



## RRH46410 Firmware Documentation

Firmware Version: 1.0.0

# Contents

<b>1</b>	<b>Firmware Documentation</b>	<b>1</b>
1.1	Overview	1
1.2	Folder Structure	1
1.3	Building	2
1.3.1	Windows	2
1.3.2	Raspberry Pi	2
<b>2</b>	<b>Module Index</b>	<b>3</b>
2.1	Modules	3
<b>3</b>	<b>File Index</b>	<b>4</b>
3.1	File List	4
<b>4</b>	<b>Module Documentation</b>	<b>5</b>
4.1	RRH46410 Sensor API	5
4.1.1	Detailed Description	6
4.1.1.1	HAL API Requirements	7
4.1.2	Data Structure Documentation	8
4.1.2.1	struct RRH46410_Common_Results_t	8
4.1.2.2	struct RRH46410_IAQ2_Results_t	8
4.1.2.3	struct RRH46410_PBAQ_Results_t	9
4.1.2.4	union RRH46410_Results_t	10
4.1.2.5	union RRH46410_Debug_t	11
4.1.2.6	struct RRH46410_t	11

4.1.2.7	struct RRH46410_SensorInfo_t . . . . .	13
4.1.3	Enumeration Type Documentation . . . . .	13
4.1.3.1	RRH46410_Commands_t . . . . .	13
4.1.3.2	RRH46410_ErrorCodes_t . . . . .	14
4.1.3.3	RRH46410_OperatingMode_t . . . . .	14
4.1.4	Function Documentation . . . . .	15
4.1.4.1	RRH46410_ConfigureGPIO() . . . . .	15
4.1.4.2	RRH46410_GetSensorInfo() . . . . .	15
4.1.4.3	RRH46410_Init() . . . . .	17
4.1.4.4	RRH46410_PerformCleaning() . . . . .	18
4.1.4.5	RRH46410_ReadDebug() . . . . .	18
4.1.4.6	RRH46410_ReadGPIO() . . . . .	19
4.1.4.7	RRH46410_ReadOperatingMode() . . . . .	20
4.1.4.8	RRH46410_ReadResults() . . . . .	20
4.1.4.9	RRH46410_Reset() . . . . .	21
4.1.4.10	RRH46410_SetHumidity() . . . . .	21
4.1.4.11	RRH46410_SetOperatingMode() . . . . .	22
4.1.4.12	RRH46410_SoftReset() . . . . .	23
4.1.4.13	RRH46410_WriteGPIO() . . . . .	23
4.2	Hardware Abstraction Layer API . . . . .	25
4.2.1	Detailed Description . . . . .	26
4.2.2	Data Structure Documentation . . . . .	26
4.2.2.1	struct Interface_t . . . . .	26
4.2.3	Typedef Documentation . . . . .	27
4.2.3.1	ErrorStringGenerator_t . . . . .	27
4.2.3.2	I2CImpl_t . . . . .	27
4.2.4	Enumeration Type Documentation . . . . .	27
4.2.4.1	ErrorScope_t . . . . .	27
4.2.4.2	GenericError_t . . . . .	28

4.2.4.3	HAL_Error_t . . . . .	28
4.2.5	Function Documentation . . . . .	28
4.2.5.1	HAL_Deinit() . . . . .	29
4.2.5.2	HAL_GetErrorInfo() . . . . .	29
4.2.5.3	HAL_GetErrorString() . . . . .	30
4.2.5.4	HAL_HandleError() . . . . .	30
4.2.5.5	HAL_Init() . . . . .	31
4.2.5.6	HAL_SetError() . . . . .	31
4.3	HSxxxx Sensor API . . . . .	33
4.3.1	Detailed Description . . . . .	33
4.3.1.1	HAL API Requirements . . . . .	33
4.3.2	Data Structure Documentation . . . . .	34
4.3.2.1	struct HSxxxx_Results_t . . . . .	34
4.3.2.2	struct HSxxxx_t . . . . .	34
4.3.3	Function Documentation . . . . .	35
4.3.3.1	HSxxxx_Init() . . . . .	35
4.3.3.2	HSxxxx_Measure() . . . . .	36
4.3.3.3	HSxxxx_Name() . . . . .	36
4.4	HS4xxx Sensor API . . . . .	38
4.4.1	Detailed Description . . . . .	38
4.4.2	Enumeration Type Documentation . . . . .	38
4.4.2.1	HS4xxx_ErrorCodes_t . . . . .	38
4.4.3	Function Documentation . . . . .	39
4.4.3.1	HS4xxx_Init() . . . . .	39
4.4.3.2	HS4xxx_Measure() . . . . .	39
4.4.3.3	HS4xxx_MeasureHold() . . . . .	40
4.4.3.4	HS4xxx_MeasureRead() . . . . .	41
4.4.3.5	HS4xxx_MeasureStart() . . . . .	41
4.4.3.6	HS4xxx_ReadID() . . . . .	42
4.5	HS3xxx Sensor API . . . . .	43
4.5.1	Detailed Description . . . . .	43
4.5.2	Enumeration Type Documentation . . . . .	43
4.5.2.1	HTError_t . . . . .	43
4.5.3	Function Documentation . . . . .	44
4.5.3.1	HS3xxx_Init() . . . . .	44
4.5.3.2	HS3xxx_Measure() . . . . .	44
4.5.3.3	HS3xxx_MeasureRead() . . . . .	45
4.5.3.4	HS3xxx_MeasureStart() . . . . .	46
4.5.3.5	HS3xxx_ReadID() . . . . .	46

<b>5</b>	<b>File Documentation</b>	<b>47</b>
5.1	<a href="#">/builds/industrial/zmod4xxx/zmod6xxx_generic_example/hal/custom/template.c File Reference</a>	47
5.1.1	Function Documentation	47
5.1.1.1	<a href="#">_I2CRead()</a>	48
5.1.1.2	<a href="#">_I2CWrite()</a>	48
5.1.1.3	<a href="#">_Reset()</a>	49
5.1.1.4	<a href="#">_Sleep()</a>	49
5.2	<a href="#">/builds/industrial/zmod4xxx/zmod6xxx_generic_example/hal/hal.h File Reference</a>	49
5.3	<a href="#">/builds/industrial/zmod4xxx/zmod6xxx_generic_example/sensors/hs3xxx.h File Reference</a>	50
5.4	<a href="#">/builds/industrial/zmod4xxx/zmod6xxx_generic_example/sensors/hs4xxx.h File Reference</a>	50
5.5	<a href="#">/builds/industrial/zmod4xxx/zmod6xxx_generic_example/sensors/hsxxxx.h File Reference</a>	51
5.6	<a href="#">/builds/industrial/zmod4xxx/zmod6xxx_generic_example/sensors/rrh46410.h File Reference</a>	51

## Chapter 1

# Firmware Documentation

### 1.1 Overview

The RRH46410 Firmware Package provides a set of APIs to access the RRH46410 sensor and optionally an Renesas humidity sensor along with an example of how to configure and use these sensors in an application.

The example can be compiled for the use with an EVK board which is connected to a host computer or for an Raspberry Pi. In addition an Arduino library is provided, that allows using the RRH46410 gas sensor module on the Arduino platform.

If required, the example can be adapted to run on customer specific hardware. For the adaption to the customer hardware, this firmware packages provides a HAL template in [hal/custom/template.c](#). Please refer to the documentation of this file, and the [HAL API](#) documentation. Note that some of the HAL functions may not be required by specific sensors. Please refer to the HAL API Requirements\* section of the sensor you're trying to interface.

### 1.2 Folder Structure

- `src`: Source code directory, including a Makefile for below mentioned binaries
  - `sensors`: Headers and sources for different sensors
  - `hal`: Headers, sources and libraries for hardware abstraction
- `doc`: Documentation
- `windows`: Pre-compiled example for the EVK board on a 64-bit Windows platform
- `raspberrypi`: Pre-compiled example for the 32 bit Raspberry-Pi, using the PiGPIO library
- `arduino`: Arduino library for installation in Arduino IDE (including fully working example code)

## 1.3 Building

### 1.3.1 Windows

The Windows build assumes the availability of the w64devkit build tools. w64devkit requires no installation. Please follow the instructions below to build the example code for Windows:

- Unzip contents of the RRH46410-Firmware.zip to a location of your choice, e.g. C:\myCode
- Download w64devkit.zip
- Unzip contents w64devkit.zip to a location of your choice, e.g. C:\myTools
- Open a command prompt
- Change to C:\myCode\RRH46410-Firmware
- type  
`set PATH=C:\myTools\w64devkit\bin;%PATH%`  
followed by enter
- type  
`make`  
followed by enter
- If the build process succeeds, a rrh46410-example.exe is generated in subfolder build

### 1.3.2 Raspberry Pi

The Raspberry Pi build assumes the availability of the pigpio library. Please follow the instructions below to build the example for the Raspberry Pi.

