AS7352 ChipLib

API Documentation
revision v1.1.3

Generated by Doxygen

Contents

1 AS7352 ChipLib

1.1 Introduction

This document provides an overview on how the spectral sensor AS7352 can be controlled through the ChipLib (AS7352 Chip Library) and the corresponding OSAL (operating system abstraction layer).

The ChipLib's generic API allows for usage in several different fields of application. It is programmed in standard C language.

Key features:

- Standardized configuration via items
- Standardized measurement routine via callback handler
- Independent of hardware and platform
- Tested code source by ams
- Support of different measurement configurations
- Independent on register interface description

Limitations:

- Sensor calibration topics are not part of this library.
- The library provides only raw values.

The documentation is split into three sections:

- ChipLib: This describes how an application should handle the main library.
- OSAL: This section describes the adaptation of the ChipLib to other platforms.
- Module: Interface description of both APIs: ChipLib and OSAL

1.2 Overview

The ChipLib can directly be called by the user application and can be used without adaption. Hardware and platform dependencies must be specified separately in the OSAL. The OSAL interface is kept simple to keep integration efforts for the customers to a minimum. The simplest implementation would be a measurement routine without interrupt and without external peripherals. More information on the API functions can be found here: OSAL Functions. Details on how to implement application specific OSAL are described in OSAL

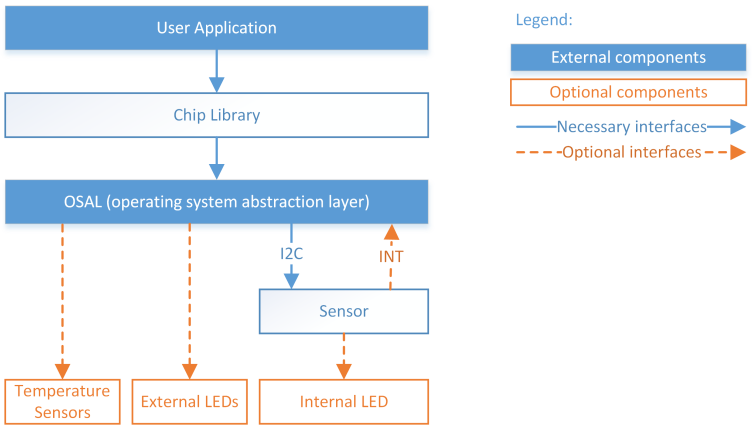

Figure 1 Structure Overview

- Internal LED: The AS7352 provides an internal LED driver to handle one LED.
- External LEDs: Further LEDs have to be specified in the OSAL. The ChipLib provides synchronized switching capabilities.
- Temperature sensors: The ChipLib allows for reading of external temperature sensor values synchronously to the AS7352 spectral measurement.
- Interrupt pin: To reduce system load, it is recommended to use the interrupt pin of the AS7352. Otherwise measurement times will be calculated by the ChipLib and status registers will be polled periodically.

1.3 Acronyms and Abbreviations

1.3.1 Acronyms

API = Application Programming Interface
I2C = Inter-Integrated Circuit
FIFO = First in, first out
LED = Light-Emitting Diode
OSAL = Operating System Abstraction Layer

1.3.2 Abbreviations

ChipLib = Chip Library for higher level communication with the sensor
Item = A property or a setting inside the ChipLib or the sensor

2 ChipLib

This page describes the work with the ChipLib in more detail.

2.1 Block diagram

This block diagram shows you all the provided functions of the library. Three groups of functions are available:

- initialize/shutdown: starts and stops the work with the ChipLib
- item/configuration: allows the user to configure the ChipLib. A single property will be called item. A collection of items will be called configuration
- measurement: this functions handle the real measurement

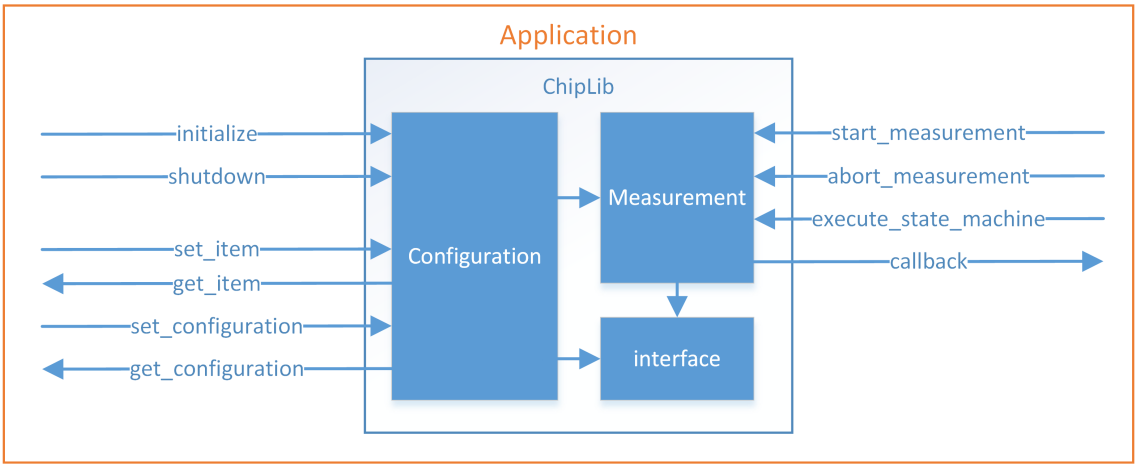

Figure 2 ChipLib API

The complete functions and their arguments will be described in section ChipLib Functions.

2.2 ChipLib states

Following image describes the internal states of the ChipLib:

- uninitialized: The library must be initialized first and than it jumps to the state configuration
- configuration: Only here, the item handling is possible.
- measurement: This state starts with start_measurement and measures data inside the function execute_↩ state_machine. After finished measurement or abort, state configuration will be active again.

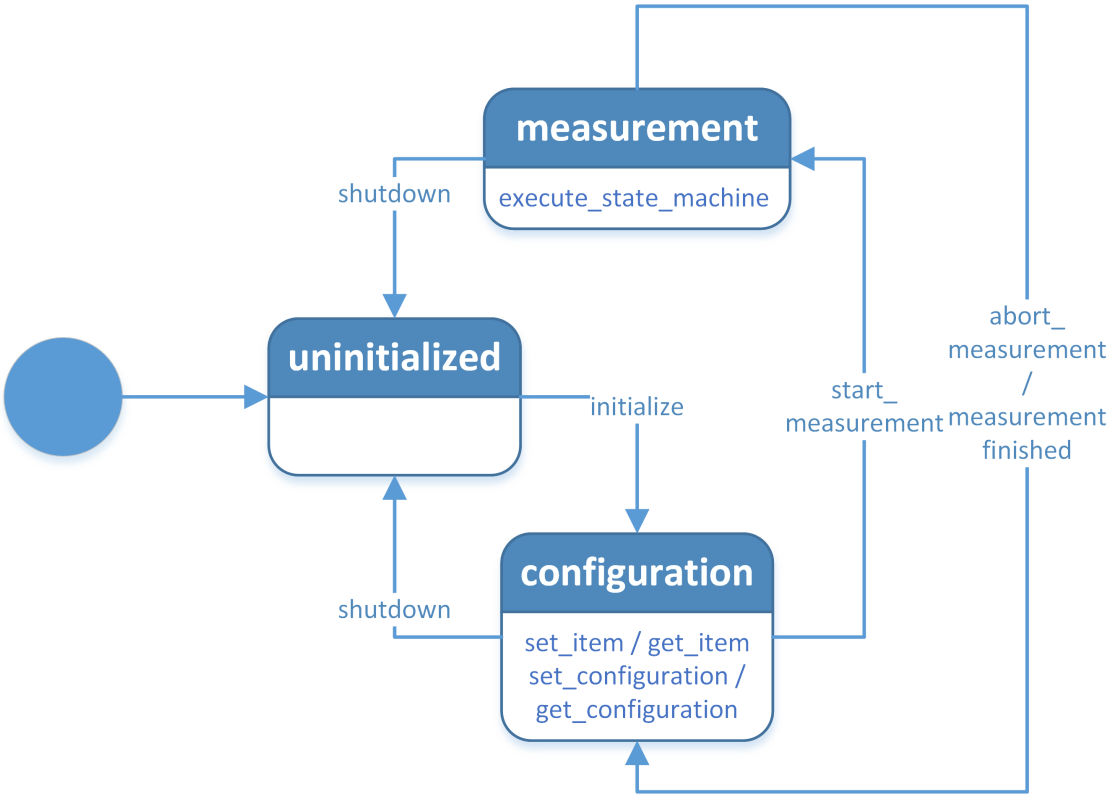

Figure 3 ChipLib States

2.3 Multi device support

The compile flag NUM_SUPPORTED_DEVICES describes how many devices from type AS7352 will be supported by this library. In most cases, one device is enough. But in some special cases, two or more are necessary.

The first parameter of all those functions is the parameter device. If NUM_SUPPORTED_DEVICES equals 1, than the parameter device must be always 0.

Note

In all examples the macro DEV_ID = 0 will be used for the parameter device.

