



# AS7050 Application Manager

API Documentation  
revision v4.1.0

Generated by Doxygen

## Contents

<b>1 Module Documentation</b>	<b>1</b>
1.1 Application Manager . . . . .	1
1.1.1 Detailed Description . . . . .	2
1.1.2 Macro Definition Documentation . . . . .	4
1.1.3 Typedef Documentation . . . . .	4
1.1.4 Function Documentation . . . . .	5
1.2 Supported Vital Signs Applications . . . . .	12
1.2.1 Detailed Description . . . . .	12
1.2.2 Typedef Documentation . . . . .	12
1.2.3 Enumeration Type Documentation . . . . .	13
1.3 Application Flags . . . . .	15
1.3.1 Detailed Description . . . . .	15
1.3.2 Macro Definition Documentation . . . . .	15
1.4 Raw Data Streaming . . . . .	16
1.4.1 Detailed Description . . . . .	16
1.5 ams HRM Algorithm . . . . .	17
1.5.1 Detailed Description . . . . .	17
1.5.2 Macro Definition Documentation . . . . .	17
1.5.3 Enumeration Type Documentation . . . . .	17
1.6 ams SpO2 Algorithm . . . . .	18
1.6.1 Detailed Description . . . . .	18
1.6.2 Enumeration Type Documentation . . . . .	18
1.7 Galvanic Skin Resistance . . . . .	19
1.7.1 Detailed Description . . . . .	19
1.7.2 Enumeration Type Documentation . . . . .	19
1.8 Error Codes . . . . .	20
1.8.1 Detailed Description . . . . .	21
1.8.2 Typedef Documentation . . . . .	21
1.8.3 Enumeration Type Documentation . . . . .	21
<b>2 Data Structure Documentation</b>	<b>23</b>
2.1 as7050_appmgr_version_t Struct Reference . . . . .	23
2.1.1 Detailed Description . . . . .	23
2.1.2 Field Documentation . . . . .	23
2.2 bio_hrm_a0_configuration_t Struct Reference . . . . .	23
2.2.1 Detailed Description . . . . .	23
2.2.2 Field Documentation . . . . .	24
2.3 bio_hrm_a0_output_t Struct Reference . . . . .	24
2.3.1 Detailed Description . . . . .	24
2.3.2 Field Documentation . . . . .	24
2.4 bio_spo2_a0_configuration_t Struct Reference . . . . .	25
2.4.1 Detailed Description . . . . .	25
2.4.2 Field Documentation . . . . .	25
2.5 bio_spo2_a0_output_t Struct Reference . . . . .	26
2.5.1 Detailed Description . . . . .	26
2.5.2 Field Documentation . . . . .	26
2.6 gsr_app_configuration_t Struct Reference . . . . .	27
2.6.1 Detailed Description . . . . .	27
2.6.2 Field Documentation . . . . .	27
2.7 gsr_app_output_t Struct Reference . . . . .	28
2.7.1 Detailed Description . . . . .	28
2.7.2 Field Documentation . . . . .	28

2.8 raw_app_configuration_t Struct Reference . . . . .	28
2.8.1 Detailed Description . . . . .	28
2.8.2 Field Documentation . . . . .	28

## 1 Module Documentation

### 1.1 Application Manager

The Application Manager is a component of the AS7050 Software Support Package that handles and executes all Vital Signs Applications in a platform independent way. It manages the execution of Vital Signs Applications and passes Chip Library FIFO data, accelerometer measurement data, and AGC status information to the individual Vital Signs Applications.

#### Data Structures

- struct [as7050\\_appmgr\\_version\\_t](#)

#### Macros

- #define AS7050\_APPMGR\_VER\_MAJOR 4
- #define AS7050\_APPMGR\_VER\_MINOR 1
- #define AS7050\_APPMGR\_VER\_PATCH 0

#### Typedefs

- typedef uint8\_t [as7050\\_appmgr\\_channel\\_id\\_t](#)

#### Functions

- [err\\_code\\_t as7050\\_appmgr\\_initialize \(void\)](#)  
*Initializes the Application Manager.*
- [err\\_code\\_t as7050\\_appmgr\\_enable\\_apps \(uint32\\_t enabled\\_apps\)](#)  
*Sets the enabled Vital Signs Applications.*
- [err\\_code\\_t as7050\\_appmgr\\_set\\_signal\\_routing \(as7050\\_appmgr\\_app\\_id\\_t app, const as7050\\_appmgr\\_channel\\_id\\_t \\*p\\_channels, uint8\\_t channels\\_num\)](#)  
*Sets the signal routing for a Vital Signs Application.*
- [err\\_code\\_t as7050\\_appmgr\\_configure\\_app \(as7050\\_appmgr\\_app\\_id\\_t app, const void \\*p\\_config, uint8\\_t size\)](#)  
*Configures a Vital Signs Application.*
- [err\\_code\\_t as7050\\_appmgr\\_start\\_processing \(as7050\\_meas\\_config\\_t measurement\\_config, uint32\\_t acc\\_sample\\_period\\_us, const as7050\\_appmgr\\_channel\\_id\\_t \\*p\\_agc\\_mappings, uint8\\_t agc\\_mappings\\_num\)](#)  
*Starts processing.*
- [err\\_code\\_t as7050\\_appmgr\\_set\\_input \(const uint8\\_t \\*p\\_fifo\\_data, uint16\\_t fifo\\_data\\_size, const agc\\_status\\_t \\*p\\_agc\\_statuses, uint8\\_t agc\\_statuses\\_num, const vs\\_acc\\_data\\_t \\*p\\_acc\\_samples, uint16\\_t acc\\_samples\\_num, uint32\\_t \\*p\\_ready\\_for\\_execution\)](#)  
*Provides measurement data to the Application Manager.*
- [err\\_code\\_t as7050\\_appmgr\\_set\\_ext\\_event\\_occurred \(void\)](#)  
*Informs the Application Manager that an external event occurred.*
- [err\\_code\\_t as7050\\_appmgr\\_execute \(uint32\\_t \\*p\\_data\\_available\)](#)

*Executes enabled Vital Signs Applications.*

- `err_code_t as7050_appmgr_get_output (as7050_appmgr_app_id_t app, void *p_dest, uint16_t *p_size)`

*Writes output of a Vital Signs Application to a buffer provided by the caller.*

- `err_code_t as7050_appmgr_stop_processing (void)`

*Stops processing.*

- `err_code_t as7050_appmgr_shutdown (void)`

*De-initializes the Application Manager.*

- `err_code_t as7050_appmgr_get_version (as7050_appmgr_version_t *p_version)`

*Gets the version of the Application Manager.*

### 1.1.1 Detailed Description

The Application Manager is a component of the AS7050 Software Support Package that handles and executes all Vital Signs Applications in a platform independent way. It manages the execution of Vital Signs Applications and passes Chip Library FIFO data, accelerometer measurement data, and AGC status information to the individual Vital Signs Applications.

