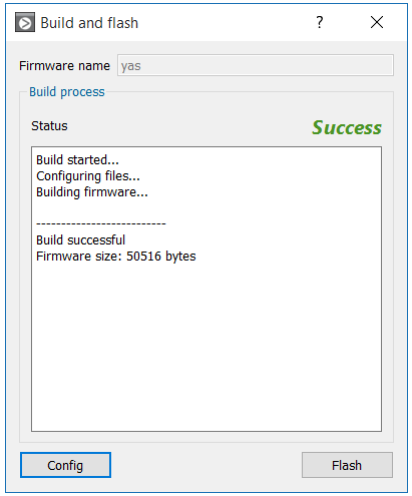


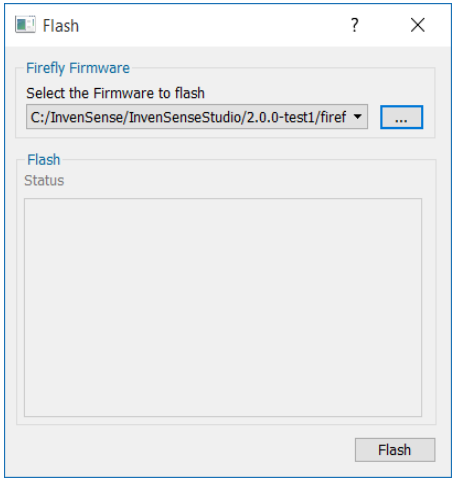


Design your Sensor Driver

- Select your features, assign CustomSensor #IDs
- Cross-build your code for Nucleo
- Flash your Firmware

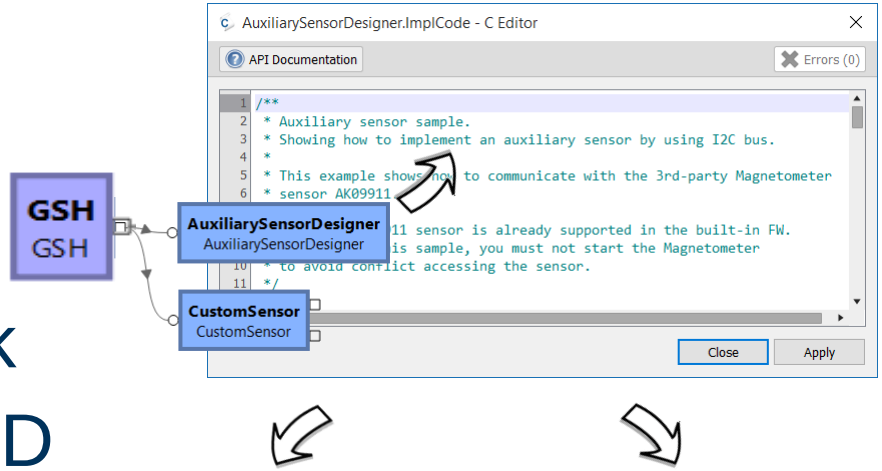






Design your Sensor Driver - Summary

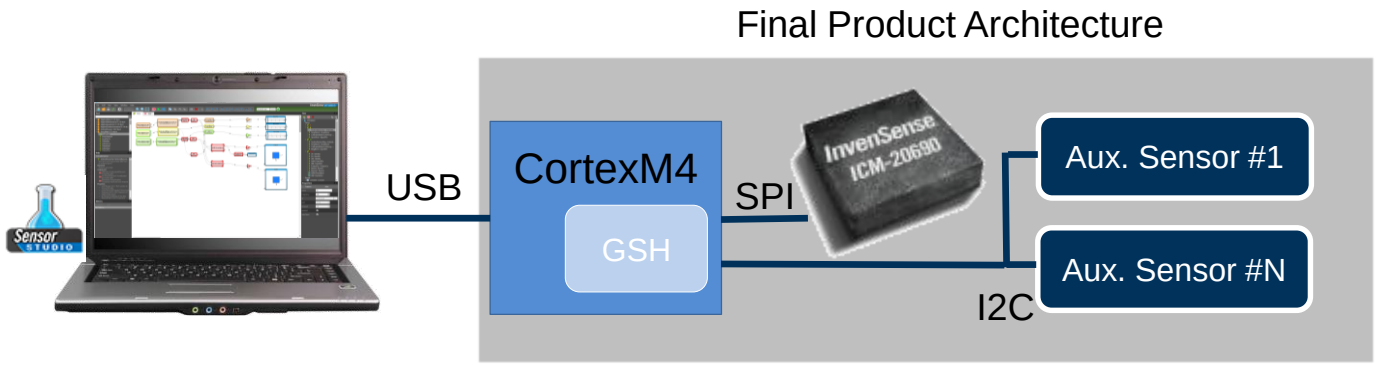
- Add an AuxiliarySensorDesigner block
- Open C Editor
- Implement the Sensor Framework interface
  - Sample code is provided to simplify programming tasks
- Connect it to GSH block
- Build your code for GSH
- Flash the Nucleo
- Add a CustomSensor block
- Select the CustomSensor ID
- Observer GSH outputs



Working with your final Product

sensing the FUTURE

- SensorStudio can work with your final product too
- “DynamicProtocol” code connect SensorStudio & GenericSensorHub
- “DynamicProtocol” source code is available for you to modify if needed



Presentation Summary

sensing the FUTURE

- Learned about SensorStudio
- Learned about GenerciSensorHub
- Evaluated INVN sensor, fusion & hub
- Understood SensorStudio Samples
- Designed your Sensor fusion
- Designed your Sensor driver
- Used your CustomSensor
- Used SensorStudio with your product

Thank You