2.4 Usage

It depends on platform and software architecture how the measurement will be controlled. Important is the implementation of the OSAL as well. Two main architectures will be discussed here:

- bare metal
- multi tasking.

To support this two different architectures, the function as7352_execute_state_machine is not a blocking function inside the ChipLib. But the function spectral_osal_wait_for_event, which is called every time by function as7352_execute_state_machine can be implemented blocking to support sleeping threads. In both architectures, the function as7352_execute_state_machine must be called cyclic!

The measurement state machine can started with as7352_start_measurement only. A measurement finished automatically in dependency of the parameter ::ITEM_ID_MEAS_COUNT or it aborts the measurement by calling the function as7352_abort_measurement.

2.4.1 Architecture bare-metal

The measurement state machine of the ChipLib will be called cyclic in a super-loop (single task). The call interval shall be as fast as possible, but depends on the integration time. It is no problem if the application needs more time between two calls. But as a result the application measures less data. The function as7352_execute_state_machine must be return immediately. Therefore, the OSAL function spectral_osal_wait_for_event can't be implemented as blocking function!

The following example shows pseudo code how the library will be used in this case:

```
// initialize the library
as7352_initialize(DEV_ID, ...);
// set properties
as7352_set_item(DEV_ID, ...);
as7352_set_item(DEV_ID, ...);
// alternative
as7352_set_configuration(DEV_ID, ...);
// starts the measurement
as7352_start_measurement(DEV_ID);
// call the statemachine cyclic
while (true) {
     // cyclic measurement
     as7352_execute_state_machine(DEV_ID, ...);
     // do other application staff
}
// shutdown the library on exit
as7352_shutdown(DEV_ID);
```

Note

Setting of items/configuration is allowed in ChipLib state "configuration" of the state machine only and will be executed directly with-out processing the state machine. The state machine must be run for measurement only!

2.4.2 Architecture Multi tasking

In this scenario, the function as7352_execute_state_machine will be called in a separate worker thread in background. So, function spectral_osal_wait_for_event can be implemented as blocking to support sleeping in the worker thread.

Pseudo code for the application task:

```
// initialize the library
as7352_initialize(DEV_ID, ...);
stop_condition = false;
// start background task
// set properties
as7352_set_item(DEV_ID, ...);
as7352_set_item(DEV_ID, ...);
```

```
// alternative
as7352_set_configuration(DEV_ID, ...);
// starts the measurement
as7352_start_measurement(DEV_ID);
// wait for data which will be received by the background task
stop_condition = true;
// close background task
// shutdown the library on exit
as7352_shutdown(DEV_ID);
```

Pseudo code for the background task:

```
while (not stop_condition) {
    // cyclic measurement
    as7352_execute_state_machine(DEV_ID, ...);
}
```

2.5 Measurement modes

The measurement data will be received via the callback function as7352_callback_t It depends on the measurement mode and the configuration which data will be received.

The kind and the format of the transmitted data is configured by item ITEM_ID_MEAS_TYPE.

2.5.1 Mode: spectral measurement

The callback function returns 16, 24, or 32 bytes of data in dependency of the configured channel (8, 12, or 16 times 16bit data). With the help of ITEM_ID_MEASURE_ITEMS it is possible to append additional item data.

2.5.2 Mode: FIFO measurement

The callback function returns in maximum 252 bytes. This is the maximum size, before an overflow will be detected. In dependency of the system performance and the configured integration time, the transferred data differs. Because of FIFO data size of 16bit, it is necessary to cast the data to 16bit.

If the programmed flicker integration time is smaller or equal 255, the resulting ADC values also cannot be greater than 255. Therefore, the 8-Bit FIFO mode is automatically used. In this mode, the FIFO only stores 8-Bit values, which can be read out in half the time. Other than speed, the 8-Bit mode has no effect on data aquisition. The data will still be saved in the 16-Bit data buffer to remain compatible with the rest of the chip library.

Like the spectral measurement, additional items can be transferred as well. This will be done after each FIFO readout (at least 16 values).

2.5.3 Mode: Parallel measurement

The callback function returns in maximum 252 bytes + 2 bytes for the measurement type (unsigned 16bit value. See as7352_measurement_types). The following bytes after the measurement type will be handled like in the other chapters described: Mode: spectral measurement & Mode: FIFO measurement

2.6 Item definition

An item is a property, which can configure or retrieve information of a part of the library or the sensor. For example an item is a number how often shall be measured with the sensor. Each item has a unique size. The size is described in bytes.

2.7 Multiple item configuration

The ChipLib provides a function named as7352_set_configuration to write more than one item in one function call. This can be used in scenarios where the configuration for a measurement is saved in a file or a persistent storage.
Example:

```
uint8_t data_buffer[10];
// Create the buffer with items ATIME, ASTEP and AGAIN
data_buffer[0] = ITEM_SIZE_ATIME;
data_buffer[1] = ITEM_ID_ATIME;
data_buffer[2] = 25;
data_buffer[3] = ITEM_SIZE_ASTEP;
data_buffer[4] = ITEM_ID_ASTEP;
data_buffer[5] = 0x1F;
data_buffer[6] = 0x1A;
data_buffer[7] = ITEM_SIZE_AGAIN;
data_buffer[8] = ITEM_ID_AGAIN;
data_buffer[9] = 4;
// write the data to the chip library
as7352_set_configuration(DEV_ID, data_buffer, 10);
```

Furthermore, you can readout the current configuration of the library with a single function: as7352_get_configuration. You have to call the function twice, if you don't have information about the necessary memory space.
Example:

```
uint32_t size = 0;
uint8_t data_buffer[1000];
// readout the buffer size
ASSERT_EQ(as7352_get_configuration(DEV_ID, NULL, &size), ERR_SUCCESS);
// readout all items
ASSERT_EQ(as7352_get_configuration(DEV_ID, data_buffer, &size), ERR_SUCCESS);
```

3 OSAL

The following pages describe the OSAL (operating system abstraction layer) of the ChipLib. This is used to encapsulate platform and operating system specific function to a separate layer, where everybody can customize this interface to their own requirements.

This interface is implemented as easy as possible. No complex data types are used and dependencies between functions are kept as small as possible. An overview of the OSAL is shown in the next section.

3.1 Block diagram

The following figure describes the dependencies to the ChipLib. Inside the OSAL, functions like I2C and Interrupt handling must be implemented. Other components like external temperature sensors and LEDs are optional.

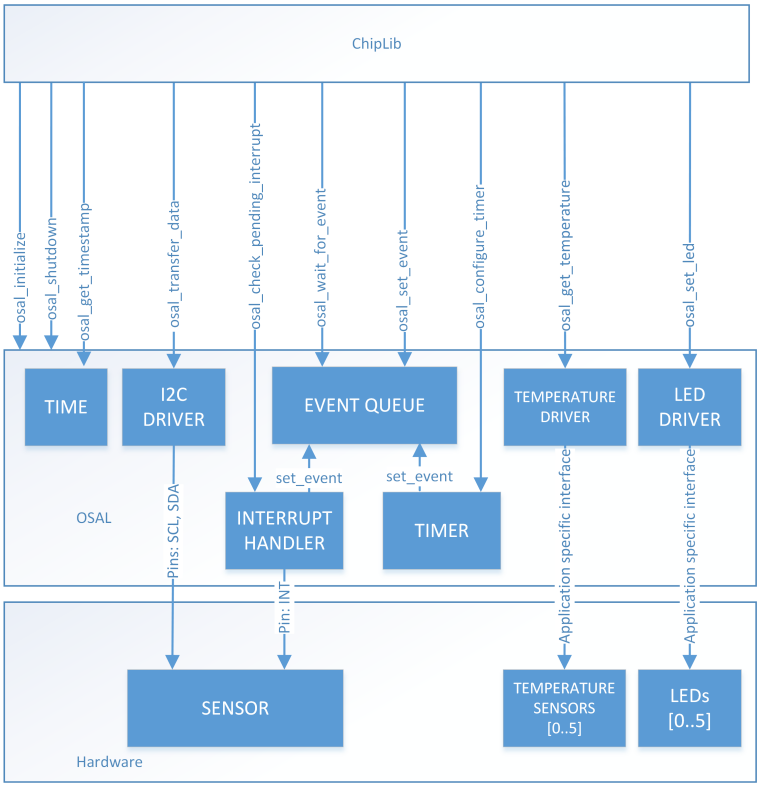

Figure 4 Overview OSAL

Function blocks:

- I2C DRIVER: Is used to communicate with the sensor. The definition of the transfer function allows other communication interfaces as well (future use only).
- INTERRUPT HANDLER: This component handles the interrupt pin and fires an event on falling edge. Usage of the interrupt pin is recommend. If not possible, the TIMER can be used instead.
- TIMER: Timer component will be used to calculate measurement times or timeouts.
- EVENT QUEUE: All event messages will be collected here. The ChipLib polls cyclic or waits for new events.
- TEMPERATURE DRIVER: This component is optional and allows synchronized readout of temperature values during spectral measurement.
- LED DRIVER: The same as TEMPERATURE SENSOR. It is optional and can synchronize external LEDs to the measurement.
- TIME: This module is used to provide an actual timestamp.