The AS7050 Application Manager routes measurement data to Vital Signs Applications and provides a unified interface for application configuration and obtaining resulting output data.

On a high-level overview, the component is used as follows:

- After initialization, the component is configured based on sensor configuration and based on the Vital Signs Applications that shall be executed.
- After the configuration has been set, the component enters the Processing state.
- While in the Processing state, measurement data from the Chip Library, accelerometer measurement data, and AGC status information can be provided to the component at any time. To each enabled Vital Signs Application, the required subset of this data is forwarded. Vital Signs Applications queue the data internally for processing and indicate whether enough data has been received for execution.
- A function to execute enabled Vital Signs Applications can be called anytime while in the Processing state. The Vital Signs Applications process the queued input data and generate output data, that is also for output internally by the corresponding Vital Signs Application. It is indicated to the caller of the function whether output has been generated.
- Queued output data can be collected via separate function, that can also be called anytime while in the Processing state.
- After the measurement has been stopped, the component exits the Processing state. The configuration of the component can now be updated again.

This design allows for efficient execution of the Vital Signs Applications on any platform. Basic embedded systems can call the functions of the Application Manager in a simple super loop, while more advanced systems can call use the Application Manager from multiple threads.

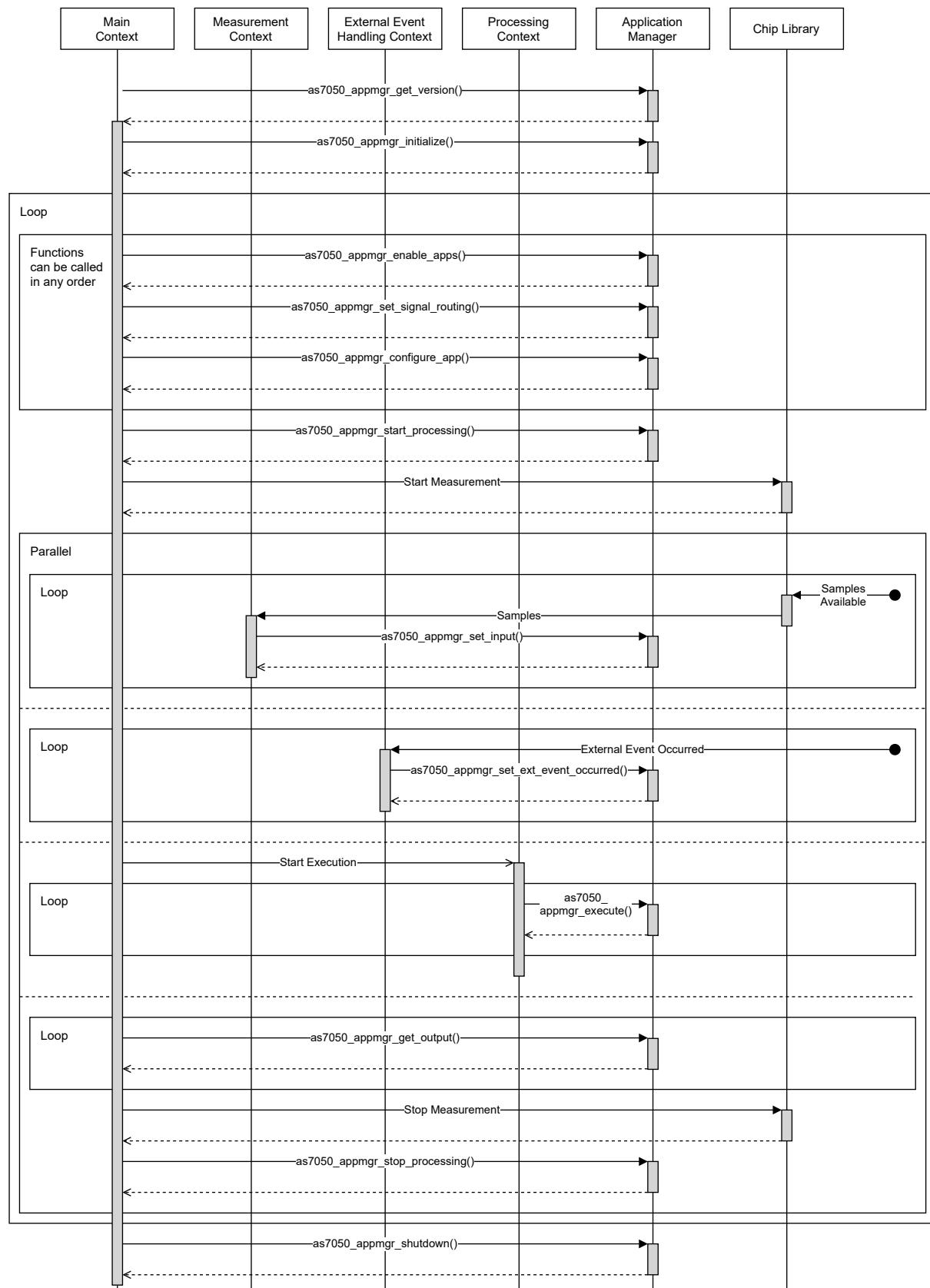


Figure 1 Sequence Diagram

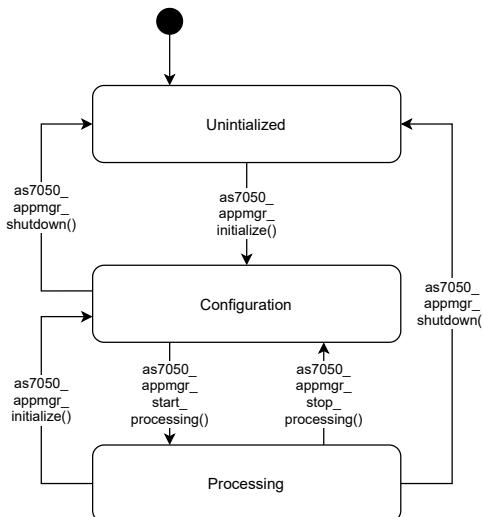


Figure 2 State Diagram

### 1.1.2 Macro Definition Documentation

#### 1.1.2.1 AS7050\_APPMGR\_VER\_MAJOR `#define AS7050_APPMGR_VER_MAJOR 4`

Major version of the Application Manager.