- copy the RRH46410-Firmware.zip to the Raspberry Pi
- unzip it to a location of your choice
- open a command prompt and change the directory to the location of the unzipped RRH46410-Firmware.zip
- type  
`make`  
followed by enter
- If the build process succeeds, a rrh46410-example binary is generated in subfolder build

#### Note

Root privileges are required to execute the rrh46410-example.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

RRH46410 Sensor API . . . . .	5
Hardware Abstraction Layer API . . . . .	25
HSxxxx Sensor API . . . . .	33
HS3xxx Sensor API . . . . .	43
HS4xxx Sensor API . . . . .	38



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

/builds/industrial/zmod4xxx/zmod6xxx_generic_example/hal/ <a href="#">hal.h</a>	
Renesas Environmental Sensor HAL definitions	49
/builds/industrial/zmod4xxx/zmod6xxx_generic_example/hal/custom/ <a href="#">template.c</a>	
This file serves as a template for a customer specific HAL	47
/builds/industrial/zmod4xxx/zmod6xxx_generic_example/sensors/ <a href="#">hs3xxx.h</a>	
HS3xxx sensor declarations	50
/builds/industrial/zmod4xxx/zmod6xxx_generic_example/sensors/ <a href="#">hs4xxx.h</a>	
HS4xxx sensor declarations	50
/builds/industrial/zmod4xxx/zmod6xxx_generic_example/sensors/ <a href="#">hsxxxx.h</a>	
Renesas humidity sensors (HS3xxx, HS4xxx) abstraction	51
/builds/industrial/zmod4xxx/zmod6xxx_generic_example/sensors/ <a href="#">rrh46410.h</a>	
RRH46410 API declarations	51

## Chapter 4

# Module Documentation

### 4.1 RRH46410 Sensor API

This API provides high-level access to the RRH46410 sensor.

#### Files

- file [rrh46410.h](#)  
*RRH46410 API declarations.*

#### Functions

- int [RRH46410\\_Init](#) ([RRH46410\\_t](#) \*sensor, [Interface\\_t](#) \*hal)  
*Initialize sensor object.*
- int [RRH46410\\_GetSensorInfo](#) ([RRH46410\\_t](#) \*sensor, [RRH46410\\_SensorInfo\\_t](#) \*info)  
*Read additional sensor information.*
- int [RRH46410\\_ReadOperatingMode](#) ([RRH46410\\_t](#) \*sensor)  
*Read the current operating mode.*
- int [RRH46410\\_SetOperatingMode](#) ([RRH46410\\_t](#) \*sensor, [RRH46410\\_OperatingMode\\_t](#) mode)  
*Set the current operating mode.*
- int [RRH46410\\_SetHumidity](#) ([RRH46410\\_t](#) \*sensor, float humidity)  
*Set humidity value to be used in gas algorithm computations.*
- int [RRH46410\\_ReadResults](#) ([RRH46410\\_t](#) \*sensor, void \*result)  
*Read latest sensor results.*
- int [RRH46410\\_ReadDebug](#) ([RRH46410\\_t](#) \*sensor, void \*result)  
*Read debug output.*
- int [RRH46410\\_PerformCleaning](#) ([RRH46410\\_t](#) \*sensor)  
*Perform sensor cleaning.*
- int [RRH46410\\_Reset](#) ([RRH46410\\_t](#) \*sensor)  
*Reset the sensor.*
- int [RRH46410\\_SoftReset](#) ([RRH46410\\_t](#) \*sensor)  
*Perform a software reset.*
- int [RRH46410\\_ConfigureGPIO](#) ([RRH46410\\_t](#) \*sensor, uint8\_t outputMask)  
*Configure GPIO1 and GPIO2 direction.*
- int [RRH46410\\_WriteGPIO](#) ([RRH46410\\_t](#) \*sensor, uint8\_t mask, uint8\_t state)  
*Write one or multiple GPIO pins.*
- int [RRH46410\\_ReadGPIO](#) ([RRH46410\\_t](#) \*sensor, uint8\_t \*state)  
*Read the state of all GPIO pins.*

## Data Structures

- struct [RRH46410\\_Common\\_Results\\_t](#)  
*This struct helps accessing sample ID independent from operating mode. [More...](#)*
- struct [RRH46410\\_IAQ2\\_Results\\_t](#)  
*Results returned from sensor in IAQ2 and IAQ2 ULP mode. [More...](#)*
- struct [RRH46410\\_PBAQ\\_Results\\_t](#)  
*Raw result returned from sensor in PBAQ mode. [More...](#)*
- union [RRH46410\\_Results\\_t](#)  
*Generic result data structure, joining the results of the different operating modes. [More...](#)*
- union [RRH46410\\_Debug\\_t](#)  
*Debug output data structure. [More...](#)*
- struct [RRH46410\\_t](#)  
*Data structure holding information required for RRH46410 API operation. [More...](#)*
- struct [RRH46410\\_SensorInfo\\_t](#)  
*Sensor information data structure. [More...](#)*

## Enumerations

- enum [RRH46410\\_OperatingMode\\_t](#) {  
[omSuspend](#) = 0x00, [omIAQ2](#) = 0x01, [omIAQ2\\_ULP](#) = 0x02, [omPBAQ](#) = 0x05,  
[omCleaning](#) = 0x80, [omUnknown](#) = 0xff }  
*Operating mode types.*
- enum [RRH46410\\_Commands\\_t](#) {  
[cmdGetSensorInfo](#) = 0x00, [cmdGetTrackingNumber](#) = 0x01, [cmdGetOperatingMode](#) = 0x10, [cmdSetOperatingMode](#) = 0x11,  
[cmdSetHumidity](#) = 0x12, [cmdGetResults](#) = 0x18, [cmdSetGPIOConfig](#) = 0x20, [cmdGetGPIOState](#) = 0x21,  
[cmdSetOutputHigh](#) = 0x22, [cmdSetOutputLow](#) = 0x23, [cmdGetDebugData](#) = 0x31, [cmdSoftwareReset](#) = 0x8f  
}  
*Define names for RRH46410 commands.*
- enum [RRH46410\\_ErrorCodes\\_t](#) {  
[ecWarmup](#) = 0x01, [ecCleaningCountExceeded](#) = 0x08, [ecPOR](#) = 0x09, [ecDamage](#) = 0x0a,  
[ecDataNotReady](#) = 0x10, [ecInternalComm](#) = 0x20, [ecHostToSensorChecksum](#) = 0x40, [ecInvalidCommand](#) = 0x80,  
[ecSampleNotNew](#) = 0x100, [ecSensorToHostChecksum](#) = 0x101, [ecCleaningTimeout](#) = 0x102, [ecGPIOConfigInvalid](#) = 0x103 }  
*Definition of error codes for RRH46410 sensors.*

### 4.1.1 Detailed Description

This API provides high-level access to the RRH46410 sensor.

The RRH46410 is a sensor module that combines a gas sensor with an embedded microcontroller, removing the need for doing algorithmic computations in the application processor. In addition the embedded MCU maintains the correct timing required to get reliable results from the sensor.

The RRH46410 API is the most convenient way of accessing the sensor. The example code provided with this firmware package may be used as basis for custom applications. All that needs to be done to port this example to the customer hardware is modifying the HAL template, provided in directory `src/hal/custom/template.c`. Refer to the [template.c](#) documentation for further details.

#### 4.1.1.1 HAL API Requirements

HAL API ports to a customer specific hardware require the following function pointers of the [Interface\\_t](#) HAL data structure to be initialized and working as documented:

Function Pointer	Required
<a href="#">i2cRead</a>	<b>Mandatory</b>
<a href="#">i2cWrite</a>	Not required
<a href="#">msSleep</a>	<b>Mandatory</b>
<a href="#">reset</a>	Optional

## 4.1.2 Data Structure Documentation

### 4.1.2.1 struct RRH46410\_Common\_Results\_t

This struct helps accessing sample ID independent from operating mode.

#### Data Fields

- `uint8_t` [sampleID](#)

#### 4.1.2.1.1 Field Documentation

##### 4.1.2.1.1.1 sampleID

`uint8_t` [sampleID](#)

Sample counter (wrapping at 255)

### 4.1.2.2 struct RRH46410\_IAQ2\_Results\_t

Results returned from sensor in IAQ2 and IAQ2 ULP mode.