3.2 Function dependencies ChipLib - OSAL

Follwing table describes, which OSAL function is called by which ChipLib function:

Table 1 Function dependencies ChipLib - OSAL

OSAL functions	ChipLib functions
spectral_osal_initialize	as7352_initialize
spectral_osal_shutdown	as7352_shutdown
spectral_osal_transfer_data	as7352_initialize, as7352_shutdown, as7352_set_item, as7352_get_item, as7352_set_configuration, as7352_get_configuration, as7352_execute_state_machine
spectral_osal_set_event	as7352_start_measurement, as7352_execute_state_machine, as7352_abort_measurement
spectral_osal_wait_for_event	as7352_execute_state_machine
spectral_osal_check_pending_interrupt	as7352_execute_state_machine
spectral_osal_configure_timer	as7352_execute_state_machine
spectral_osal_set_led	as7352_set_item, as7352_set_configuration, as7352_execute_state_machine
spectral_osal_get_temperature	as7352_get_item, as7352_get_configuration, as7352_execute_state_machine
spectral_osal_get_timestamp	as7352_get_item, as7352_get_configuration, as7352_execute_state_machine

3.3 Interfaces

The complete functions and their arguments will be described in section OSAL Functions.

4 Module Index

4.1 Modules

Here is a list of all modules:

5 Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

6 Module Documentation

6.1 ChipLib Functions

This is the chip library for ams spectral sensor AS7352.

Functions

- err_code_t CHIPLIB_DECLDIR as7352_initialize (const uint8_t device, const as7352_callback_t p_callback, const void ∗p_cb_param, const char ∗p_interface_descr)

  Initializes the library and the device.
- err_code_t CHIPLIB_DECLDIR as7352_shutdown (const uint8_t device)

  Stops all internal actions and power down the device.
- err_code_t CHIPLIB_DECLDIR as7352_set_item (const uint8_t device, const enum as7352_item_ids id, void ∗p_data, const uint8_t size)

  Set an item.
- err_code_t CHIPLIB_DECLDIR as7352_get_item (const uint8_t device, const enum as7352_item_ids id, void ∗p_data, const uint8_t size)

  Get an item.
- err_code_t CHIPLIB_DECLDIR as7352_set_configuration (const uint8_t device, uint8_t ∗p_data, const uint32_t size)

  Configuration of multiple sensor or library items.
- err_code_t CHIPLIB_DECLDIR as7352_get_configuration (const uint8_t device, uint8_t ∗p_data, uint32_t ∗p_size)

  Request of all supported sensor and library items.
- err_code_t CHIPLIB_DECLDIR as7352_start_measurement (const uint8_t device)

  Starts a measurement.
- err_code_t CHIPLIB_DECLDIR as7352_execute_state_machine (const uint8_t device, enum as7352_states ∗p_state)

  Executes a measurement step.
- err_code_t CHIPLIB_DECLDIR as7352_abort_measurement (const uint8_t device)

  Abort a measurement.

6.1.1 Detailed Description

This is the chip library for ams spectral sensor AS7352.

6.1.2 Function Documentation

6.1.2.1 as7352_initialize()

```
err_code_t CHIPLIB_DECLDIR as7352_initialize (
        const uint8_t device,
        const as7352_callback_t p_callback,
        const void * p_cb_param,
        const char * p_interface_descr )
```

Initializes the library and the device.

Following tasks will be done here:

- Initialize hardware abstraction layer.
- Set default configuration values.

Note

This function must be called at first, otherwise all other functions return with error code.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes)
in	*p_callback*	Pointer to the callback function, see as7352_callback_t
in	*p_cb_param*	Optional pointer to an application parameter, which will be transmitted with every callback.
in	*p_interface_descr*	Chiplib forwards this interface description to spectral_osal_initialize.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_IDENTIFICATION	The specified sensor was not found.
ERR_DATA_TRANSFER	Communication error to sensor.

6.1.2.2 as7352_shutdown()

```
err_code_t CHIPLIB_DECLDIR as7352_shutdown (
        const uint8_t device )
```

Stops all internal actions and power down the device.

Following tasks will be done here:

- Stops measurement, if running.
- Power down the sensor device.
- Shutdown the hardware abstraction layer.
- Block calling of all other functions, but initialize.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).

Return values

ERR_SUCCESS	Function returns without error.
ERR_DATA_TRANSFER	Communication error to sensor.

6.1.2.3 as7352_set_item()

```
err_code_t CHIPLIB_DECLDIR as7352_set_item (
        const uint8_t device,
        const enum as7352_item_ids id,
        void * p_data,
        const uint8_t size )
```

Set an item.

This function configures the chip library or the sensor directly.

Following tasks will be done here:

- Searches for the corresponding internal configuration function of this item-ID.
- Calls the internal set-function if available.

Note

This function can only called if ChipLib is in state configuration.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).
in	*id*	Identification number of an item, see as7352_item_ids.
in	*p_data*	Pointer to the data of the item.
in	*size*	Sets the size in byte of data, see as7352_item_sizes.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor.
ERR_NOT_SUPPORTED	e.g. Configuration of item ITEM_ID_LED_EXT_0, but LED is not implemented. Used item-ID is not writable.

6.1.2.4 as7352_get_item()

```
err_code_t CHIPLIB_DECLDIR as7352_get_item (
    const uint8_t device,
    const enum as7352_item_ids id,
    void * p_data,
    const uint8_t size )
```

Get an item.

Reads the actual settings of sensor or library items.

Following tasks will be done here:

- Searches for the corresponding internal request function of this item-ID.
- Calls the internal request-function if available.

Note

This function can be called in state configuration or measurement.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).
in	*id*	Identification number of an item, see as7352_item_ids.
out	*p_data*	Pointer, where the data of the item can be saved.
in	*size*	Size in byte of data, see as7352_item_sizes.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor
ERR_NOT_SUPPORTED	e.g. Readout of item ITEM_ID_TEMP_EXT_2, but function is not implemented.

6.1.2.5 as7352_set_configuration()

```
err_code_t CHIPLIB_DECLDIR as7352_set_configuration (
        const uint8_t device,
        uint8_t * p_data,
        const uint32_t size )
```

Configuration of multiple sensor or library items.

Following tasks will be done here:

- Searches for the corresponding internal configuration function of the given item-IDs.
- Calls the internal set-function if available.

Byte sequence of data [S0][I0][D0_0 - D0_N][S1][I1][D1_0 - D1_N] ... [SM][IM][DM_0 - DM_N] S = size of D in bytes, see enumeration as7352_item_sizes I = item id, see enumeration as7352_item_ids D = payload of the corresponding item

Note

More than one item can be string together

This function can only called if ChipLib is in state configuration.

IDs, which are readable only or not supported, will be ignored.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).
in	*p_data*	Pointer to the data of the item.
in	*size*	Size in byte of data.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor.
ERR_NOT_SUPPORTED	e.g. Configuration of item ITEM_ID_LED_EXT_0, but LED is not implemented.

See also

Multiple item configuration

6.1.2.6 as7352_get_configuration()

```
err_code_t CHIPLIB_DECLDIR as7352_get_configuration (
        const uint8_t device,
        uint8_t * p_data,
        uint32_t * p_size )
```

Request of all supported sensor and library items.

Following tasks will be done here:

- Iterate over all internal item-IDs
- Calls the internal get-function of each item-ID, if available
- Saves all data in the given data buffer

Byte sequence of data [S0][I0][D0_0 - D0_N][S1][I1][D1_0 - D1_N] ... [SM][IM][DM_0 - DM_N] S = size of D in bytes, see enumeration as7352_item_sizes I = item id, see enumeration as7352_item_ids D = payload of the corresponding item

Note

The size of the needed data buffer can be requested by calling this function with p_data = NULL and the value of p_size must be set to zero. The size will be copied to p_size.

This function can only called if ChipLib is in state configuration.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).
out	*p_data*	Pointer, where the item data can be saved.
in,out	*p_size*	Size in byte of p_data. After return of this function the used size will be saved here.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor.
ERR_NOT_SUPPORTED	e.g. Readout of item ITEM_ID_TEMP_EXT_2, but function is not implemented.

See also

Multiple item configuration

6.1.2.7 as7352_start_measurement()

```
err_code_t CHIPLIB_DECLDIR as7352_start_measurement (
        const uint8_t device )
```

Starts a measurement.

This function sets only an internal event, that the measurement starts on next state machine step.

Note

This function can only called if ChipLib is in state configuration.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor.

6.1.2.8 as7352_execute_state_machine()

```
err_code_t CHIPLIB_DECLDIR as7352_execute_state_machine
(
        const uint8_t device,
        enum as7352_states * p_state )
```

Executes a measurement step.

Following tasks will be done here:

- Checks if an internal event was set.
- Checks if the internal event can be handled on actual state machine state.
- Calls the corresponding transition-function.