#### 1.1.2.2 AS7050\_APPMGR\_VER\_MINOR `#define AS7050_APPMGR_VER_MINOR 1`

Minor version of the Application Manager.

#### 1.1.2.3 AS7050\_APPMGR\_VER\_PATCH `#define AS7050_APPMGR_VER_PATCH 0`

Patch version of the Application Manager.

### 1.1.3 Typedef Documentation

#### 1.1.3.1 as7050\_appmgr\_channel\_id\_t `typedef uint8_t as7050_appmgr_channel_id_t`

Chip-specific type representing the identifier of a channel.

## 1.1.4 Function Documentation

**1.1.4.1 as7050\_appmgr\_initialize()** `err_code_t as7050_appmgr_initialize ( void )`

Initializes the Application Manager.

The Application Manager transitions to Configuration state after initialization.

## Return values

<code>ERR_SUCCESS</code>	Initialized successfully.
--------------------------	---------------------------

**1.1.4.2 `as7050_appmgr_enable_apps()`** `err_code_t as7050_appmgr_enable_apps (`  
`uint32_t enabled_apps )`

Sets the enabled Vital Signs Applications.

This function can only be called when the Application Manager is in Configuration state.

## Parameters

in	<code>enabled_apps</code>	Flags of enabled Vital Signs Applications, see <a href="#">Application Flags</a> . A Vital Signs Application is enabled when the corresponding bit is set and disabled when the bit is not set. At least one app must be enabled. <a href="#">AS7050_APPMGR_APP_FLAG_HRM_A0</a> and <a href="#">AS7050_APPMGR_APP_FLAG_SPO2_A0</a> cannot be enabled at the same time.
----	---------------------------	--

## Return values

<code>ERR_SUCCESS</code>	Updated successfully.
<code>ERR_ARGUMENT</code>	Invalid selection of application.
<code>ERR_PERMISSION</code>	Invalid state.

**1.1.4.3 `as7050_appmgr_set_signal_routing()`** `err_code_t as7050_appmgr_set_signal_routing (`  
`as7050_appmgr_app_id_t app,`  
`const as7050_appmgr_channel_id_t * p_channels,`  
`uint8_t channels_num )`

Sets the signal routing for a Vital Signs Application.

Each Vital Signs Application has an ordered list of signals assigned, which can be found in [Supported Vital Signs Applications](#). When `as7050_appmgr_set_input` is called, the Application Manager extracts the samples of each signal of each enabled Vital Signs Application. In order to extract samples from the provided sensor FIFO data, the Application Manager requires a mapping between sensor channels and Vital Signs Application signals. This mapping is provided to the Application Manager for each Vital Signs Application using this function.

No signal routing is required for [AS7050\\_APPMGR\\_APP\\_ID\\_RAW](#).

This function can only be called when the Application Manager is in Configuration state.

## Parameters

in	<code>app</code>	Identifier of the Vital Signs Application for which the signal routing shall be set.
----	------------------	--

**Parameters**

in	<i>p_channels</i>	Pointer to the start of an array containing the channel identifiers used for each signal of the given Vital Signs Application, in the order specified in the subsections of <a href="#">Supported Vital Signs Applications</a> . Can be NULL if channels_num is zero.
in	<i>channels_num</i>	Number of items contained in the <i>p_channels</i> array. This number must be equal to the signal count of the given Vital Signs Application, which can be found in the subsections of <a href="#">Supported Vital Signs Applications</a> .

**Return values**

<i>ERR_SUCCESS</i>	Updated successfully.
<i>ERR_ARGUMENT</i>	Invalid application identifier or invalid channel identifiers.
<i>ERR_SIZE</i>	Mismatching number of channels.
<i>ERR_POINTER</i>	Invalid pointer argument value.
<i>ERR_PERMISSION</i>	Invalid state.

```
1.1.4.4 as7050_appmgr_configure_app() err_code_t as7050_appmgr_configure_app (
    as7050_appmgr_app_id_t app,
    const void * p_config,
    uint8_t size )
```

Configures a Vital Signs Application.

This function can only be called when the Application Manager is in Configuration state.

**Parameters**

in	<i>app</i>	Identifier of the Vital Signs Application to configure.
in	<i>p_config</i>	Pointer to the configuration structure for the given Vital Signs Application, which can be found in the subsections of <a href="#">Supported Vital Signs Applications</a> .
in	<i>size</i>	Size of the configuration structure.

**Return values**

<i>ERR_SUCCESS</i>	Updated successfully.
<i>ERR_ARGUMENT</i>	Invalid application identifier or mismatching configuration structure size.
<i>ERR_POINTER</i>	Invalid pointer argument value.
<i>ERR_PERMISSION</i>	Invalid state.

```
1.1.4.5 as7050_appmgr_start_processing() err_code_t as7050_appmgr_start_processing (
    as7050_meas_config_t measurement_config,
```

```
    uint32_t acc_sample_period_us,
    const as7050_appmgr_channel_id_t * p_agc_mappings,
    uint8_t agc_mappings_num )
```

Starts processing.

This function can only be called when the Application Manager is in Configuration state. The Application Manager transitions to Processing state when this function executes successfully.