#### Data Fields

- `uint8_t` [sampleID](#)
- `uint8_t` [iaq](#)
- `uint16_t` [tvoc](#)
- `uint16_t` [etoh](#)
- `uint16_t` [eco2](#)
- `uint8_t` [rel\\_IAQ](#)

#### 4.1.2.2.1 Field Documentation

#### 4.1.2.2.1.1 eco2

`uint16_t eco2`

eCO2 concentration (ppm)

#### 4.1.2.2.1.2 etoh

`uint16_t etoh`

EtOH concentration \* 100 (ppm)

#### 4.1.2.2.1.3 iaq

`uint8_t iaq`

IAQ index \* 10

#### 4.1.2.2.1.4 rel\_IAQ

`uint8_t rel_IAQ`

Relative IAQ index / 10

#### 4.1.2.2.1.5 sampleID

`uint8_t sampleID`

Sample counter (wrapping at 255)

#### 4.1.2.2.1.6 tvoc

`uint16_t tvoc`

TVOC concentration \* 100 (mg/m<sup>3</sup>)

#### 4.1.2.3 struct RRH46410\_PBAQ\_Results\_t

Raw result returned from sensor in PBAQ mode.

##### Data Fields

- `uint8_t` [sampleID](#)
- `uint16_t` [tvoc](#)
- `uint16_t` [etoh](#)

#### 4.1.2.3.1 Field Documentation

##### 4.1.2.3.1.1 etoh

`uint16_t etoh`

EtOH concentration \* 100 (ppm)

##### 4.1.2.3.1.2 sampleID

`uint8_t sampleID`

Sample counter (wrapping at 255)

##### 4.1.2.3.1.3 tvoc

`uint16_t tvoc`

TVOC concentration \* 100 (mg/m<sup>3</sup>)

#### 4.1.2.4 union RRH46410\_Results\_t

Generic result data structure, joining the results of the different operating modes.

##### Data Fields

- [RRH46410\\_Common\\_Results\\_t](#) common
- [RRH46410\\_IAQ2\\_Results\\_t](#) iaq2
- [RRH46410\\_PBAQ\\_Results\\_t](#) pbaq

#### 4.1.2.4.1 Field Documentation

##### 4.1.2.4.1.1 common

[RRH46410\\_Common\\_Results\\_t](#) common

Common part of all result data structures

##### 4.1.2.4.1.2 iaq2

[RRH46410\\_IAQ2\\_Results\\_t](#) iaq2

Result data structure used in IAQ2 and IAQ2\_ULP operating mode

#### 4.1.2.4.1.3 pbaq

`RRH46410_PBAQ_Results_t` pbaq

Result data structure used in PBAQ operating mode

#### 4.1.2.5 union RRH46410\_Debug\_t

Debug output data structure.

##### Data Fields

- `uint8_t` `sampleID`
- `uint8_t` `raw` [100]

##### 4.1.2.5.1 Field Documentation

###### 4.1.2.5.1.1 raw

`uint8_t` `raw`[100]

Raw debug data

###### 4.1.2.5.1.2 sampleID

`uint8_t` `sampleID`

Sample counter (wrapping at 255)

#### 4.1.2.6 struct RRH46410\_t

Data structure holding information required for RRH46410 API operation.

##### Data Fields

- `Interface_t` \* `hal`
- `uint8_t` `i2cAddress`
- `RRH46410_OperatingMode_t` `operatingMode`
- `uint8_t` `resultsLength`
- `uint8_t` `debugLength`
- `uint32_t` `measurementInterval`
- `uint8_t` `recentsampleID`
- `uint8_t` `remainingWarmup`



#### 4.1.2.6.1 Field Documentation

##### 4.1.2.6.1.1 debugLength

`uint8_t debugLength`

Length of the debug results in current operating mode

##### 4.1.2.6.1.2 hal

`Interface_t* hal`

Pointer to the hal object for physical communication

##### 4.1.2.6.1.3 i2cAddress

`uint8_t i2cAddress`

I2C slave address of the RRH46410

##### 4.1.2.6.1.4 measurementInterval

`uint32_t measurementInterval`

Measurement interval of the current operating mode

##### 4.1.2.6.1.5 operatingMode

`RRH46410_OperatingMode_t operatingMode`

Current operating mode of the RRH46410

##### 4.1.2.6.1.6 recentsampleID

`uint8_t recentsampleID`

Id of last sample read from sensor

##### 4.1.2.6.1.7 remainingWarmup

`uint8_t remainingWarmup`

Remaining number of samples required for sensor stabilization

##### 4.1.2.6.1.8 resultsLength

`uint8_t resultsLength`

Length of the sensor results in current operating mode

#### 4.1.2.7 struct RRH46410\_SensorInfo\_t

Sensor information data structure.

##### Data Fields

- uint16\_t [productId](#)
- uint8\_t [trackingNumber](#) [6]
- struct {
  - uint8\_t **major**
  - uint8\_t **minor**
  - uint8\_t **patch**
- } [fwVersion](#)

##### 4.1.2.7.1 Field Documentation

###### 4.1.2.7.1.1 fwVersion

```
struct { ... } fwVersion
```

RRH46410 internal firmware version

###### 4.1.2.7.1.2 productId

```
uint16_t productId
```

Sensor product Id

###### 4.1.2.7.1.3 trackingNumber

```
uint8_t trackingNumber[6]
```

Sensor tracking number

#### 4.1.3 Enumeration Type Documentation

##### 4.1.3.1 RRH46410\_Commands\_t

```
enum RRH46410\_Commands\_t
```

Define names for RRH46410 commands.

## Enumerator

cmdGetSensorInfo	Read sensor product id and firmware version
cmdGetTrackingNumber	Read sensor tracking number (unique ID)
cmdGetOperatingMode	Read the operating mode of the sensor
cmdSetOperatingMode	Change the operating mode of the sensor
cmdSetHumidity	Set the ambient humidity value
cmdGetResults	Read sensor results
cmdSetGPIOConfig	Set GPIOs as input or output
cmdGetGPIOState	Read the state of GPIO pins
cmdSetOutputHigh	Set output pin to high level
cmdSetOutputLow	Set output pin to low level
cmdGetDebugData	Read debug data
cmdSoftwareReset	Reset the sensor

## 4.1.3.2 RRH46410\_ErrorCodes\_t

```
enum RRH46410_ErrorCodes_t
```

Definition of error codes for RRH46410 sensors.

## Enumerator

ecWarmup	Sensor is in stabilization phase, results may be inaccurate
ecCleaningCountExceeded	Cleaning has been completed before.
ecPOR	Power-on reset event. Check power supply and reset pin.
ecDamage	Gas sensor may be damaged. Refer to datasheet for details.
ecDataNotReady	Data requested before data is available.
ecInternalComm	Internal communication error in RRH46410 module.
ecHostToSensorChecksum	Checksum received by sensor does not match data.
ecInvalidCommand	Command is unknown to the sensor. Check datasheet for available commands.
ecSampleNotNew	Sensor has not taken a new sample after reading the previous one. Wait until new sample is available
ecSensorToHostChecksum	The checksum received does not match the preceded data.
ecCleaningTimeout	Sensor cleaning took longer than specified.
ecGPIOConfigInvalid	The GPIO configuration provided is invalid.

## 4.1.3.3 RRH46410\_OperatingMode\_t

```
enum RRH46410_OperatingMode_t
```

Operating mode types.

Enumerator

omSuspend	Sensor is not taking measurements
omIAQ2	IAQ 2nd Gen mode
omIAQ2_ULP	IAQ 2nd Gen Ultra Low Power mode
omPBAQ	Public Building Air Quality mode
omCleaning	Execute cleaning cycle
omUnknown	Used by application before initialization

4.1.4 Function Documentation

4.1.4.1 RRH46410\_ConfigureGPIO()

```
int RRH46410_ConfigureGPIO (
    RRH46410_t * sensor,
    uint8_t outputMask )
```

Configure GPIO1 and GPIO2 direction.

Bits 0 and 1 determine the direction of GPIO1 and GPIO2, respectively. Setting the corresponding bit to 1 configures the pin as output. Otherwise the pin is configured as input.

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
<i>outputMask</i>	bit field defining GPIO direction

Returns

int error code

Return values

0	Success
other	Error

4.1.4.2 RRH46410\_GetSensorInfo()

```
int RRH46410_GetSensorInfo (
    RRH46410_t * sensor,
    RRH46410_SensorInfo_t * info )
```

Read additional sensor information.

Renesas Confidential

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
<i>info</i>	pointer to <a href="#">RRH46410_SensorInfo_t</a> object to be populated

Returns

int error code

Return values

0	Success
!=0	Error

4.1.4.3 RRH46410\_Init()

```
int RRH46410_Init (
    RRH46410\_t * sensor,
    Interface\_t * hal )
```

Initialize sensor object.