Note

This function can be called after finished initialization, but does only anything if the measurement was started.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).
out	*p_state*	Pointer, where the current state can be saved, see enumeration as7352_states.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid.
ERR_PERMISSION	The access to the library is blocked, call as7352_initialize at first.
ERR_DATA_TRANSFER	Communication error to sensor.
ERR_NOT_SUPPORTED	e.g. readout of item ITEM_ID_TEMP_EXT_2, but function is not implemented.
ERR_OVERFLOW	Get overflow in FIFO mode.
ERR_SENSOR_CONFIG	Break time is too high.

6.1.2.9 as7352_abort_measurement()

```
err_code_t CHIPLIB_DECLDIR as7352_abort_measurement (
        const uint8_t device )
```

Abort a measurement.

This function sets only an internal event, that the measurement stops as soon as possible.

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes).

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	An argument is invalid
ERR_PERMISSION	Access to the library is blocked, call as7352_initialize at first.

6.2 OSAL Functions

This is the abstraction layer for the chip library.

Data Structures

- struct spectral_osal_id

Macros

- #define **EVENT_TIMER_MEASUREMENT** EVENT_TIMER_SPECTRAL

Typedefs

- typedef struct spectral_osal_id osal_id_t

Enumerations

- enum EVENT_TYPES {
  EVENT_NONE = 0,
  EVENT_NEW_STATE = 1,
  EVENT_ERROR = 2,
  EVENT_START = 3,
  EVENT_ABORT = 4,
  EVENT_INTERRUPT = 5,
  EVENT_TIMER_SPECTRAL = 6,
  EVENT_TIMER_TIMEOUT = 7,
  EVENT_TIMER_LED = 8,
  EVENT_TIMER_FIFO = 9,
  EVENT_TIMER_4 = 10,
  EVENT_TIMER_5 = 11,
  EVENT_TIMER_6 = 12,
  EVENT_TIMER_7 = 13 }

Functions

- err_code_t spectral_osal_initialize (const osal_id_t osal_id, const char *p_interface_desc)

  Initialization of the hardware abstraction layer.
- err_code_t spectral_osal_shutdown (const osal_id_t osal_id)

  Shutdown of the hardware abstraction layer.
- err_code_t spectral_osal_transfer_data (const osal_id_t osal_id, uint8_t *p_send_data, const uint8_t send↩ _data_size, uint8_t *p_receive_data, const uint8_t receive_data_size)

  Write and read data in one transfer.
- err_code_t spectral_osal_set_event (const osal_id_t osal_id, const uint16_t event, const uint16_t payload)

  Pushes an event into the event queue.
- err_code_t spectral_osal_wait_for_event (const osal_id_t osal_id, uint16_t *p_event, uint16_t *p_payload)

Waits for an event / Checks if an event is active.

- err_code_t spectral_osal_check_pending_interrupt (const osal_id_t osal_id)

  Retriggers the EVENT_INTERRUPT-event, if the interrupt pin is still low.
- err_code_t spectral_osal_configure_timer (const osal_id_t osal_id, const uint8_t timer_id, const uint32_t timer_us)

  Triggers the start of a timer.
- err_code_t spectral_osal_set_led (const osal_id_t osal_id, const uint8_t led_id, const uint16_t brightness)

  Configures an external LED.
- err_code_t spectral_osal_get_temperature (const osal_id_t osal_id, const uint8_t temp_id, int32_t *p_temperature)

  Reads temperature of an external temperature sensor.
- err_code_t spectral_osal_get_timestamp (const osal_id_t osal_id, uint32_t *p_timestamp_us_l, uint32_t *p_timestamp_us_h)

  Reads timestamp of the system in microseconds.

6.2.1 Detailed Description

This is the abstraction layer for the chip library.

The functions has dependencies to the operatation system. So the function must be implemented application specific. Following function groups must be implemented:

- initialize/shutdown
- I2C-transfers
- message event handler
- timer control

Following functions are application specific and optional:

- temperature readout
- external LED control
- interrupt handler
- timestamp reading

6.2.2 Typedef Documentation

6.2.2.1 osal_id_t `typedef struct spectral_osal_id osal_id_t`

Specifies the device type and id to link to the right driver.

6.2.3 Enumeration Type Documentation

6.2.3.1 EVENT_TYPES `enum EVENT_TYPES`

Event types for measurement state machine

Enumerator

EVENT_NONE	No event, will be used if no event is active.
EVENT_NEW_STATE	Will be used internally to jump between different states. A second parameter keeps the new state.
EVENT_ERROR	Signals an error state. A second parameter keeps the error code.
EVENT_START	A measurement shall be started.
EVENT_ABORT	A running measurement shall be aborted.
EVENT_INTERRUPT	Interrupt pin is low.
EVENT_TIMER_SPECTRAL	The spectral measurement timer is expired.
EVENT_TIMER_TIMEOUT	The timeout timer is expired.
EVENT_TIMER_LED	The LED switch timer is expired.
EVENT_TIMER_FIFO	The FIFO measurement timer is expired.
EVENT_TIMER_4	Not used. Reserved for future applications.
EVENT_TIMER_5	Not used. Reserved for future applications.
EVENT_TIMER_6	Not used. Reserved for future applications.
EVENT_TIMER_7	Not used. Reserved for future applications.

6.2.4 Function Documentation

6.2.4.1 spectral_osal_initialize()

```
err_code_t spectral_osal_initialize (
        const osal_id_t osal_id,
        const char * p_interface_desc )
```

Initialization of the hardware abstraction layer.

- Initialization of global parameters.
- Open the interface to the sensor.
- Initialization of all external peripheries.

Note

This function must be called at first!

Pseudo code for implementation example:

```
check_osal_chip_id();
check_device_osal_device_id();
// initialize mandatory components
initialize_i2c_interface();
initialize_event_handler();
initialize_timer_control();
// initialize optional components
initialize_interrupt_handler();
initialize_temperature_sensors();
initialize_led_drivers();
```

Parameters

in	*osal_id*	Handle to the device.
in	*p_interface_desc*	Can be used to transfer special initialization data like interface description.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	If the osal_id was not fitted.
ERR_COM_INTERFACE	The interface to the sensor is faulty.

6.2.4.2 spectral_osal_shutdown() `err_code_t spectral_osal_shutdown (`

```
const osal_id_t osal_id )
```

Shutdown of the hardware abstraction layer.

- Closes the interface to the sensor
- Power down of all external peripheries

Note

This function must be called for cleanup

Parameters

in	*osal_id*	handle to the device

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	If the osal_id was not fitted.
ERR_COM_INTERFACE	The interface to the sensor is faulty

6.2.4.3 spectral_osal_transfer_data() `err_code_t spectral_osal_transfer_data (`

```
const osal_id_t osal_id,
uint8_t * p_send_data,
const uint8_t send_data_size,
```

```
            uint8_t * p_receive_data,
            const uint8_t receive_data_size )
```

Write and read data in one transfer.

This function is a abstraction layer for normal I2C-transfers. All kinds of different modes are possible

- register access
- single register write
- single register read
- burst write
- burst read

Pseudo code for implementation example:

```
check_osal_chip_id();
check_device_osal_device_id();
if (0 < send_data_size) {
    write_i2c_data(i2c_address, p_send_data, send_data_size)
};
if (0 < receive_data_size) {
    read_i2c_data(i2c_address, p_receive_data, receive_data_size)
};
```

Parameters

in	*osal_id*	Handle to the device.
in	*p_send_data*	Pointer to data which shall be sent.
in	*send_data_size*	Size of send data in bytes.
out	*p_receive_data*	Pointer to memory, where received data can be saved.
in	*receive_data_size*	Number of data, which shall be read.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	If the osal_id was not fitted.
ERR_POINTER	If buffer size is unequal 0 and buffer address is zero.
ERR_COM_INTERFACE	The interface to the sensor is faulty.
ERR_DATA_TRANSFER	Data transfer error, like bus error or timeout.

6.2.4.4 spectral_osal_set_event()

```
err_code_t spectral_osal_set_event (
            const osal_id_t osal_id,
            const uint16_t event,
            const uint16_t payload )
```

Pushes an event into the event queue.

Note

Must be return immediately.

Parameters

in	*osal_id*	Handle to the device.
in	*event*	Event type, see enum EVENT_TYPES.
in	*payload*	Optional parameter of event, type specific.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	If the osal_id was not fitted or the event is not supported.
ERR_EVENT	Error during event registration (e.g. overflow)

6.2.4.5 spectral_osal_wait_for_event()

```
err_code_t spectral_osal_wait_for_event (
        const osal_id_t osal_id,
        uint16_t * p_event,
        uint16_t * p_payload )
```

Waits for an event / Checks if an event is active.

Note

In non-blocking mode, the event parameter returns EVENT_NONE on empty queue.