#### Parameters

in	<i>measurement_config</i>	Measurement configuration used to acquire the data that will be provided to the Application Manager. This value is typically obtained from the Chip Library.
in	<i>acc_sample_period_us</i>	Sample period of the accelerometer in microseconds.
in	<i>p_agc_mappings</i>	Pointer to the start of an array containing the identifiers of the channels controlled by Automatic Gain Control (AGC). When providing AGC status information via <a href="#">as7050_appmgr_set_input</a> , the status information items must be ordered identically. For example, if <i>p_agc_mappings</i> [0] contains the identifier of the first channel, then the AGC status information for the first channel must be provided to <a href="#">as7050_appmgr_set_input</a> via the first item of the AGC status information array. Can be NULL if <i>agc_mappings_num</i> is zero.
in	<i>agc_mappings_num</i>	Number of items in the <i>p_agc_mappings</i> array.

#### Return values

<a href="#">ERR_SUCCESS</a>	Updated successfully.
<a href="#">ERR_CONFIG</a>	Measurement configuration incompatible with application configurations.
<a href="#">ERR_ARGUMENT</a>	Invalid accelerometer sample period.
<a href="#">ERR_PERMISSION</a>	Invalid state.

```
1.1.4.6 as7050_appmgr_set_input() err_code_t as7050_appmgr_set_input (
    const uint8_t * p_fifo_data,
    uint16_t fifo_data_size,
    const agc_status_t * p_agc_statuses,
    uint8_t agc_statuses_num,
    const vs_acc_data_t * p_acc_samples,
    uint16_t acc_samples_num,
    uint32_t * p_ready_for_execution )
```

Provides measurement data to the Application Manager.

This function can only be called when the Application Manager is in Processing state.

**Parameters**

in	<i>p_fifo_data</i>	Pointer to the start of the sensor FIFO data. This data is typically obtained from the Chip Library.
in	<i>fifo_data_size</i>	Size of the FIFO data.
in	<i>p_agc_statuses</i>	Pointer to the start of an array containing Automatic Gain Control (AGC) status information for each channel with enabled AGC. The order of the items must be identical to the order of items in the AGC mapping array provided to <a href="#">as7050_appmgr_start_processing</a> . Can be NULL if <i>agc_statuses_num</i> is zero.
in	<i>agc_statuses_num</i>	Number of AGC status information items. Must be equivalent to the number of AGC mappings provided to <a href="#">as7050_appmgr_start_processing</a> .
in	<i>p_acc_samples</i>	Pointer to the start of an array containing accelerometer samples.
in	<i>acc_samples_num</i>	Number of accelerometer samples.
out	<i>p_ready_for_execution</i>	Flags of Vital Signs Applications that indicated that they are ready for execution, see <a href="#">Application Flags</a> . An app is ready for execution when the corresponding bit is set.

**Return values**

<a href="#">ERR_SUCCESS</a>	Measurement data accepted.
<a href="#">ERR_SIZE</a>	Too many samples in the FIFO data or invalid number of AGC status information items.
<a href="#">ERR_OVERFLOW</a>	Signals of an app have sample counts which are too different to be handled.
<a href="#">ERR_POINTER</a>	Invalid pointer argument value.
<a href="#">ERR_PERMISSION</a>	Invalid state.

```
1.1.4.7 as7050_appmgr_set_ext_event_occurred() err_code_t as7050_appmgr_set_ext_event_occurred(
(
    void )
```

Informs the Application Manager that an external event occurred.

The function counts how many times it has been called and provides the count to applications on the next invocation of [as7050\\_appmgr\\_set\\_input](#). This function can only be called when the Application Manager is in Processing state.

**Return values**

<a href="#">ERR_SUCCESS</a>	Information accepted.
<a href="#">ERR_OVERFLOW</a>	Too many events occurred since last <a href="#">as7050_appmgr_set_input</a> call.
<a href="#">ERR_PERMISSION</a>	Invalid state.

```
1.1.4.8 as7050_appmgr_execute() err_code_t as7050_appmgr_execute (
    uint32_t * p_data_available )
```

Executes enabled Vital Signs Applications.

This function can only be called when the Application Manager is in Processing state.

#### Parameters

<code>out</code>	<code>p_data_available</code>	Flags of Vital Signs Applications that indicated that their execution generated output data, see <a href="#">Application Flags</a> . An app has output data available when the corresponding bit is set.
------------------	-------------------------------	--

#### Return values

<a href="#"><code>ERR_SUCCESS</code></a>	Execution successful.
<a href="#"><code>ERR_POINTER</code></a>	Invalid pointer argument value.
<a href="#"><code>ERR_PERMISSION</code></a>	Invalid state.

```
1.1.4.9 as7050_appmgr_get_output() err_code_t as7050_appmgr_get_output (
    as7050_appmgr_app_id_t app,
    void * p_dest,
    uint16_t * p_size )
```

Writes output of a Vital Signs Application to a buffer provided by the caller.

This function can only be called when the Application Manager is in Processing state.

#### Parameters

<code>in</code>	<code>app</code>	Identifier of the Vital Signs Application to get output from.
<code>out</code>	<code>p_dest</code>	Pointer to the buffer where the output shall be written to. The output is application-specific and is described in the subsections of <a href="#">Supported Vital Signs Applications</a> .
<code>in,out</code>	<code>p_size</code>	Pointer to the amount of memory allocated for the output data. The function updates the value pointed to with the actual size of the written output data.

#### Return values

<a href="#"><code>ERR_SUCCESS</code></a>	Output data write successful.
<a href="#"><code>ERR_ARGUMENT</code></a>	Invalid app identifier or disabled application.
<a href="#"><code>ERR_POINTER</code></a>	Invalid pointer argument value.
<a href="#"><code>ERR_PERMISSION</code></a>	Invalid state.

```
1.1.4.10 as7050_appmgr_stop_processing() err_code_t as7050_appmgr_stop_processing (
```

```
void )
```

Stops processing.

This function can only be called when the Application Manager is not in Uninitialized state.

#### Return values

<i>ERR_SUCCESS</i>	Stop successful.
<i>ERR_PERMISSION</i>	Invalid state.

**1.1.4.11 *as7050\_appmgr\_shutdown()*** *err\_code\_t as7050\_appmgr\_shutdown ( void )*

De-initializes the Application Manager.