This function tries calling [RRH46410\\_ReadOperatingMode\(\)](#) with using *hal* as hardware interface. As the sensor is not accessible immediately after power on, this operation is repeated up to 50 times, ever 20 ms.

On success the interface property of the sensor object is set to *hal*. Otherwise the interface property will be set to NULL.

Note

The RRH46410 API requires that the HAL interface object has the [Interface\\_t::i2cRead](#) and [Interface\\_t::ms↵Sleep](#) members defined. If [Interface\\_t::reset](#) is defined, the RRH46410\_Reset function will use hardware reset, otherwise software reset is used.

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
<i>hal</i>	pointer to object providing physical communication

Returns

int error code

Return values

0	Success
!=0	Error

4.1.4.4   RRH46410\_PerformCleaning()

```
int RRH46410_PerformCleaning (
    RRH46410_t * sensor )
```

Perform sensor cleaning.

The RRH46410 cleaning procedure shall be executed once after production to remove residual materials that would impair the measurement accuracy. The whole procedure requires 1 minute to complete. Do not interrupt the cleaning!

**Note**

No measurements can be taken while clening is running.  
Cleaning can only be executed once per sensor.

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
---------------	--

Returns

int error code

Return values

<i>0</i>	Success
<i>ecCleaningCountExceeded</i>	Cleaning has already been performed
<i>other</i>	Error

4.1.4.5   RRH46410\_ReadDebug()

```
int RRH46410_ReadDebug (
    RRH46410_t * sensor,
    void * result )
```

Read debug output.

If algorithm support is required, Renesas may ask to provide the output delivered by this function.

**Note**

: This function should be called shortly after [RRH46410\\_ReadResults\(\)](#)

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
<i>result</i>	pointer to debug data object (raw byte array)

Note

: The caller must ensure the buffer pointed to by result is big enough to store the debug data. The debug data size for the present operating mode is accessible through the [RRH46410\\_t::debugLength](#)

Returns

int error code

Return values

<i>0</i>	Success
<i>other</i>	Error

4.1.4.6 RRH46410\_ReadGPIO()

```
int RRH46410_ReadGPIO (
    RRH46410\_t * sensor,
    uint8_t * state )
```

Read the state of all GPIO pins.

The read operation works on inputs and outputs, and reports the value that is actually observed on the GPIO pin.

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
<i>state</i>	pointer to byte where result is written to

Returns

int error code

Return values

<i>0</i>	Success
<i>other</i>	Error



4.1.4.7 RRH46410\_ReadOperatingMode()

```
int RRH46410_ReadOperatingMode (
    RRH46410_t * sensor )
```

Read the current operating mode.

Call this function to determine the current operating mode of the sensor. On success, the object pointed to by *sensor* will be updated with the appropriate values.

**Note**  
: This function is called internally by RRH46410\_Init.

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
---------------	--

Returns

int error code

Return values

0	Success
!=0	Error

4.1.4.8 RRH46410\_ReadResults()

```
int RRH46410_ReadResults (
    RRH46410_t * sensor,
    void * result )
```

Read latest sensor results.

The data returned depends on the operating mode

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
<i>result</i>	pointer to results object

Returns

int error code

Return values

0	Success
ecSampleNotNew	result has not been updated since the last readout
other	Error

4.1.4.9 RRH46410\_Reset()

```
int RRH46410_Reset (
    RRH46410_t * sensor )
```

Reset the sensor.

This is a convenience function that toggles the reset pin if supported by the interface object. Otherwise it sends a soft reset command through I2C.

**Note**  
The sensor requires some time to be accessible after reset. Please refer to the data sheet for further details.

Parameters

sensor	pointer to <a href="#">RRH46410_t</a> object
--------	--

Returns

int error code

Return values

0	Success
other	Error

4.1.4.10 RRH46410\_SetHumidity()

```
int RRH46410_SetHumidity (
    RRH46410_t * sensor,
    float humidity )
```

Set humidity value to be used in gas algorithm computations.

The gas sensing hardware is sensitive to humidity changes. In order to get accurate gas measurement results, the actual humidity value should be provided.

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
<i>humidity</i>	Humidity value to be use in gas algorithm computations

Returns

int error code

Return values

0	Success
!=0	Error

4.1.4.11 RRH46410\_SetOperatingMode()

```
int RRH46410_SetOperatingMode (
    RRH46410\_t * sensor,
    RRH46410\_OperatingMode\_t mode )
```

Set the current operating mode.

On success, the sensor data structure will be updated.

Note

: A change of the operating mode results in a reset of the sample ID

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
<i>mode</i>	new operating mode

Returns

int error code

Return values

0	Success
other	Error

4.1.4.12 RRH46410\_SoftReset()

```
int RRH46410_SoftReset (
    RRH46410_t * sensor )
```

Perform a software reset.

This function issues a Reset command through the I2C interface.

**Note**

The sensor requires some time to be accessible after reset. Please refer to the data sheet for further details.

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
---------------	--

Returns

int error code

Return values

0	Success
<i>other</i>	Error

4.1.4.13 RRH46410\_WriteGPIO()

```
int RRH46410_WriteGPIO (
    RRH46410_t * sensor,
    uint8_t mask,
    uint8_t state )
```

Write one or multiple GPIO pins.

GPIO1 and GPIO2 can be configured as output. Only pins for which the mask bitfield contains a 1 are written. The state file contains the actual value.

**Note**

If multiple pins are changed, there may be a significant delay between the actual change of the output values of these pins. This especially applies if the pins are set to different states.

Parameters

<i>sensor</i>	pointer to <a href="#">RRH46410_t</a> object
<i>mask</i>	bit field defining GPIOs that shall be affected
<i>state</i>	bit field defining the logic output value of pins

Returns

int error code

Return values

0	Success
other	Error

## 4.2 Hardware Abstraction Layer API

Hardware abstraction layer (HAL) API, used by Renesas Environmental Sensor APIs to physically access the sensor.

### Files

- file [hal.h](#)  
*Renesas Environmental Sensor HAL definitions.*

### Functions

- int [HAL\\_Init](#) ([Interface\\_t](#) \*hal)  
*Initialize hardware and populate [Interface\\_t](#) object.*
- int [HAL\\_Deinit](#) ([Interface\\_t](#) \*hal)  
*Cleanup before program exit.*
- void [HAL\\_HandleError](#) (int errorCode, void const \*context)  
*Error handling function.*
- int [HAL\\_SetError](#) (int error, int scope, [ErrorStringGenerator\\_t](#) errStrFn)  
*Function storing error information.*
- char const \* [HAL\\_GetErrorInfo](#) (int \*error, int \*scope, char \*str, int bufSize)  
*Get detailed information for last error.*
- char const \* [HAL\\_GetErrorString](#) (int error, int scope, char \*str, int bufSize)  
*Error string generator for HAL-scoped errors.*

### Data Structures

- struct [Interface\\_t](#)  
*A structure of pointers to hardware specific functions. [More...](#)*

### Typedefs

- typedef int(\* [I2CImpl\\_t](#)) (void \*, uint8\_t \*, uint8\_t \*, int, uint8\_t \*, int)  
*Function pointer type defining signature of I2C functions.*
- typedef char const \*(\* [ErrorStringGenerator\\_t](#)) (int, int, char \*, int)  
*Function type used for generation of error strings.*

### Enumerations

- enum [GenericError\\_t](#) { [ecSuccess](#) = 0, [ecHALError](#) = 0x10000 }  
*Error code definitions.*
- enum [ErrorScope\\_t](#) { [esSensor](#) = 0x00000000, [esAlgorithm](#) = 0x10000000, [esInterface](#) = 0x20000000, [esHAL](#) = 0x30000000 }  
*Success status code and error scopes.*
- enum [HALError\\_t](#) {  
  [heNoInterface](#) = 1, [heNotImplemented](#), [heI2CReadMissing](#), [heI2CWriteMissing](#),  
  [heSleepMissing](#), [heResetMissing](#) }  
*HAL scope error definitions.*

### 4.2.1 Detailed Description

Hardware abstraction layer (HAL) API, used by Renesas Environmental Sensor APIs to physically access the sensor.

The HAL API is a generic API that is used by all Renesas types of Environmental Sensors. It defines the interface that the sensor API can use to access the sensor on an arbitrary hardware platform. Renesas provides HAL implementations for EVK boards, Raspberry PI and Arduino. In order to use a specific sensor API in the customer application, the customer must implement the HAL API for his hardware platform. For that purpose, the template file [template.c](#) is provided, which must be adapted to work on the customer hardware. Please refer to the comments in the template file and the documentation of the HAL API for details.

All HAL API related files are located in path `src/hal` and its subdirectories.

### 4.2.2 Data Structure Documentation

#### 4.2.2.1 struct Interface\_t

A structure of pointers to hardware specific functions.