Pseudo code for implementation example:

```
check_osal_chip_id();
check_device_osal_device_id();
// optional on non-blocking interface,
// if interrupt event will not be handled in an interrupt service routine.
if (interrupt_pin_low()) {
    spectral_osal_set_event(osal_id, EVENT_INTERRUPT, 0);
}
// optional on non-blocking interface,
// if a configured timer is expired
if (timer_is_expired(&timer_id)) {
    spectral_osal_set_event(osal_id, EVENT_TIMER_MEASUREMENT + timer_id, 0);
}
if (blocking_mode) {
    wait_for_new_event(p_event, p_payload);
} else {
    // return immediately
    check_for_new_event(p_event, p_payload);
}
```

Parameters

in	*osal_id*	Handle to the device.
out	*p_event*	Event type, see enum EVENT_TYPES.
out	*p_payload*	Optional parameter of event, type specific, otherwise zero.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	If the osal_id was not fitted.
ERR_EVENT	Error during event handling (e.g. overflow).
ERR_POINTER	If one of the pointers are zero.
ERR_INTERRUPT	If checking interrupt failed (optional, if not handled in separate thread)
ERR_TIMER_ACCESS	If checking timer expiration failed (optional, if not handled in separate thread)

6.2.4.6 spectral_osal_check_pending_interrupt() err_code_t spectral_osal_check_pending_interrupt (

```
            const osal_id_t osal_id )
```

Retriggers the EVENT_INTERRUPT-event, if the interrupt pin is still low.

It is not mandatory to implement this function, because ChipLib uses timer interface at default. If this function will not be used, return error code ERR_NOT_SUPPORTED.

Note

This is necessary because sometimes the measurement state machine is to slow to get the event on right time.

Pseudo code for implementation example:

```
check_osal_chip_id();
check_device_osal_device_id();
if (interrupt_pin_low()) {
    spectral_osal_set_event(osal_id, EVENT_INTERRUPT, 0);
}
```

Parameters

in	*osal_id*	Handle to the device.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	If the osal_id was not fitted.
ERR_INTERRUPT	If checking interrupt failed.
ERR_NOT_SUPPORTED	If function is not supported.

6.2.4.7 spectral_osal_configure_timer() err_code_t spectral_osal_configure_timer (

```
const osal_id_t osal_id,
const uint8_t timer_id,
const uint32_t timer_us )
```

Triggers the start of a timer.

- Up to 8 timers will be supported.
- The corresponding events are ::EVENT_TIMER_MEASUREMENT until EVENT_TIMER_7.

Parameters

in	*osal_id*	Handle to the device.
in	*timer_id*	Supports up to 8 timer. [0 - 7]
in	*timer_us*	Set the timeoffset in microseconds. After this time the corresponding event will be fired

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	If the osal_id was not fitted or the timer id is out of range.
ERR_TIMER_ACCESS	If timer configuration failed.

6.2.4.8 spectral_osal_set_led()

```
err_code_t spectral_osal_set_led (
const osal_id_t osal_id,
const uint8_t led_id,
const uint16_t brightness )
```

Configures an external LED.

Note

Optional function: If not used, return ERR_NOT_SUPPORTED.

Parameters

in	*osal_id*	Handle to the device.
in	*led_id*	Supports up to 6 LEDs. [0 - 5].
in	*brightness*	Brightness of the LED in per mile [0 - 1000].

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	If the osal_id was not fitted or the led id/brightness is out of range.
ERR_LED_ACCESS	If led configuration failed.

Return values

ERR_NOT_SUPPORTED	If function is not supported.

6.2.4.9 spectral_osal_get_temperature()

```
err_code_t spectral_osal_get_temperature (
    const osal_id_t osal_id,
    const uint8_t temp_id,
    int32_t * p_temperature )
```

Reads temperature of an external temperature sensor.

Note

Optional function: if not used, return ERR_NOT_SUPPORTED.

Parameters

in	*osal_id*	Handle to the device.
in	*temp_id*	Supports up to 6 external temperature sensors. [0 - 5].
out	*p_temperature*	Temperature in millidegree Celsius.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	If the osal_id was not fitted or the temp id is out of range.
ERR_POINTER	If the pointer is zero.
ERR_TEMP_SENSOR_ACCESS	If temperature readout failed.
ERR_NOT_SUPPORTED	If function is not supported.

6.2.4.10 spectral_osal_get_timestamp()

```
err_code_t spectral_osal_get_timestamp (
    const osal_id_t osal_id,
    uint32_t * p_timestamp_us_l,
    uint32_t * p_timestamp_us_h )
```

Reads timestamp of the system in microseconds.

Note

Optional function: if not used, return ERR_NOT_SUPPORTED.

Attention

The 32bit timestamp can only handle a time range of round about 1hour. Afterwards, is jumps back to zero.

Parameters

in	*osal_id*	Handle to the device.
out	*p_timestamp_us↩ _l*	Lower 32bit of timestamp in microseconds.
out	*p_timestamp_us↩ _h*	Higher 32bit of timestamp in microseconds.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	If the osal_id was not fitted.
ERR_POINTER	If the pointer is zero.
ERR_NOT_SUPPORTED	If function is not supported.

6.3 Error Codes

Description of given error codes.

Macros

- #define M_CHECK_ARGUMENT_LOWER_EQUAL(arg_value, max_value)
- #define M_CHECK_ARGUMENT_LOWER(arg_value, max_value)
- #define M_CHECK_ARGUMENT_MULTIPLE_OF(multiplier, value)
- #define M_CHECK_NULL_POINTER(pointer)
- #define M_CHECK_SIZE(expected, given)
- #define M_UNUSED_PARAM(x) (void)(x)

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 }

6.3.1 Detailed Description

Description of given error codes.

Here you can find a generic error code table, which is used by some ams libraries.

6.3.2 Macro Definition Documentation

6.3.2.1 M_CHECK_ARGUMENT_LOWER_EQUAL #define M_CHECK_ARGUMENT_LOWER_EQUAL(

arg_value,
max_value)

Value:

```
do {
                \
    if (arg_value > max_value) {
                \
        return ERR_ARGUMENT;
                \
    }
                \
} while (0)
```

Returns an error code if the arg_value is greater than the max_value.

6.3.2.2 M_CHECK_ARGUMENT_LOWER #define M_CHECK_ARGUMENT_LOWER(

arg_value,
max_value)

Value:

```
do {
                \
    if (arg_value >= max_value) {
                \
        return ERR_ARGUMENT;
                \
    }
                \
} while (0)
```

Returns an error code if the first value is greater or equal than the second one.

6.3.2.3 M_CHECK_ARGUMENT_MULTIPLE_OF #define M_CHECK_ARGUMENT_MULTIPLE_OF(*multiplier,* *value*)

Value:

```
do {                                                    \
    if  ((multiplier % value) != 0) {                   \
        return ERR_ARGUMENT;                            \
    }                                                   \
} while (0)
```

Returns an error code if the value is not a multiple of multiplier.

6.3.2.4 M_CHECK_NULL_POINTER #define M_CHECK_NULL_POINTER(*pointer*)

Value:

```
do {                                                    \
    if (NULL == pointer) {                              \
        return ERR_POINTER;                             \
    }                                                   \
} while (0)
```

Returns an error code if the given address is zero.

6.3.2.5 M_CHECK_SIZE #define M_CHECK_SIZE(*expected,* *given*)

Value:

```
do {                                                    \
    if (expected != given) {                            \
        return ERR_SIZE;                                \
    }                                                   \
} while (0)
```

Returns an error code if the values are not equal.

6.3.2.6 M_UNUSED_PARAM #define M_UNUSED_PARAM(*x*) (void)(x)

Mark unused arguments to get no fails on static code analysis.

6.3.3 Typedef Documentation

6.3.3.1 err_code_t `typedef enum error_codes err_code_t`

This definition will be used for function return values.

6.3.4 Enumeration Type Documentation

6.3.4.1 error_codes `enum error_codes`

Values represent the error codes.

Enumerator

ERR_SUCCESS	Normal return code if everything was successful executed.
ERR_PERMISSION	Operation not permitted
ERR_MESSAGE	Message is invalid. For example: • Message type is not supported • incorrect crc ...
ERR_MESSAGE_SIZE	Message has the wrong size.
ERR_POINTER	Pointer is invalid. Can be a NULL Pointer or point to a wrong memory area.
ERR_ACCESS	Access denied
ERR_ARGUMENT	Invalid argument
ERR_SIZE	Argument size is too long or too short.
ERR_NOT_SUPPORTED	Function is not supported/implemented.
ERR_TIMEOUT	Got timeout while waiting for answer.
ERR_CHECKSUM	Checksum comparision failed.
ERR_OVERFLOW	Data overflow detected.
ERR_EVENT	Error to get or set an event. For example: • event queue is full or empty • receive an unexpected event ...
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.

Enumerator

ERR_COM_INTERFACE	Generic communication interface error. For example: • communication interface is not available • error during open or close an communication interface ...
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

6.4 Definitions

Description of the used data types.

Data Structures

- struct as7352_serial
- struct as7352_version
- struct as7352_led_pattern
- struct as7352_led_config
- struct as7352_auto_gain

Macros

- #define CHIP_LIB_IDENT 7352
- #define AS7352_LED_PATTERN_NUM 10
- #define AS7352_MAX_ITEM_BUFFER_SIZE 80
- #define AS7352_MAX_DATA_BUFFER_SIZE 256
- #define AS7352_GAIN_FACTOR_NUM 15
- #define AS7352_SATURATED 65535
- #define **CHIPLIB_DECLDIR**
- #define TRUE 1
- #define FALSE 0
- #define DIV64_S64(s64dividend, s64divisor) (s64dividend / s64divisor)
- #define DIV64_U64(u64dividend, u64divisor) (u64dividend / u64divisor)

Typedefs

- typedef void(∗ as7352_callback_t) (uint8_t device, uint8_t error, void ∗p_data, uint32_t data_size, void ∗p_↩ items, uint32_t items_size, void ∗p_cb_param)

  Callback function, which transfers the measurement results to the application.