#### Return values

<i>ERR_SUCCESS</i>	De-initialization successful.
--------------------	-------------------------------

**1.1.4.12 *as7050\_appmgr\_get\_version()*** *err\_code\_t as7050\_appmgr\_get\_version ( as7050\_appmgr\_version\_t \* p\_version )*

Gets the version of the Application Manager.

#### Parameters

<i>out</i>	<i>p_version</i>	Pointer to the location where the version shall be written to.
------------	------------------	--

#### Return values

<i>ERR_SUCCESS</i>	Version write successful.
<i>ERR_POINTER</i>	Invalid pointer argument value.

## 1.2 Supported Vital Signs Applications

### Modules

- Raw Data Streaming
- ams HRM Algorithm
- ams SpO2 Algorithm
- Galvanic Skin Resistance
- Application Flags

*These definitions are used by the `as7050_appmgr_enable_apps`, `as7050_appmgr_set_input`, and `as7050_appmgr_execute` functions. Use these definitions with bitwise operators to set, clear, or read the flag bit corresponding to a given Vital Signs Application.*

### TypeDefs

- `typedef uint8_t as7050_appmgr_app_id_t`

### Enumerations

- `enum as7050_appmgr_app_id {  
 AS7050_APPMGR_APP_ID_RAW = 0,  
 AS7050_APPMGR_APP_ID_HRM_A0,  
 AS7050_APPMGR_APP_ID_SPO2_A0,  
 AS7050_APPMGR_APP_ID_GSR,  
 AS7050_APPMGR_APP_ID_NUM }  
}`

#### 1.2.1 Detailed Description

The Application Manager supports four Vital Signs Applications:

- Raw Data Streaming
- ams HRM Algorithm
- ams SpO2 Algorithm
- Galvanic Skin Resistance

The Application Manager provides a common API to all supported Vital Signs Applications. Each application is assigned an `as7050_appmgr_app_id` identifier, which is used in the Application Manager APIs.

#### 1.2.2 Typedef Documentation

**1.2.2.1 `as7050_appmgr_app_id_t`** `typedef uint8_t as7050_appmgr_app_id_t`

Type for `as7050_appmgr_app_id`.

**1.2.3 Enumeration Type Documentation****1.2.3.1 `as7050_appmgr_app_id`** `enum as7050_appmgr_app_id`

Identifiers of Vital Signs Applications.

**Enumerator**

AS7050_APPMGR_APP_ID_RAW	Identifies the <a href="#">Raw Data Streaming</a> application.
AS7050_APPMGR_APP_ID_HRM_A0	Identifies the <a href="#">ams HRM Algorithm</a> application.
AS7050_APPMGR_APP_ID_SPO2_A0	Identifies the <a href="#">ams SpO2 Algorithm</a> application.
AS7050_APPMGR_APP_ID_GSR	Identifies the <a href="#">Galvanic Skin Resistance</a> application.
AS7050_APPMGR_APP_ID_NUM	Number of Vital Signs Applications.

## 1.3 Application Flags

These definitions are used by the [as7050\\_appmgr\\_enable\\_apps](#), [as7050\\_appmgr\\_set\\_input](#), and [as7050\\_appmgr\\_execute](#) functions. Use these definitions with bitwise operators to set, clear, or read the flag bit corresponding to a given Vital Signs Application.

### Macros

- `#define M_AS7050_APPMGR_APP_FLAG(app) (1 << (app))`
- `#define AS7050_APPMGR_APP_FLAG_RAW M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_RAW)`
- `#define AS7050_APPMGR_APP_FLAG_HRM_A0 M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_HRM_A0)`
- `#define AS7050_APPMGR_APP_FLAG_SPO2_A0 M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_SPO2_A0)`
- `#define AS7050_APPMGR_APP_FLAG_GSR M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_GSR)`

### 1.3.1 Detailed Description

These definitions are used by the [as7050\\_appmgr\\_enable\\_apps](#), [as7050\\_appmgr\\_set\\_input](#), and [as7050\\_appmgr\\_execute](#) functions. Use these definitions with bitwise operators to set, clear, or read the flag bit corresponding to a given Vital Signs Application.

### 1.3.2 Macro Definition Documentation

#### 1.3.2.1 M\_AS7050\_APPMGR\_APP\_FLAG `#define M_AS7050_APPMGR_APP_FLAG (app) (1 << (app))`

Macro to create a application flag bitmask value with the bit for the given [as7050\\_appmgr\\_app\\_id\\_t](#) set.

#### 1.3.2.2 AS7050\_APPMGR\_APP\_FLAG\_RAW `#define AS7050_APPMGR_APP_FLAG_RAW M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_RAW)`

Application flag bitmask value with the bit for [AS7050\\_APPMGR\\_APP\\_ID\\_RAW](#) set.

#### 1.3.2.3 AS7050\_APPMGR\_APP\_FLAG\_HRM\_A0 `#define AS7050_APPMGR_APP_FLAG_HRM_A0 M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_HRM_A0)`

Application flag bitmask value with the bit for [AS7050\\_APPMGR\\_APP\\_ID\\_HRM\\_A0](#) set.

#### 1.3.2.4 AS7050\_APPMGR\_APP\_FLAG\_SPO2\_A0 `#define AS7050_APPMGR_APP_FLAG_SPO2_A0 M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_SPO2_A0)`

Application flag bitmask value with the bit for [AS7050\\_APPMGR\\_APP\\_ID\\_SPO2\\_A0](#) set.

#### 1.3.2.5 AS7050\_APPMGR\_APP\_FLAG\_GSR `#define AS7050_APPMGR_APP_FLAG_GSR M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_GSR)`

Application flag bitmask value with the bit for [AS7050\\_APPMGR\\_APP\\_ID\\_GSR](#) set.