##### Data Fields

- void \* [handle](#)
- [I2CImpl\\_t](#) [i2cRead](#)
- [I2CImpl\\_t](#) [i2cWrite](#)
- void(\* [msSleep](#) )(uint32\_t ms)
- int(\* [reset](#) )(void \*[handle](#))

#### 4.2.2.1.1 Field Documentation

##### 4.2.2.1.1.1 handle

`void* handle`

handle to physical interface

##### 4.2.2.1.1.2 i2cRead

[I2CImpl\\_t](#) [i2cRead](#)

Pointer to I2C read implementation

The read operation may be preceded by a write to the same slave address. The function accepts a pointer to the interface handle, the slave address of the device to communicate with and two pairs of buffer pointer and buffer length. The first pair of buffer pointer / buffer length defines the data to be written, the second pair provides the pointer to the buffer where received data is stored and how many bytes shall be read. If the length field of the first buffer is not zero, the implementation must send an I2C write condition on the bus, transfer the data and send a repeated start condition followed by the corresponding read condition on the bus. If the length parameter of the first buffer is zero, no write is generated and the read is started immediately. The transmission must be terminated with a stop condition on the bus.

#### 4.2.2.1.1.3 i2cWrite

`I2CImpl_t i2cWrite`

Pointer to I2C write implementation

For convenience the write operation accepts two pairs of buffer pointer and buffer length. This allows to transfer addresses or commands prior to data without the need to manually concatenate transmit data in a buffer. The implementation of this function must generate a start condition followed by the I2C slave address of the target device, followed by all data from both buffers. At the end of the transmission a stop bit must be generated on the bus.

#### 4.2.2.1.1.4 msSleep

`void( * msSleep) (uint32_t ms)`

Pointer to delay function

An implementation must delay execution by the specified number of `ms`

#### 4.2.2.1.1.5 reset

`int( * reset) (void *handle)`

Pointer to reset function

Implementation must pulse the reset pin

### 4.2.3 Typedef Documentation

#### 4.2.3.1 ErrorStringGenerator\_t

`typedef char const*( * ErrorStringGenerator_t) (int, int, char *, int)`

Function type used for generation of error strings.

Functions of this type may be passed to [HAL\\_SetError\(\)](#) to generate meaningful descriptions of error conditions.

#### 4.2.3.2 I2CImpl\_t

`typedef int( * I2CImpl_t) (void *, uint8_t, uint8_t *, int, uint8_t *, int)`

Function pointer type defining signature of I2C functions.

This function pointer typedef is used in [Interface\\_t](#) objects to hold pointers to I2C implementations of read and write.

### 4.2.4 Enumeration Type Documentation

#### 4.2.4.1 ErrorScope\_t

`enum ErrorScope_t`

Success status code and error scopes.



## Enumerator

esSensor	Sensor scope
esAlgorithm	Algorithm scope
esInterface	Interface scope
esHAL	HAL scope

## 4.2.4.2 GenericError\_t

```
enum GenericError_t
```

Error code definitions.

## Enumerator

ecSuccess	Operation completed successfully
ecHALError	Returned by sensor API if a HAL function failed. Specific information about the error can be obtained using the function <a href="#">HAL_GetErrorInfo()</a> .

## 4.2.4.3 HALError\_t

```
enum HALError_t
```

HAL scope error definitions.

When sensors are initialized (e.g. `init_hardware()`), the hal objects is checked whether all HAL functions required by the sensor are provided. If a function is missing one of the errors from this enumeration is returned.

## Enumerator

heNoInterface	There was no interface found
heNotImplemented	The requested function is not implemented
heI2CReadMissing	<a href="#">Interface_t::i2cRead</a> not provided
heI2CWriteMissing	<a href="#">Interface_t::i2cWrite</a> not provided
heSleepMissing	<a href="#">Interface_t::msSleep</a> not provided
heResetMissing	<a href="#">Interface_t::reset</a> not provided

## 4.2.5 Function Documentation

4.2.5.1 HAL\_Deinit()

```
int HAL_Deinit (
    Interface_t * hal )
```

Cleanup before program exit.

This function shall free up resources that have been allocated through HAL\_Init().

Parameters

hal	pointer to Interface_t object to be deinitialized
-----	---

Returns

error code

Return values

0	on success
!=0	in case of error

4.2.5.2 HAL\_GetErrorInfo()

```
char const* HAL_GetErrorInfo (
    int * error,
    int * scope,
    char * str,
    int bufSize )
```

Get detailed information for last error.

Use this function in the error handler to obtain information about the last error code and which component generated it. In addition if an error string generator function was provided during error generation, this function may return a text string, describing the error in more detail.

Parameters

error	Pointer to integer, where the error code is written
scope	Pointer to integer, where the error scope (module that was generating the error is written)
str	Pointer to string buffer, where error message is written. If no string information is required, pass NULL pointer.
bufSize	Size of the string buffer, pass 0 if not used

Returns

Value passed in str

4.2.5.3 HAL\_GetErrorString()

```
char const* HAL_GetErrorString (
    int error,
    int scope,
    char * str,
    int bufSize )
```

Error string generator for HAL-scoped errors.

This function generates error information for HAL scoped errors. Usually user code does not need to use this function directly.

Parameters

<i>error</i>	Error code for which error information is to be returned
<i>scope</i>	Error scope for which error information is to be returned
<i>str</i>	Pointer to string buffer, where error message is written
<i>bufSize</i>	Size of the string buffer

Returns

Value passed in str

4.2.5.4 HAL\_HandleError()

```
void HAL_HandleError (
    int errorCode,
    void const * context )
```

Error handling function.

The implementation of this function defines the behavior of the application code when an error occurs during execution.

Parameters

<i>errorCode</i>	code of the error to be handled
<i>context</i>	additional context information

Define what happens in case of error

4.2.5.5 HAL\_Init()

```
int HAL_Init (
    Interface_t * hal )
```

Initialize hardware and populate `Interface_t` object.

Any implementation must initialize those members of the `Interface_t` object that are required by the sensor being operated with pointers to functions that implement the behavior as specified in the `Interface_t` member documentation.

Parameters

<i>hal</i>	pointer to <code>Interface_t</code> object to be initialized
------------	--

Returns

error code

Return values

0	on success
!=0	in case of error

4.2.5.6 HAL\_SetError()

```
int HAL_SetError (
    int error,
    int scope,
    ErrorStringGenerator_t errStrFn )
```

Function storing error information.

The sensor interface has different components which may generate error conditions. To keep the sensor interface as simple as possible, this function is called in case of an error. The return value of this function will be returned as error code of the function in which an error has occurred.

Internally, this function stores the error code and the scope of the error (that is which module was generating the error) in a data structure. For all errors which do not have the scope `esSensor`, this function will return the generic error code `ecHALError`. Error codes generated by the sensor, are returned directly.

It is possible to pass an error string generator function along with the error information. If this function is provided, the error handler can query an error string, providing more meaningful error information.

## Parameters

<i>error</i>	An error code
<i>scope</i>	The scope of the error (integer identifying a module)
<i>errStrFn</i>	Optional function pointer that can decode generate a meaningful message for the error code. Pass NULL if not used.

## 4.3 HSxxxx Sensor API

API providing a unified interface for Renesas Humidity and Temperature sensors.

### Modules

- [HS3xxx Sensor API](#)  
*HS3xxx Temperature/Humidity Sensor API.*
- [HS4xxx Sensor API](#)  
*HS4xxx Temperature/Humidity Sensor API.*

### Files

- file [hsxxxx.h](#)  
*Renesas humidity sensors (HS3xxx, HS4xxx) abstraction.*

### Functions

- int [HSxxxx\\_Init](#) ([HSxxxx\\_t](#) \*sensor, [Interface\\_t](#) \*hal)  
*Initialize the sensor object.*
- int [HSxxxx\\_Measure](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Perform one temperature measurement.*
- char const \* [HSxxxx\\_Name](#) ([HSxxxx\\_t](#) \*sensor)  
*Return the temperature/humidity sensor type name.*

### Data Structures

- struct [HSxxxx\\_Results\\_t](#)  
*Data structure holding humidity/temperature results. [More...](#)*
- struct [HSxxxx\\_t](#)  
*Data structure holding information required for HSxxxx API operation. [More...](#)*

#### 4.3.1 Detailed Description

API providing a unified interface for Renesas Humidity and Temperature sensors.

The Renesas HS3xxx and HS4xxx are highly accurate, ultra-low power, fully calibrated relative humidity and temperature sensors. These sensors may be used as standalone sensors or used as companion sensor for humidity and temperature sensitive applications (e.g., for Renesas gas sensors).

The present API is provided as a unified interface to both of these sensor types.