Enumerations

- enum as7352_saturation_flags {
  SATURATION_FLAG_FD_DIGITAL = 0x01,
  SATURATION_FLAG_FD_ANALOG = 0x02,
  SATURATION_FLAG_SPECTRAL_ANALOG = 0x08,
  SATURATION_FLAG_SPECTRAL_DIGITAL = 0x10 }
- enum as7352_measurement_types {
  MEASUREMENT_TYPE_SPECTRAL = 0,
  MEASUREMENT_TYPE_FIFO = 1,
  MEASUREMENT_TYPE_SPECTRAL_FIFO = 2,
  MEASUREMENT_TYPE_GOERTZEL = 3,
  MEASUREMENT_TYPE_NUM = 4 }

- enum as7352_channels {
  CHANNEL_DISABLED = 0,
  CHANNEL_F1 = 1,
  CHANNEL_F2 = 2,
  CHANNEL_FZ = 3,
  CHANNEL_F3 = 4,
  CHANNEL_F4 = 5,
  CHANNEL_FY = 6,
  CHANNEL_F5 = 7,
  CHANNEL_FXL = 8,
  CHANNEL_F6 = 9,
  CHANNEL_F7 = 10,
  CHANNEL_F8 = 11,
  CHANNEL_NIR = 12,
  CHANNEL_FD = 13,
  CHANNEL_DARK = 14,
  CHANNEL_VISLT = 16,
  CHANNEL_VISRB = 32,
  CHANNEL_VISRT = 64,
  CHANNEL_VISLB = 128 }
- enum as7352_gain {
  GAIN_0_5X = 0,
  GAIN_1X = 1,
  GAIN_2X = 2,
  GAIN_4X = 3,
  GAIN_8X = 4,
  GAIN_16X = 5,
  GAIN_32X = 6,
  GAIN_64X = 7,
  GAIN_128X = 8,
  GAIN_256X = 9,
  GAIN_512X = 10,
  GAIN_1024X = 11,
  GAIN_2048X = 12,
  GAIN_4096X = 13,
  GAIN_5120X = 14 }
- enum as7352_led_masks {
  LED_MASK_OFF = 0x00,
  LED_MASK_INTERN = 0x01,
  LED_MASK_EXT_0 = 0x02,
  LED_MASK_EXT_1 = 0x04,
  LED_MASK_EXT_2 = 0x08,
  LED_MASK_EXT_3 = 0x10,
  LED_MASK_EXT_4 = 0x20,
  LED_MASK_EXT_5 = 0x40,
  LED_MASK_OUTPUT = 0x80 }
- enum as7352_sync_mode {
  SYNC_MODE_DISABLED = 0,
  SYNC_MODE_RISING_EDGE = 1,
  SYNC_MODE_FALLING_EDGE = 2,
  SYNC_MODE_NUMBER = 3 }
- enum as7352_item_ids {
  ITEM_ID_RESERVED = 0,
  ITEM_ID_ASTEP = 1,

ITEM_ID_ATIME = 2,
ITEM_ID_ITIME = 3,
ITEM_ID_AGAIN = 4,
ITEM_ID_MEAS_TYPE = 5,
ITEM_ID_BREAK = 6,
ITEM_ID_CHANNELS = 7,
ITEM_ID_VERSION = 8,
ITEM_ID_SERIAL = 9,
ITEM_ID_AUTOZERO = 10,
ITEM_ID_MEAS_COUNT_ALS = 11,
ITEM_ID_LED_PATTERN = 12,
ITEM_ID_LED_WAIT_TIME = 13,
ITEM_ID_INTERRUPT_PIN = 14,
ITEM_ID_LED_INTERN = 15,
ITEM_ID_LED_EXT_0 = 16,
ITEM_ID_LED_EXT_1 = 17,
ITEM_ID_LED_EXT_2 = 18,
ITEM_ID_LED_EXT_3 = 19,
ITEM_ID_LED_EXT_4 = 20,
ITEM_ID_LED_EXT_5 = 21,
ITEM_ID_OUTPUT = 22,
ITEM_ID_TEMP_EXT_0 = 23,
ITEM_ID_TEMP_EXT_1 = 24,
ITEM_ID_TEMP_EXT_2 = 25,
ITEM_ID_TEMP_EXT_3 = 26,
ITEM_ID_TEMP_EXT_4 = 27,
ITEM_ID_TEMP_EXT_5 = 28,
ITEM_ID_MEASURE_ITEMS = 29,
ITEM_ID_FGAIN = 30,
ITEM_ID_FTIME = 31,
ITEM_ID_FTIME_US = 32,
ITEM_ID_TIMESTAMP = 33,
ITEM_ID_AUTO_GAIN_RANGE_ALS = 34,
ITEM_ID_AUTO_GAIN_RANGE_FIFO = 35,
ITEM_ID_GAIN_FACTORS = 36,
ITEM_ID_REG_READ = 37,
ITEM_ID_SATURATION = 38,
ITEM_ID_SYNC_MODE = 39,
ITEM_ID_FIFO_AGC_BLOCKSIZE = 40,
ITEM_ID_FD_COEFF = 41,
ITEM_ID_FD_DCR = 42,
ITEM_ID_FD_PERS = 43,
ITEM_ID_FD_AGC_DISABLE = 44,
ITEM_ID_FD_MODCLK = 45,
ITEM_ID_FD_SAMPLES = 46,
ITEM_ID_FD_COMPARE_VALUE = 47,
ITEM_ID_MEAS_COUNT_FIFO = 48,
ITEM_ID_MAX = 49 }

- enum as7352_item_sizes {
ITEM_SIZE_RESERVED = 0,
ITEM_SIZE_ASTEP = 2,
ITEM_SIZE_ATIME = 1,
ITEM_SIZE_ITIME = 4,
ITEM_SIZE_AGAIN = 1,

ITEM_SIZE_MEAS_TYPE = 1,
ITEM_SIZE_BREAK = 4,
ITEM_SIZE_CHANNELS_MAX = 18,
ITEM_SIZE_VERSION = 4,
ITEM_SIZE_SERIAL = 8,
ITEM_SIZE_SATURATION = 1,
ITEM_SIZE_AUTOZERO = 1,
ITEM_SIZE_MEAS_COUNT_ALS = 2,
ITEM_SIZE_MEAS_COUNT_FIFO = 4,
ITEM_SIZE_LED_PATTERN = 20,
ITEM_SIZE_LED_WAIT_TIME = 4,
ITEM_SIZE_INTERRUPT_PIN = 1,
ITEM_SIZE_LED_INTERN = 4,
ITEM_SIZE_LED_EXT_0 = 4,
ITEM_SIZE_LED_EXT_1 = 4,
ITEM_SIZE_LED_EXT_2 = 4,
ITEM_SIZE_LED_EXT_3 = 4,
ITEM_SIZE_LED_EXT_4 = 4,
ITEM_SIZE_LED_EXT_5 = 4,
ITEM_SIZE_OUTPUT = 1,
ITEM_SIZE_TEMP_EXT_0 = 4,
ITEM_SIZE_TEMP_EXT_1 = 4,
ITEM_SIZE_TEMP_EXT_2 = 4,
ITEM_SIZE_TEMP_EXT_3 = 4,
ITEM_SIZE_TEMP_EXT_4 = 4,
ITEM_SIZE_TEMP_EXT_5 = 4,
ITEM_SIZE_MEASURE_ITEMS = 10,
ITEM_SIZE_FGAIN = 1,
ITEM_SIZE_FTIME = 2,
ITEM_SIZE_FTIME_US = 4,
ITEM_SIZE_TIMESTAMP = 8,
ITEM_SIZE_AUTO_GAIN_RANGE_ALS = 2,
ITEM_SIZE_AUTO_GAIN_RANGE_FIFO = 2,
ITEM_SIZE_GAIN_FACTORS = 30,
ITEM_SIZE_REG_READ = 1,
ITEM_SIZE_SYNC_MODE = 1,
ITEM_SIZE_FIFO_AGC_BLOCKSIZE = 2,
ITEM_SIZE_FD_COEFF = 1,
ITEM_SIZE_FD_DCR = 1,
ITEM_SIZE_FD_PERS = 1,
ITEM_SIZE_FD_AGC_DISABLE = 1,
ITEM_SIZE_FD_MODCLK = 1,
ITEM_SIZE_FD_SAMPLES = 1,
ITEM_SIZE_FD_COMPARE_VALUE = 1 }

- enum as7352_states {
  STATE_CONFIG = 0,
  STATE_MEASURE = 1 }

6.4.1 Detailed Description

Description of the used data types.

These are the type definitions used by AS7352 chip library.