## 1.4 Raw Data Streaming

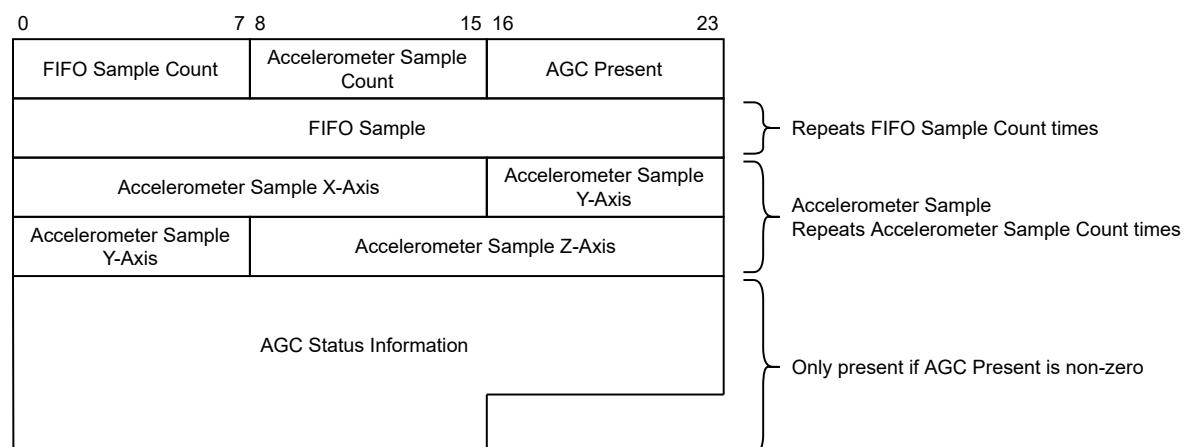
### Data Structures

- struct `raw_app_configuration_t`

#### 1.4.1 Detailed Description

The output of the raw app includes FIFO data as received from the AS7050 device, accelerometer samples, and Automatic Gain Correction (AGC) status information. Accelerometer samples are only included in the output when enabled in the `raw_app_configuration_t`. AGC status information is only transmitted when the status information of at least one AGC instance has changed since the last transmission. If AGC status information is transmitted, the output always contains the status information of each enabled AGC instance. The AGC status information entries are ordered in the same order as the AGC instances have been configured.

The output has the following format:



**Figure 3 Raw App Output**

The header fields (FIFO Sample Count, Accelerometer Sample Count, AGC Status Count) each have a size of 1 byte. Each FIFO Sample has a size of 3 bytes. The FIFO Sample field repeats FIFO Sample Count times. An Accelerometer Sample has a total size of 6 bytes. The Accelerometer Sample field repeats Accelerometer Sample Count times. An AGC Status Information has a size of 4 bytes and repeats AGC Status Count times.

## 1.5 ams HRM Algorithm

### Data Structures

- struct [bio\\_hrm\\_a0\\_configuration\\_t](#)
- struct [bio\\_hrm\\_a0\\_output\\_t](#)

### Macros

- `#define PRV_DATA_NUM 5`

### Enumerations

- enum [bio\\_hrm\\_a0\\_signal](#) {  
    [BIO\\_HRM\\_A0\\_SIGNAL\\_PPG](#) = 0,  
    [BIO\\_HRM\\_A0\\_SIGNAL\\_NUM](#) }

#### 1.5.1 Detailed Description

#### 1.5.2 Macro Definition Documentation

##### 1.5.2.1 PRV\_DATA\_NUM `#define PRV_DATA_NUM 5`

Maximum number of PRV data items in a single [bio\\_hrm\\_a0\\_output\\_t](#) structure.

#### 1.5.3 Enumeration Type Documentation

##### 1.5.3.1 bio\_hrm\_a0\_signal enum [bio\\_hrm\\_a0\\_signal](#)

Sensor signals provided to the HRM bio app.

#### Enumerator

<a href="#">BIO_HRM_A0_SIGNAL_PPG</a>	PPG signal.
<a href="#">BIO_HRM_A0_SIGNAL_NUM</a>	Number of sensor signals provided to the bio app.

## 1.6 ams SpO2 Algorithm

### Data Structures

- struct [bio\\_spo2\\_a0\\_configuration\\_t](#)
- struct [bio\\_spo2\\_a0\\_output\\_t](#)

### Enumerations

- enum [bio\\_spo2\\_a0\\_signal](#) {  
    BIO\_SPO2\_A0\_SIGNAL\_PPG\_RED = 0,  
    BIO\_SPO2\_A0\_SIGNAL\_PPG\_IR = 1,  
    BIO\_SPO2\_A0\_SIGNAL\_AMBIENT = 2,  
    BIO\_SPO2\_A0\_SIGNAL\_NUM }

#### 1.6.1 Detailed Description

#### 1.6.2 Enumeration Type Documentation

##### 1.6.2.1 [bio\\_spo2\\_a0\\_signal](#) enum [bio\\_spo2\\_a0\\_signal](#)

Sensor signals provided to the SpO2 bio app.

#### Enumerator

BIO_SPO2_A0_SIGNAL_PPG_RED	Red PPG signal.
BIO_SPO2_A0_SIGNAL_PPG_IR	Infrared PPG signal.
BIO_SPO2_A0_SIGNAL_AMBIENT	Ambient light signal.
BIO_SPO2_A0_SIGNAL_NUM	Number of sensor signals provided to the bio app.

## 1.7 Galvanic Skin Resistance

### Data Structures

- struct [gsr\\_app\\_configuration\\_t](#)
- struct [gsr\\_app\\_output\\_t](#)

### Enumerations

- enum [gsr\\_app\\_signal](#) {  
    [GSR\\_APP\\_SIGNAL\\_ECG](#) = 0,  
    [GSR\\_APP\\_SIGNAL\\_NUM](#) }

#### 1.7.1 Detailed Description

The output of the GSR app includes the measured Galvanic Skin Resistance value. For correct GSR measurement, the DAC reference value of the used AS7050 device needs to be determined using the corresponding function of the AS7050 Chip Library. The acquired DAC reference value needs to be provided to the GSR app via the app configuration.

#### 1.7.2 Enumeration Type Documentation

##### 1.7.2.1 [gsr\\_app\\_signal](#) enum [gsr\\_app\\_signal](#)

Sensor signals provided to the GSR app.

###### Enumerator

<a href="#">GSR_APP_SIGNAL_ECG</a>	ECG signal.
<a href="#">GSR_APP_SIGNAL_NUM</a>	Number of sensor signals provided to the GSR app.