##### 4.3.1.1 HAL API Requirements

HAL API ports to a customer specific hardware require the following function pointers of the [Interface\\_t](#) HAL data structure to be initialized and working as documented:

Function Pointer	Required
<a href="#">i2cRead</a>	<b>Mandatory</b>
<a href="#">i2cWrite</a>	<b>Mandatory</b>
<a href="#">msSleep</a>	<b>Mandatory</b>
<a href="#">reset</a>	Not required

## 4.3.2 Data Structure Documentation

### 4.3.2.1 struct HSxxxx\_Results\_t

Data structure holding humidity/temperature results.

#### Data Fields

- float [temperature](#)
- float [humidity](#)

#### 4.3.2.1.1 Field Documentation

##### 4.3.2.1.1.1 humidity

`float humidity`

Relative humidity

##### 4.3.2.1.1.2 temperature

`float temperature`

Temperature value in degree Celsius

### 4.3.2.2 struct HSxxxx\_t

Data structure holding information required for HSxxxx API operation.

#### Data Fields

- [Interface\\_t](#) \* [interface](#)
- `uint8_t` [i2cAddress](#)

#### 4.3.2.2.1 Field Documentation

4.3.2.2.1.1 i2cAddress

uint8\_t i2cAddress

I2C slave address of the HSxxxx sensor

4.3.2.2.1.2 interface

Interface\_t\* interface

Pointer to the hal object for physical communication

4.3.3 Function Documentation

4.3.3.1 HSxxxx\_Init()

```
int HSxxxx_Init (
    HSxxxx_t * sensor,
    Interface_t * hal )
```

Initialize the sensor object.

This function tries searching for a HS4xxx sensor first, by accessing the corresponding I2C address. If no such sensor is found, the function searches for a HS3xxx.

The type of sensor being detected can be determined using the function [HSxxxx\\_Name\(\)](#).

Parameters

<i>sensor</i>	Pointer to sensor object to be initialized.
<i>hal</i>	Pointer to hal object for physical communication.

Returns

int Error code

Return values

<i>0</i>	On success
<i>other</i>	On error



4.3.3.2 HSxxx\_Measure()

```
int HSxxx_Measure (
    HSxxx_t * sensor,
    HSxxx_Results_t * results )
```

Perform one temperature measurement.

This function starts a temperature/humidity measurement and waits for the availability of the result before it returns.

Note

This function is implemented as blocking function. Thus while the measurement is ongoing no other code is executed. Depending on the sensor the bocking time may be multiple tens of milliseconds.

Parameters

<i>sensor</i>	Pointer to the sensor object to be used.
<i>results</i>	Pointer to data structure for result storage.

Returns

int Error code

Return values

<i>0</i>	On success
<i>other</i>	On error

4.3.3.3 HSxxx\_Name()

```
char const* HSxxx_Name (
    HSxxx_t * sensor )
```

Return the temperature/humidity sensor type name.

The function HSxxx\_Init can identify different types of temperature/ humidity sensors. This function may be used to determine the sensor type that has been identified. The identification is performed based on the sensors I2C address.

Parameters

<i>sensor</i>	Pointer to the sensor object to be queried.
---------------	---

Returns

int Error code

Return values

0	On success
other	On error

## 4.4 HS4xxx Sensor API

HS4xxx Temperature/Humidity Sensor API.

### Files

- file [hs4xxx.h](#)  
*HS4xxx sensor declarations.*

### Functions

- int [HS4xxx\\_Init](#) ([HSxxxx\\_t](#) \*sensor, [Interface\\_t](#) \*hal)  
*Initialize the sensor object.*
- int [HS4xxx\\_ReadID](#) ([HSxxxx\\_t](#) \*sensor, uint32\_t \*id)  
*Read the unique sensor ID of the HS4xxx.*
- int [HS4xxx\\_Measure](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Perform a temperature/humidity measurement cycle.*
- int [HS4xxx\\_MeasureHold](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Perform a temperature/humidity measurement cycle using hold mode.*
- int [HS4xxx\\_MeasureStart](#) ([HSxxxx\\_t](#) \*sensor)  
*Start a temperature/humidity measurement cycle in non-hold mode.*
- int [HS4xxx\\_MeasureRead](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Read temperature/humidity results in non-hold mode.*

### Enumerations

- enum [HS4xxx\\_ErrorCodes\\_t](#) { [hteHS4xxxCRCError](#) = 1 }  
*Error code definitions specific for the HS4xxx API.*

#### 4.4.1 Detailed Description

HS4xxx Temperature/Humidity Sensor API.

HS4xxx is a series of highly accurate, fully calibrated automotive-grade relative humidity and temperature sensors. HS4xxx sensors provide a CRC checksum for communication integrity checking and have different modes of operation.

The HS4xxx API provides convenient access to the sensor's temperature & humidity measurement capabilities.

#### 4.4.2 Enumeration Type Documentation

##### 4.4.2.1 HS4xxx\_ErrorCodes\_t

enum [HS4xxx\\_ErrorCodes\\_t](#)

Error code definitions specific for the HS4xxx API.

Enumerator

hteHS4xxxCRCError	Result CRC error
-------------------	------------------

4.4.3 Function Documentation

4.4.3.1 HS4xxx\_Init()

```
int HS4xxx_Init (
    HSxxxx_t * sensor,
    Interface_t * hal )
```

Initialize the sensor object.

This function tries accessing the HS4xxx I2C address using *hal* as hardware interface. On success, the sensor object is initialized and can be used afterwards. Otherwise, an error code is returned and the sensor object is not usable.

Note

The HS4xxx API requires that the HAL interface object has the [Interface\\_t::i2cRead](#) and [Interface\\_t::i2cWrite](#) members defined. In addition, [HS4xxx\\_Measure\(\)](#) requires the [Interface\\_t::msSleep](#) member.

Parameters

<i>sensor</i>	Pointer to sensor object to be initialized
<i>hal</i>	Pointer to HAL object providing physical communication

Returns

int Error code

Return values

0	On success
other	On error

4.4.3.2 HS4xxx\_Measure()

```
int HS4xxx_Measure (
    HSxxxx_t * sensor,
    HSxxxx_Results_t * results )
```

Perform a temperature/humidity measurement cycle.

This function starts a measurement cycle in non-hold mode, waits for the result to be available and reads it. If the checksum computation is correct the result is stored in the *results* data structure. Otherwise the contents of *results* is left unmodified and an error code is returned.

Note

Although the HS4xxx is able to perform the requested measurement with a single I2C transaction (refer to H↔S4xxx\_MeasureHold), this function uses the [HS4xxx\\_MeasureStart\(\)](#) and [HS4xxx\\_MeasureRead\(\)](#) functions together with the [Interface\\_t::msSleep](#) function. This is to allow operation on a wider variety of I2C interfaces. [HS4xxx\\_MeasureHold\(\)](#) requires a minimum of 200kHz I2C clock frequency which is not supported by all interfaces.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>results</i>	Pointer to a data structure, to store results in

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

4.4.3.3 HS4xxx\_MeasureHold()

```
int HS4xxx_MeasureHold (
    HSxxxx_t * sensor,
    HSxxxx_Results_t * results )
```

Perform a temperature/humidity measurement cycle using hold mode.

This function performs temperature/humidity measurements in hold mode and reads the results form the sensor. If the checksum computation is correct the result is stored in the *results* data structure. Otherwise the contents of *results* is left unmodified and an error code is returned.

Note

For hold measurements, the I2C clock frequency must be at least 200kHz. Otherwise the result readout is unreliable.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>results</i>	Pointer to a data structure, to store results in

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

4.4.3.4 HS4xxx\_MeasureRead()

```
int HS4xxx_MeasureRead (
    HSxxxx_t * sensor,
    HSxxxx_Results_t * results )
```

Read temperature/humidity results in non-hold mode.

This function reads the temperature/humidity results from a measurement that has been started through [HS4xxx↵\\_MeasureStart\(\)](#) previously. If the checksum computation is correct the result is stored in the *results* data structure. Otherwise the contents of *results* is left unmodified and an error code is returned.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>results</i>	Pointer to a data structure, to store results in

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

4.4.3.5 HS4xxx\_MeasureStart()

```
int HS4xxx_MeasureStart (
```

```
HSxxxx_t * sensor )
```

Start a temperature/humidity measurement cycle in non-hold mode.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
---------------	---

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

4.4.3.6 HS4xxx\_ReadID()

```
int HS4xxx_ReadID (
    HSxxxx_t * sensor,
    uint32_t * id )
```

Read the unique sensor ID of the HS4xxx.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>id</i>	Pointer buffer to store the ID

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

## 4.5 HS3xxx Sensor API

HS3xxx Temperature/Humidity Sensor API.