6.4.2 Macro Definition Documentation

6.4.2.1 CHIP_LIB_IDENT `#define CHIP_LIB_IDENT 7352`

ChipLib identification

6.4.2.2 AS7352_LED_PATTERN_NUM `#define AS7352_LED_PATTERN_NUM 10`

Number of supported LED pattern configuration

6.4.2.3 AS7352_MAX_ITEM_BUFFER_SIZE `#define AS7352_MAX_ITEM_BUFFER_SIZE 80`

Maximum size in bytes of additional item buffer in callback function

6.4.2.4 AS7352_MAX_DATA_BUFFER_SIZE `#define AS7352_MAX_DATA_BUFFER_SIZE 256`

Maximum size in bytes of data buffer in callback function

6.4.2.5 AS7352_GAIN_FACTOR_NUM `#define AS7352_GAIN_FACTOR_NUM 15`

Number of supported gains

6.4.2.6 AS7352_SATURATED `#define AS7352_SATURATED 65535`

Measured ADC value is saturated

6.4.2.7 TRUE `#define TRUE 1`

definition for bool value true

6.4.2.8 FALSE `#define FALSE 0`

definition for bool value false

6.4.2.9 DIV64_S64

```
#define DIV64_S64(
        s64dividend,
        s64divisor ) (s64dividend / s64divisor)
```

Use this macro for signed 64 Bit divisions

6.4.2.10 DIV64_U64

```
#define DIV64_U64(
        u64dividend,
        u64divisor ) (u64dividend / u64divisor)
```

Use this macro for unsigned 64 Bit divisions

6.4.3 Typedef Documentation

6.4.3.1 as7352_callback_t

```
typedef void(* as7352_callback_t) (uint8_t device, uint8_t error,
void *p_data, uint32_t data_size, void *p_items, uint32_t items_size, void *p_cb_param)
```

Callback function, which transfers the measurement results to the application.

This callback type will be registered via the function as7352_initialize. During the measurement, this function transfers the cyclic results.

p_data transfers the measured sensor data. p_items transfers additional item content, if configured. The content of the items is without item size and without item id. The order of the item can be readout by ITEM_ID_MEASURE_ITEMS.

See also

Measurement modes

Parameters

in	*device*	Handle to the device (default 0, only for multi device purposes)
in	*error*	Default ERR_SUCCESS, otherwise an error is occurred during measurement and the measurement. stops. See error_codes
in	*p_data*	Pointer to the measurement data, the content depends on configuration. See ITEM_ID_MEAS_TYPE.
in	*data_size*	Size of measurement data in bytes.
in	*p_items*	Returns an optional array with data, which can be configured with item ITEM_ID_MEASURE_ITEMS.
in	*items_size*	Size of item data in bytes. The maximum supported size is defined in definition AS7352_MAX_ITEM_BUFFER_SIZE
in	*p_cb_param*	Application parameter which was defined during call of as7352_initialize.

6.4.4 Enumeration Type Documentation

6.4.4.1 as7352_saturation_flags

```
enum as7352_saturation_flags
```

List of saturation flags ::ITEM_ID_SATURATION

6.4.4.2 as7352_measurement_types `enum as7352_measurement_types`

List of supported measurement types for ITEM_ID_MEAS_TYPE

Enumerator

MEASUREMENT_TYPE_SPECTRAL	Spectral measurement: Measures six chanels in parallel, otherwise you can measure 12 channels in two blocks a 6 chanels (twice integration time!).
MEASUREMENT_TYPE_FIFO	FIFO measurement: Measures with one ADC only in fast FIFO mode: It is possible to map more than one channel to this ADC.
MEASUREMENT_TYPE_SPECTRAL_FIFO	Parallel FIFO and spectral measurement. Combinatation of mode MEASUREMENT_TYPE_SPECTRAL and MEASUREMENT_TYPE_FIFO with less options
MEASUREMENT_TYPE_GOERTZEL	On chip internal flicker detection: Uses internal flicker detection with Goertzel algorithm. Callback payload contains 2 bytes: • byte[0] : flicker frequency • byte[1] : FD_Status register value
MEASUREMENT_TYPE_NUM	Maximum supported measurement types.

6.4.4.3 as7352_channels `enum as7352_channels`

Supported channel types for spectral channels. Used to ITEM_ID_CHANNELS.

Enumerator

CHANNEL_DISABLED	place holder for unused channels
CHANNEL_F1	channel F1
CHANNEL_F2	channel F2
CHANNEL_FZ	channel FZ
CHANNEL_F3	channel F3
CHANNEL_F4	channel F4
CHANNEL_FY	channel FY
CHANNEL_F5	channel F5
CHANNEL_FXL	channel FXL
CHANNEL_F6	channel F6
CHANNEL_F7	channel F7
CHANNEL_F8	channel F8
CHANNEL_NIR	channel NIR
CHANNEL_FD	channel FLICKER
CHANNEL_DARK	channel DARK
CHANNEL_VISLT	channel VISLT
CHANNEL_VISRB	channel VISRB
CHANNEL_VISRT	channel VISRT
CHANNEL_VISLB	channel VISLB

6.4.4.4 as7352_gain enum as7352_gain

Supported gains for items ITEM_ID_AGAIN and ITEM_ID_FGAIN to set the spectral sensitivity.

Enumerator

GAIN_0_5X	gain of 0.5x
GAIN_1X	gain of 1x
GAIN_2X	gain of 2x
GAIN_4X	gain of 4x
GAIN_8X	gain of 8x
GAIN_16X	gain of 16x
GAIN_32X	gain of 32x
GAIN_64X	gain of 64x
GAIN_128X	gain of 128x
GAIN_256X	gain of 256x
GAIN_512X	gain of 512x
GAIN_1024X	gain of 1024x
GAIN_2048X	gain of 2048x
GAIN_4096X	gain of 4096x
GAIN_5120X	gain of 5120x

6.4.4.5 as7352_led_masks enum as7352_led_masks

This bit masks will be used for configuration of the LED pattern.

See also

ITEM_ID_LED_PATTERN

Enumerator

LED_MASK_OFF	mask that no LED will be set
LED_MASK_INTERN	mask that the internal LED will be set
LED_MASK_EXT_0	mask that the external LED 0 will be set
LED_MASK_EXT_1	mask that the external LED 1 will be set
LED_MASK_EXT_2	mask that the external LED 2 will be set
LED_MASK_EXT_3	mask that the external LED 3 will be set
LED_MASK_EXT_4	mask that the external LED 4 will be set
LED_MASK_EXT_5	mask that the external LED 5 will be set
LED_MASK_OUTPUT	mask that the output pin will be set

6.4.4.6 as7352_sync_mode enum as7352_sync_mode

Supported sync modes for ITEM_ID_SYNC_MODE

Enumerator

SYNC_MODE_DISABLED	Trigger measurement via GPIO pin disabled
SYNC_MODE_RISING_EDGE	Trigger measurement on rising edge of signal connected to GPIO pin
SYNC_MODE_FALLING_EDGE	Trigger measurement on falling edge of signal connected to GPIO pin
SYNC_MODE_NUMBER	Number of supported sync modes (not a valid payload value for ITEM_ID_SYNC_MODE)

6.4.4.7 as7352_item_ids enum as7352_item_ids

List of all supported item IDs

Enumerator

ITEM_ID_RESERVED	This ID will be used for special cases, like detection of undefined item.

ITEM_ID_ASTEP	This item access directly the register ASTEP 0xCA/0xCB. Those registers are one of the two parts which allows the user to set the integration time for spectral measurement.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_ASTEP (unsigned 16bit)
	• *Parameter range:* 1 - 65534
	• *Default value:* 599
	• *Direction:* write/read
	
	Example
	
	`uint16_t astep_set, astep_get;`
	`// set ASTEP-property`
	`astep_set = 999;`
	`as7352_set_item(DEV_ID, ITEM_ID_ASTEP, (void *)&astep_set,`
	`ITEM_SIZE_ASTEP);`
	`// get ASTEP-property`
	`as7352_get_item(DEV_ID, ITEM_ID_ASTEP, (void *)&astep_get,`
	`ITEM_SIZE_ASTEP);`
	
	Note
	
	If you want to set the integration time in microseconds directly, use ITEM_ID_ITIME. Here you can find additional information as well.

Enumerator

ITEM_ID_ATIME	This item accesses directly the AS7352 ATIME-register 0x81. ATIME is the second part to calculate the integration time.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_ATIME (unsigned 8bit)
	• *Parameter range:* 0 - 255
	• *Default value:* 29
	• *Direction:* write/read
	
	Example
	
	`uint8_t atime_set, atime_get;`
	`// set ATIME-property`
	`atime_set = 19;`
	`as7352_set_item(DEV_ID, ITEM_ID_ATIME, (void *)&atime_set,`
	`ITEM_SIZE_ATIME);`
	`// get ATIME-property`
	`as7352_get_item(DEV_ID, ITEM_ID_ATIME, (void *)&atime_get,`
	`ITEM_SIZE_ATIME);`
	
	Note
	
	If you want to set the integration time in microseconds directly, use ITEM_ID_ITIME. Here you can find additional information as well.