## 1.8 Error Codes

### TypeDefs

- `typedef enum error_codes err_code_t`

### Enumerations

- `enum error_codes {  
 ERR_SUCCESS = 0,  
 ERR_PERMISSION = 1,  
 ERR_MESSAGE = 2,  
 ERR_MESSAGE_SIZE = 3,  
 ERR_POINTER = 4,  
 ERR_ACCESS = 5,  
 ERR_ARGUMENT = 6,  
 ERR_SIZE = 7,  
 ERR_NOT_SUPPORTED = 8,  
 ERR_TIMEOUT = 9,  
 ERR_CHECKSUM = 10,  
 ERR_OVERFLOW = 11,  
 ERR_EVENT = 12,  
 ERR_INTERRUPT = 13,  
 ERR_TIMER_ACCESS = 14,  
 ERR_LED_ACCESS = 15,  
 ERR_TEMP_SENSOR_ACCESS = 16,  
 ERR_DATA_TRANSFER = 17,  
 ERR_FIFO = 18,  
 ERR_OVER_TEMP = 19,  
 ERR_IDENTIFICATION = 20,  
 ERR_COM_INTERFACE = 21,  
 ERR_SYNCHRONISATION = 22,  
 ERR_PROTOCOL = 23,  
 ERR_MEMORY = 24,  
 ERR_THREAD = 25,  
 ERR_SPI = 26,  
 ERR_DAC_ACCESS = 27,  
 ERR_I2C = 28,  
 ERR_NO_DATA = 29,  
 ERR_SYSTEM_CONFIG = 30,  
 ERR_USB_ACCESS = 31,  
 ERR_ADC_ACCESS = 32,  
 ERR_SENSOR_CONFIG = 33,  
 ERR_SATURATION = 34,  
 ERR_MUTEX = 35,  
 ERR_ACCELEROMETER = 36,  
 ERR_CONFIG = 37,  
 ERR_BLE = 38,  
 ERR_FILE = 39,  
 ERR_DATA = 40 }`

### 1.8.1 Detailed Description

Generic error codes used by ams libraries.

### 1.8.2 Typedef Documentation

#### 1.8.2.1 `err_code_t` `typedef enum error_codes err_code_t`

This definition will be used for function return values.

### 1.8.3 Enumeration Type Documentation

#### 1.8.3.1 `error_codes` `enum error_codes`

Values represent the error codes.

##### Enumerator

<code>ERR_SUCCESS</code>	Normal return code if everything was successful executed.
<code>ERR_PERMISSION</code>	Operation not permitted
<code>ERR_MESSAGE</code>	Message is invalid. For example: <ul style="list-style-type: none"><li>• Message type is not supported</li><li>• incorrect crc</li><li>• ...</li></ul>
<code>ERR_MESSAGE_SIZE</code>	Message has the wrong size.
<code>ERR_POINTER</code>	Pointer is invalid. Can be a NULL Pointer or point to a wrong memory area.
<code>ERR_ACCESS</code>	Access denied
<code>ERR_ARGUMENT</code>	Invalid argument
<code>ERR_SIZE</code>	Argument size is too long or too short.
<code>ERR_NOT_SUPPORTED</code>	Function is not supported/implemented.
<code>ERR_TIMEOUT</code>	Got timeout while waiting for answer.
<code>ERR_CHECKSUM</code>	Checksum comparision failed.
<code>ERR_OVERFLOW</code>	Data overflow detected.
<code>ERR_EVENT</code>	Error to get or set an event. For example: <ul style="list-style-type: none"><li>• event queue is full or empty</li><li>• receive an unexpected event</li><li>• ...</li></ul>

## Enumerator

ERR_INTERRUPT	Error to get or set an interrupt. For example a interrupt resource is not available.
ERR_TIMER_ACCESS	Error while accessing timer periphery.
ERR_LED_ACCESS	Error while accessing LED periphery.
ERR_TEMP_SENSOR_ACCESS	Error while accessing temperature sensor.
ERR_DATA_TRANSFER	Communication error
ERR_FIFO	Faulty FIFO handling
ERR_OVER_TEMP	Overtemperature detected.
ERR_IDENTIFICATION	Sensor identification failed.
ERR_COM_INTERFACE	Generic communication interface error. For example: <ul style="list-style-type: none"><li>• communication interface is not available</li><li>• error during open or close an communication interface</li><li>• ...</li></ul>
ERR_SYNCHRONISATION	Synchronisation error, e.g. on protocol
ERR_PROTOCOL	Generic protocol error
ERR_MEMORY	Memory allocation error
ERR_THREAD	Thread can not created.
ERR_SPI	Error while accessing SPI periphery
ERR_DAC_ACCESS	Error while accessing DAC periphery.
ERR_I2C	Error while accessing I2C periphery.
ERR_NO_DATA	No data available.
ERR_SYSTEM_CONFIG	Error during system configuration. When a system resource is not available or generates an error for example.
ERR_USB_ACCESS	USB error
ERR_ADC_ACCESS	Error while accessing ADC periphery.
ERR_SENSOR_CONFIG	Error during sensor configuration.
ERR_SATURATION	Saturation detected
ERR_MUTEX	Error while mutex handling
ERR_ACCELEROMETER	Error while reading accelerometer data
ERR_CONFIG	Software component is not fully or correctly configured
ERR_BLE	Error while executing BLE stack function
ERR_FILE	Error during file access
ERR_DATA	Internal data is faulty

## 2 Data Structure Documentation

### 2.1 as7050\_appmgr\_version\_t Struct Reference

#### Data Fields

- uint8\_t major
- uint8\_t minor
- uint8\_t patch

#### 2.1.1 Detailed Description

Describes the version of the Application Manager.

#### 2.1.2 Field Documentation

##### 2.1.2.1 **major** uint8\_t as7050\_appmgr\_version\_t::major

Major version

##### 2.1.2.2 **minor** uint8\_t as7050\_appmgr\_version\_t::minor

Minor version

##### 2.1.2.3 **patch** uint8\_t as7050\_appmgr\_version\_t::patch

Patch version

### 2.2 bio\_hrm\_a0\_configuration\_t Struct Reference

#### Data Fields

- uint8\_t enable\_prv

#### 2.2.1 Detailed Description

Contains the configuration options of the HRM bio app.

## 2.2.2 Field Documentation

### 2.2.2.1 **enable\_prv** `uint8_t bio_hrm_a0_configuration_t::enable_prv`

PRV data is included in the output when set to TRUE.