### Files

- file [hs3xxx.h](#)  
*HS3xxx sensor declarations.*

### Functions

- int [HS3xxx\\_Init](#) ([HSxxxx\\_t](#) \*sensor, [Interface\\_t](#) \*hal)  
*Initialize the sensor object.*
- int [HS3xxx\\_ReadID](#) ([HSxxxx\\_t](#) \*sensor, uint32\_t \*id)  
*Read the unique sensor ID of the HS3xxx.*
- int [HS3xxx\\_Measure](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Execute a temperature/humidity measurement cycle.*
- int [HS3xxx\\_MeasureStart](#) ([HSxxxx\\_t](#) \*sensor)  
*Start a temperature/humidity measurement cycle.*
- int [HS3xxx\\_MeasureRead](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Read temperature/humidity results.*

### Enumerations

- enum [HTErrors\\_t](#) { [hteStaleData](#) = 1 }  
*Error code definitions specific for the HS3xxx API.*

#### 4.5.1 Detailed Description

HS3xxx Temperature/Humidity Sensor API.

HS3xxx is a series of highly-accurate, fully-calibrated relative humidity and temperature sensors.

The HS3xxx API provides convenient access to the temperature & humidity measurement capabilities of the sensor.

#### 4.5.2 Enumeration Type Documentation

##### 4.5.2.1 HTErrors\_t

enum [HTErrors\\_t](#)

Error code definitions specific for the HS3xxx API.



Enumerator

hteStaleData	Sensor reported stale data.
--------------	-----------------------------

4.5.3 Function Documentation

4.5.3.1 HS3xxx\_Init()

```
int HS3xxx_Init (
    HSxxxx_t * sensor,
    Interface_t * hal )
```

Initialize the sensor object.

This function tries accessing the HS3xxx I2C address using *hal* as hardware interface. On success, the sensor object is initialized and can be used afterwards. Otherwise, an error code is returned and the sensor object is not usable.

Note

The HS3xxx API requires that the HAL interface object has the [Interface\\_t::i2cRead](#) and [Interface\\_t::i2cWrite](#) members defined. In addition, [HS3xxx\\_Measure\(\)](#) requires the [Interface\\_t::msSleep](#) member, however, the check for this function is done in [HS3xxx\\_Measure\(\)](#) and not in [HS3xxx\\_Init\(\)](#).

Parameters

<i>sensor</i>	Pointer to sensor object to be initialized
<i>hal</i>	Pointer to HAL object providing physical communication

Returns

int Error code

Return values

0	On success
other	On error

4.5.3.2 HS3xxx\_Measure()

```
int HS3xxx_Measure (
    HSxxxx_t * sensor,
    HSxxxx_Results_t * results )
```

Execute a temperature/humidity measurement cycle.

This function starts a measurement cycle in non-hold mode, waits for the result to be available and reads it. This is a convenience function which calls `HS3xxx_MeasureStart()` and `HS3xxx_MeasureRead()` with a delay between both calls, to allow the sensor to complete its measurement.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>results</i>	Pointer to a data structure, to store results in

Returns

int Error code

Return values

0	On success
other	On error

4.5.3.3 HS3xxx\_MeasureRead()

```
int HS3xxx_MeasureRead (
    HSxxxx_t * sensor,
    HSxxxx_Results_t * results )
```

Read temperature/humidity results.

This function reads the temperature/humidity results from a measurement that has been started through `HS3xxx_MeasureStart()` previously.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>results</i>	Pointer to a data structure, to store results in

Returns

int Error code

Return values

0	On success
other	On error

4.5.3.4 HS3xxx\_MeasureStart()

```
int HS3xxx_MeasureStart (
    HSxxxx_t * sensor )
```

Start a temperature/humidity measurement cycle.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
---------------	---

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

4.5.3.5 HS3xxx\_ReadID()

```
int HS3xxx_ReadID (
    HSxxxx_t * sensor,
    uint32_t * id )
```

Read the unique sensor ID of the HS3xxx.

Parameters

<i>sensor</i>	Pointer to an initialized sensor object
<i>id</i>	Pointer buffer to store the ID

Returns

int Error code

Return values

0	On success
<i>other</i>	On error

## Chapter 5

# File Documentation

### 5.1 /builds/industrial/zmod4xxx/zmod6xxx\_generic\_example/hal/custom/template.c File Reference

This file serves as a template for a customer specific HAL.

#### Functions

- static void [\\_Sleep](#) (uint32\_t ms)
- static int [\\_I2CRead](#) (void \*handle, uint8\_t slAddr, uint8\_t \*wrData, int wrLen, uint8\_t \*rdData, int rdLen)
- static int [\\_I2CWrite](#) (void \*handle, uint8\_t slAddr, uint8\_t \*wrData1, int wrLen1, uint8\_t \*wrData2, int wrLen2)
- static int [\\_Reset](#) ()
- int [HAL\\_Init](#) ([Interface\\_t](#) \*hal)  
*Initialize hardware and populate [Interface\\_t](#) object.*
- int [HAL\\_Deinit](#) ([Interface\\_t](#) \*hal)  
*Cleanup before program exit.*
- void [HAL\\_HandleError](#) (int errorCode, void const \*context)  
*Error handling function.*

#### 5.1.1 Function Documentation

### 5.1.1.1 \_I2CRead()

```
static int _I2CRead (
    void * handle,
    uint8_t slAddr,
    uint8_t * wrData,
    int wrLen,
    uint8_t * rdData,
    int rdLen ) [static]
```

The implementation of this function shall read data from I2C, strictly following the procedure below. If wrLen is not zero, the read must be preceded by an I2C write.

- I2C Start
- if wrLen != 0:
  - Send: slAddr + WRITE
  - Send: wrLen bytes from wrData
  - I2C Restart ( no stop before this start condition! )
- Send: SlaveAddress + READ
- Receive: rdLen bytes into rdData
- I2CStop

### 5.1.1.2 \_I2CWrite()

```
static int _I2CWrite (
    void * handle,
    uint8_t slAddr,
    uint8_t * wrData1,
    int wrLen1,
    uint8_t * wrData2,
    int wrLen2 ) [static]
```

The implementation of this function shall send the data provided in arguments wrData1 and wrData2 over the I2C bus. Both data blocks shall be sent directly after each other without a new start/restart condition.

- I2C Start
- Send: slAddr + WRITE
- if wrLen1 != 0:
  - Send: wrLen1 bytes from wrData1
- if wrLen2 != 0;
  - Send: wrLen2 bytes from wrData2
- I2CStop

### 5.1.1.3 \_Reset()

```
static int _Reset ( ) [static]
```

This function shall pulse the reset pin of the RRH46410.

#### Note

Refer to the datasheet of the sensor(s) being interfaced for reset timing requirements.

### 5.1.1.4 \_Sleep()

```
static void _Sleep (
    uint32_t ms ) [static]
```

This function should do nothing but returning after the number of milliseconds passed in the 'ms' argument.

## 5.2 /builds/industrial/zmod4xxx/zmod6xxx\_generic\_example/hal/hal.h File Reference

Renesas Environmental Sensor HAL definitions.

### Data Structures

- struct [Interface\\_t](#)  
A structure of pointers to hardware specific functions. [More...](#)

### Typedefs

- typedef int(\* [I2CImpl\\_t](#)) (void \*, uint8\_t, uint8\_t \*, int, uint8\_t \*, int)  
Function pointer type defining signature of I2C functions.
- typedef char const \*(\* [ErrorStringGenerator\\_t](#)) (int, int, char \*, int)  
Function type used for generation of error strings.

### Enumerations

- enum [GenericError\\_t](#) { [ecSuccess](#) = 0, [ecHALError](#) = 0x10000 }  
Error code definitions.
- enum [ErrorScope\\_t](#) { [esSensor](#) = 0x00000000, [esAlgorithm](#) = 0x10000000, [esInterface](#) = 0x20000000, [esHAL](#) = 0x30000000 }  
Success status code and error scopes.
- enum [HALError\\_t](#) {  
    [heNoInterface](#) = 1, [heNotImplemented](#), [heI2CReadMissing](#), [heI2CWriteMissing](#),  
    [heSleepMissing](#), [heResetMissing](#) }  
HAL scope error definitions.

## Functions

- int [HAL\\_Init](#) ([Interface\\_t](#) \*hal)  
*Initialize hardware and populate [Interface\\_t](#) object.*
- int [HAL\\_Deinit](#) ([Interface\\_t](#) \*hal)  
*Cleanup before program exit.*
- void [HAL\\_HandleError](#) (int errorCode, void const \*context)  
*Error handling function.*
- int [HAL\\_SetError](#) (int error, int scope, [ErrorStringGenerator\\_t](#) errStrFn)  
*Function storing error information.*
- char const \* [HAL\\_GetErrorInfo](#) (int \*error, int \*scope, char \*str, int bufSize)  
*Get detailed information for last error.*
- char const \* [HAL\\_GetErrorString](#) (int error, int scope, char \*str, int bufSize)  
*Error string generator for HAL-scoped errors.*

## 5.3 /builds/industrial/zmod4xxx/zmod6xxx\_generic\_example/sensors/hs3xxx.h File Reference

HS3xxx sensor declarations.