## 2.3 **bio\_hrm\_a0\_output\_t** Struct Reference

### Data Fields

- `uint16_t heart_rate`
- `uint8_t quality`
- `uint8_t motion_frequency`
- `uint16_t prv_ms [PRV_DATA_NUM]`
- `uint8_t prv_ms_num`
- `uint8_t reserved`

### 2.3.1 Detailed Description

Describes the output of the HRM bio app.

### 2.3.2 Field Documentation

#### 2.3.2.1 **heart\_rate** `uint16_t bio_hrm_a0_output_t::heart_rate`

Contains the heart rate. The unit is 0.1 bpm.

#### 2.3.2.2 **quality** `uint8_t bio_hrm_a0_output_t::quality`

Contains information about the quality of the heart rate signal. A value of zero means best quality.

#### 2.3.2.3 **motion\_frequency** `uint8_t bio_hrm_a0_output_t::motion_frequency`

Contains the detected motion frequency. The unit is bpm. A value of zero means that no motion has been detected.

**2.3.2.4 prv\_ms** uint16\_t bio\_hrm\_a0\_output\_t::prv\_ms[PRV\_DATA\_NUM]

Contains PRV data. The unit is milliseconds.

**2.3.2.5 prv\_ms\_num** uint8\_t bio\_hrm\_a0\_output\_t::prv\_ms\_num

Contains the number of prv\_ms fields that are used.

**2.3.2.6 reserved** uint8\_t bio\_hrm\_a0\_output\_t::reserved

Padding byte.

## 2.4 bio\_spo2\_a0\_configuration\_t Struct Reference

### Data Fields

- uint16\_t **a**
- uint16\_t **b**
- uint16\_t **c**
- uint16\_t **dc\_comp\_red**
- uint16\_t **dc\_comp\_ir**

### 2.4.1 Detailed Description

Contains the configuration options of the SpO2 bio app.

### 2.4.2 Field Documentation

**2.4.2.1 a** uint16\_t bio\_spo2\_a0\_configuration\_t::a

Quadratic correction coefficient a.

**2.4.2.2 b** uint16\_t bio\_spo2\_a0\_configuration\_t::b

Quadratic correction coefficient b.

**2.4.2.3 c** uint16\_t bio\_spo2\_a0\_configuration\_t::c

Quadratic correction coefficient c.

**2.4.2.4 dc\_comp\_red** uint16\_t bio\_spo2\_a0\_configuration\_t::dc\_comp\_red

DC compensation for the red signal.

**2.4.2.5 dc\_comp\_ir** uint16\_t bio\_spo2\_a0\_configuration\_t::dc\_comp\_ir

DC compensation for the infrared signal.

## 2.5 bio\_spo2\_a0\_output\_t Struct Reference

### Data Fields

- uint8\_t status
- uint8\_t quality
- uint16\_t spo2
- uint16\_t heart\_rate
- uint16\_t pi
- uint16\_t average\_r
- uint16\_t ac\_comp\_red
- uint16\_t dc\_comp\_red
- uint16\_t ac\_comp\_ir
- uint16\_t dc\_comp\_ir

### 2.5.1 Detailed Description

Describes the output of the SpO2 bio app.

### 2.5.2 Field Documentation

**2.5.2.1 status** uint8\_t bio\_spo2\_a0\_output\_t::status

The value of the field is zero when the structure contains valid SpO2 data. It is one when no result is present in the data.

**2.5.2.2 quality** uint8\_t bio\_spo2\_a0\_output\_t::quality

Contains the quality of the signal. The unit is 1%.

**2.5.2.3 spo2** uint16\_t bio\_spo2\_a0\_output\_t::spo2

Contains the SpO2 measurement. The unit is 0.01%.

**2.5.2.4 heart\_rate** uint16\_t bio\_spo2\_a0\_output\_t::heart\_rate

Contains the heart rate. The unit is 0.1 bpm.

**2.5.2.5 pi** uint16\_t bio\_spo2\_a0\_output\_t::pi

Contains the Perfusion Index measurement. The unit is 0.01%.

**2.5.2.6 average\_r** uint16\_t bio\_spo2\_a0\_output\_t::average\_r

Contains the Average R value. The unit is 10000.

**2.5.2.7 ac\_comp\_red** uint16\_t bio\_spo2\_a0\_output\_t::ac\_comp\_red

Contains the AC component of the red PPG signal. It is used for calibration purposes.

**2.5.2.8 dc\_comp\_red** uint16\_t bio\_spo2\_a0\_output\_t::dc\_comp\_red

Contains the DC component of the red PPG signal. It is used for calibration purposes.

**2.5.2.9 ac\_comp\_ir** uint16\_t bio\_spo2\_a0\_output\_t::ac\_comp\_ir

Contains the AC component of the infrared PPG signal. It is used for calibration purposes.

**2.5.2.10 dc\_comp\_ir** uint16\_t bio\_spo2\_a0\_output\_t::dc\_comp\_ir

Contains the DC component of the infrared PPG signal. It is used for calibration purposes.

## 2.6 gsr\_app\_configuration\_t Struct Reference

### Data Fields

- uint32\_t [dac\\_ref](#)

### 2.6.1 Detailed Description

Configuration structure of the GSR app.

### 2.6.2 Field Documentation

**2.6.2.1 dac\_ref** uint32\_t gsr\_app\_configuration\_t::dac\_ref

DAC reference value which must be determined before.

## 2.7 gsr\_app\_output\_t Struct Reference

### Data Fields

- uint32\_t [resistor](#)

### 2.7.1 Detailed Description

Describes the output of the GSR app.

### 2.7.2 Field Documentation

**2.7.2.1 resistor** uint32\_t gsr\_app\_output\_t::resistor

Contains the resistance value in Ohm. The output is invalid if the value of this field is 0xFFFFFFFF.

## 2.8 raw\_app\_configuration\_t Struct Reference

### Data Fields

- uint8\_t [include\\_acc](#)

### 2.8.1 Detailed Description

Configuration structure of the raw app.

### 2.8.2 Field Documentation

**2.8.2.1 include\_acc** uint8\_t raw\_app\_configuration\_t::include\_acc

A non-zero value indicates that accelerometer data shall be included in the output data.