## Enumerations

- enum [HTErrors\\_t](#) { [hteStaleData](#) = 1 }  
*Error code definitions specific for the HS3xxx API.*

## Functions

- int [HS3xxx\\_Init](#) ([HSxxxx\\_t](#) \*sensor, [Interface\\_t](#) \*hal)  
*Initialize the sensor object.*
- int [HS3xxx\\_ReadID](#) ([HSxxxx\\_t](#) \*sensor, uint32\_t \*id)  
*Read the unique sensor ID of the HS3xxx.*
- int [HS3xxx\\_Measure](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Execute a temperature/humidity measurement cycle.*
- int [HS3xxx\\_MeasureStart](#) ([HSxxxx\\_t](#) \*sensor)  
*Start a temperature/humidity measurement cycle.*
- int [HS3xxx\\_MeasureRead](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Read temperature/humidity results.*

## 5.4 /builds/industrial/zmod4xxx/zmod6xxx\_generic\_example/sensors/hs4xxx.h File Reference

HS4xxx sensor declarations.

## Enumerations

- enum [HS4xxx\\_ErrorCodes\\_t](#) { [hteHS4xxxCRCErr](#) = 1 }  
*Error code definitions specific for the HS4xxx API.*

## Functions

- int [HS4xxx\\_Init](#) ([HSxxxx\\_t](#) \*sensor, [Interface\\_t](#) \*hal)  
*Initialize the sensor object.*
- int [HS4xxx\\_ReadID](#) ([HSxxxx\\_t](#) \*sensor, uint32\_t \*id)  
*Read the unique sensor ID of the HS4xxx.*
- int [HS4xxx\\_Measure](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Perform a temperature/humidity measurement cycle.*
- int [HS4xxx\\_MeasureHold](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Perform a temperature/humidity measurement cycle using hold mode.*
- int [HS4xxx\\_MeasureStart](#) ([HSxxxx\\_t](#) \*sensor)  
*Start a temperature/humidity measurement cycle in non-hold mode.*
- int [HS4xxx\\_MeasureRead](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Read temperature/humidity results in non-hold mode.*

## 5.5 /builds/industrial/zmod4xxx/zmod6xxx\_generic\_example/sensors/hsxxxx.h File Reference

Renesas humidity sensors (HS3xxx, HS4xxx) abstraction.

### Data Structures

- struct [HSxxxx\\_Results\\_t](#)  
*Data structure holding humidity/temperature results. [More...](#)*
- struct [HSxxxx\\_t](#)  
*Data structure holding information required for HSxxxx API operation. [More...](#)*

### Functions

- int [HSxxxx\\_Init](#) ([HSxxxx\\_t](#) \*sensor, [Interface\\_t](#) \*hal)  
*Initialize the sensor object.*
- int [HSxxxx\\_Measure](#) ([HSxxxx\\_t](#) \*sensor, [HSxxxx\\_Results\\_t](#) \*results)  
*Perform one temperature measurement.*
- char const \* [HSxxxx\\_Name](#) ([HSxxxx\\_t](#) \*sensor)  
*Return the temperature/humidity sensor type name.*

## 5.6 /builds/industrial/zmod4xxx/zmod6xxx\_generic\_example/sensors/rrh46410.h File Reference

RRH46410 API declarations.



## Data Structures

- struct [RRH46410\\_Common\\_Results\\_t](#)  
*This struct helps accessing sample ID independent from operating mode. [More...](#)*
- struct [RRH46410\\_IQ2\\_Results\\_t](#)  
*Results returned from sensor in IQ2 and IQ2 ULP mode. [More...](#)*
- struct [RRH46410\\_PBAQ\\_Results\\_t](#)  
*Raw result returned from sensor in PBAQ mode. [More...](#)*
- union [RRH46410\\_Results\\_t](#)  
*Generic result data structure, joining the results of the different operating modes. [More...](#)*
- union [RRH46410\\_Debug\\_t](#)  
*Debug output data structure. [More...](#)*
- struct [RRH46410\\_t](#)  
*Data structure holding information required for RRH46410 API operation. [More...](#)*
- struct [RRH46410\\_SensorInfo\\_t](#)  
*Sensor information data structure. [More...](#)*

## Enumerations

- enum [RRH46410\\_OperatingMode\\_t](#) {  
[omSuspend](#) = 0x00, [omIAQ2](#) = 0x01, [omIAQ2\\_ULP](#) = 0x02, [omPBAQ](#) = 0x05,  
[omCleaning](#) = 0x80, [omUnknown](#) = 0xff }  
*Operating mode types.*
- enum [RRH46410\\_Commands\\_t](#) {  
[cmdGetSensorInfo](#) = 0x00, [cmdGetTrackingNumber](#) = 0x01, [cmdGetOperatingMode](#) = 0x10, [cmdSetOperatingMode](#) = 0x11,  
[cmdSetHumidity](#) = 0x12, [cmdGetResults](#) = 0x18, [cmdSetGPIOConfig](#) = 0x20, [cmdGetGPIOState](#) = 0x21,  
[cmdSetOutputHigh](#) = 0x22, [cmdSetOutputLow](#) = 0x23, [cmdGetDebugData](#) = 0x31, [cmdSoftwareReset](#) = 0x8f  
}  
*Define names for RRH46410 commands.*
- enum [RRH46410\\_ErrorCodes\\_t](#) {  
[ecWarmup](#) = 0x01, [ecCleaningCountExceeded](#) = 0x08, [ecPOR](#) = 0x09, [ecDamage](#) = 0x0a,  
[ecDataNotReady](#) = 0x10, [ecInternalComm](#) = 0x20, [ecHostToSensorChecksum](#) = 0x40, [ecInvalidCommand](#) = 0x80,  
[ecSampleNotNew](#) = 0x100, [ecSensorToHostChecksum](#) = 0x101, [ecCleaningTimeout](#) = 0x102, [ecGPIOConfigInvalid](#) = 0x103 }  
*Definition of error codes for RRH46410 sensors.*

## Functions

- int [RRH46410\\_Init](#) ([RRH46410\\_t](#) \*sensor, [Interface\\_t](#) \*hal)  
*Initialize sensor object.*
- int [RRH46410\\_GetSensorInfo](#) ([RRH46410\\_t](#) \*sensor, [RRH46410\\_SensorInfo\\_t](#) \*info)  
*Read additional sensor information.*
- int [RRH46410\\_ReadOperatingMode](#) ([RRH46410\\_t](#) \*sensor)  
*Read the current operating mode.*
- int [RRH46410\\_SetOperatingMode](#) ([RRH46410\\_t](#) \*sensor, [RRH46410\\_OperatingMode\\_t](#) mode)  
*Set the current operating mode.*
- int [RRH46410\\_SetHumidity](#) ([RRH46410\\_t](#) \*sensor, float humidity)

*Set humidity value to be used in gas algorithm computations.*

- int [RRH46410\\_ReadResults](#) (RRH46410\_t \*sensor, void \*result)

*Read latest sensor results.*

- int [RRH46410\\_ReadDebug](#) (RRH46410\_t \*sensor, void \*result)

*Read debug output.*

- int [RRH46410\\_PerformCleaning](#) (RRH46410\_t \*sensor)

*Perform sensor cleaning.*

- int [RRH46410\\_Reset](#) (RRH46410\_t \*sensor)

*Reset the sensor.*

- int [RRH46410\\_SoftReset](#) (RRH46410\_t \*sensor)

*Perform a software reset.*

- int [RRH46410\\_ConfigureGPIO](#) (RRH46410\_t \*sensor, uint8\_t outputMask)

*Configure GPIO1 and GPIO2 direction.*

- int [RRH46410\\_WriteGPIO](#) (RRH46410\_t \*sensor, uint8\_t mask, uint8\_t state)

*Write one or multiple GPIO pins.*

- int [RRH46410\\_ReadGPIO](#) (RRH46410\_t \*sensor, uint8\_t \*state)

*Read the state of all GPIO pins.*

## IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES ("RENESAS") PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.0 Mar 2020)

### Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,  
Koto-ku, Tokyo 135-0061, Japan  
[www.renesas.com](http://www.renesas.com)

### Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:  
[www.renesas.com/contact/](http://www.renesas.com/contact/)

### Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.