Enumerator

ITEM_ID_ITIME	This item calculates the integration time in microseconds. Internally, the registers ATIME and ASTEP will be set. Following formula describes the relationship:
	
	$t_{int} = (ATIME + 1) * (ASTEP + 1) * (2000/720)us$
	
	The maximum possible ADC fullscale range has a width of 16bit, but is limited by the integration time:
	
	$ADC_{fullscale} = (ATIME + 1) * (ASTEP + 1)$
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_ITIME (unsigned 32bit)
	• *Parameter range:* 6 - 46602667
	• *Default value:* 50000
	• *Direction:* write/read
	
	Example
	
	`uint32_t itime_set, itime_get;`
	`// set ITIME-property`
	`itime_set = 40000;`
	`as7352_set_item(DEV_ID, ITEM_ID_ITIME, (void *)&itime_set,`
	`ITEM_SIZE_ITIME);`
	`// get ITIME-property`
	`as7352_get_item(DEV_ID, ITEM_ID_ITIME, (void *)&itime_get,`
	`ITEM_SIZE_ITIME);`
	
	Attention
	
	It is normal that the value which you readback differs from the set value, because it depends on the resolution of ASTEP and ATIME

Enumerator

ITEM_ID_AGAIN	This item accesses directly the AS7352 CFG_1-register (0xAA) bits AGAIN to change the spectral sensitivity.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_AGAIN (unsigned 8bit)
	• *Parameter range:* see as7352_gain
	• *Default value:* GAIN_256X
	• *Direction:* write/read
	
	Example
	
	`uint8_t again_set, again_get;`
	`// set AGAIN-property`
	`again_set = GAIN_16X;`
	`as7352_set_item(DEV_ID, ITEM_ID_AGAIN, (void *)&again_set,`
	`ITEM_SIZE_AGAIN);`
	`// get AGAIN-property`
	`as7352_get_item(DEV_ID, ITEM_ID_AGAIN, (void *)&again_get,`
	`ITEM_SIZE_AGAIN);`

ITEM_ID_MEAS_TYPE	This item configures the measurement typ. (as7352_measurement_types)
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_MEAS_TYPE (unsigned 8bit)
	• *Parameter range:* see as7352_measurement_types
	• *Default value:* MEASUREMENT_TYPE_SPECTRAL
	• *Direction:* write/read
	
	Example
	
	`uint8_t meas_type_set, meas_type_get;`
	`// set MEAS_TYPE-property`
	`meas_type_set = MEASUREMENT_TYPE_FIFO;`
	`as7352_set_item(DEV_ID, ITEM_ID_MEAS_TYPE, (void`
	`*)&meas_type_set, ITEM_SIZE_MEAS_TYPE);`
	`// get MEAS_TYPE-property`
	`as7352_get_item(DEV_ID, ITEM_ID_MEAS_TYPE, (void`
	`*)&meas_type_get, ITEM_SIZE_MEAS_TYPE);`
	
	See also
	
	as7352_measurement_types

Enumerator

ITEM_ID_BREAK	This item specifies the break between the finished measurement and the start of the next measurement.
	On default this time is disabled and not used. This time depends on the integration time of the sensor because the value will be calculated by the difference of wait_time and integration_time-registers.
	If you want to readout the real configured break time, you must set integration time at first, then set break time itself and afterwards read it back.
	The maximum time of integration is much higher than the maximum possible wait time registers. So, you can't set the break time in all cases. To use this break time, decrease the integration time.
	Note
	The minimum possible break time is 2780us.
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_BREAK (unsigned 32bit)
	• *Parameter range:* 0, 2780us - 10000000us
	• *Default value:* 0
	• *Direction:* write/read
	Example
	`uint32_t break_set, break_get;`
	`// set BREAK-property`
	`break_set = 50000; 50ms`
	`as7352_set_item(DEV_ID, ITEM_ID_BREAK, (void *)&break_set, ITEM_SIZE_BREAK);`
	`// get BREAK-property`
	`as7352_get_item(DEV_ID, ITEM_ID_BREAK, (void *)&break_get, ITEM_SIZE_BREAK);`

Enumerator

ITEM_ID_CHANNELS	Configures up to 18 measurement channels, each byte for one channel. The possible values for every byte are described in as7352_channels. **Properties** • *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL • *Payload size:* ::ITEM_SIZE_CHANNELS (unsigned 18byte) • *Parameter range:* as7352_channels • *Default value:* CHANNEL_FZ CHANNEL_FY CHANNEL_FXL, CHANNEL_NIR, ::CHANNEL_CLEAR, ::CHANNEL_FLICKER, CHANNEL_F2, CHANNEL_F3, CHANNEL_F4, CHANNEL_F5, ::CHANNEL_CLEAR, ::CHANNEL_FLICKER, CHANNEL_F1, CHANNEL_F6, CHANNEL_F7, CHANNEL_F8, ::CHANNEL_CLEAR, ::CHANNEL_FLICKER, • *Direction:* write/read **Example** `// configure a single measuremt with 8 channels only uint8_t channels[ITEM_SIZE_CHANNELS_MAX] = {CHANNEL_F1, CHANNEL_F2, CHANNEL_F3, CHANNEL_F4, CHANNEL_F5, CHANNEL_F6, CHANNEL_F7, CHANNEL_F8, CHANNEL_DISABLED, CHANNEL_DISABLED, CHANNEL_DISABLED,CHANNEL_DISABLED, CHANNEL_DISABLED, CHANNEL_DISABLED, CHANNEL_DISABLED, CHANNEL_DISABLED}, CHANNEL_DISABLED, CHANNEL_DISABLED;` `// set CHANNELS-property` `as7352_set_item(DEV_ID, ITEM_ID_CHANNELS, (void *)channels, ITEM_SIZE_CHANNELS_MAX);` `// get CHANNELS-property` `as7352_get_item(DEV_ID, ITEM_ID_CHANNELS, (void *)channels, ITEM_SIZE_CHANNELS_MAX);` **See also** as7352_channels

Enumerator

ITEM_ID_VERSION	Every byte describes one version position: MAJOR MINOR PATCH BUILD. See structure as7352_version.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_VERSION (unsigned 4byte)
	• *Parameter range:* 0 - 255, 4 times
	• *Default value:* no default
	• *Direction:* read only
	
	Example
	
	`struct as7352_version version;`
	`as7352_get_item(DEV_ID, ITEM_ID_VERSION, (void *)&version,`
	`ITEM_SIZE_VERSION);`
	
	See also
	
	as7352_version

ITEM_ID_SERIAL	Unique chip identifier. See structure as7352_serial.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_SERIAL, see as7352_serial
	• *Parameter range:* see definition of as7352_serial
	• *Default value:* no default
	• *Direction:* read only
	
	Example
	
	`struct as7352_serial serial;`
	`as7352_get_item(DEV_ID, ITEM_ID_SERIAL, (void *)&serial,`
	`ITEM_SIZE_SERIAL);`
	
	See also
	
	as7352_serial

Enumerator

ITEM_ID_AUTOZERO	This item accesses directly the AS7352 AZ_CONFIG-register 0xD6. The register configures how often the spectral engine offsets are reset (auto zero) to compensate for changes of the device temperature. The typical time auto zero needs to be completed is 15ms.
	Parameter description:
	• 0: Never used
	• 1: Every integration cycle
	• 2: Every 2 integration cycle
	• ...
	• 254: Every 254 cycles
	• 255: Only before first measurement
	
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_AUTOZERO (unsigned 8bit)
	• *Parameter range:* 0 - 255
	• *Default value:* 255
	• *Direction:* write/read
	Limits
	• If ITEM_ID_MEAS_TYPE is set to MEASUREMENT_TYPE_SPECTRAL_FIFO, only a value of 0 and 255 is allowed.
	Example
	`uint8_t autozero_set, autozero_get;`
	`// set AUTOZERO-property`
	`autozero_set = 0;`
	`as7352_set_item(DEV_ID, ITEM_ID_AUTOZERO, (void *)&autozero_set, ITEM_SIZE_AUTOZERO);`
	`// get AUTOZERO-property`
	`as7352_get_item(DEV_ID, ITEM_ID_AUTOZERO, (void *)&autozero_get, ITEM_SIZE_AUTOZERO);`
	Attention
	The defintion of this item is valid, if at maximum 6 channels are configured. If more than 6 channels are used, auto zero will be restarted after every channel reconfiguration. That means, that value 1 - 255 has the same effect: Auto zero is used on every measurement.

Enumerator

ITEM_ID_MEAS_COUNT_ALS	Configured the number of measurements, 0 means continuous measurement.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL, MEASUREMENT_TYPE_SPECTRAL_FIFO
	• *Payload size:* ITEM_SIZE_MEAS_COUNT_ALS (unsigned 16bit)
	• *Parameter range:* 0 - 65535
	• *Default value:* 0 (continuous measurement)
	• *Direction:* write/read
	
	Example
	
	`uint16_t meas_count_set, meas_count_get;`
	`// set MEAS_COUNT-property`
	`meas_count_set = 5;`
	`as7352_set_item(DEV_ID, ITEM_ID_MEAS_COUNT_ALS, (void *)&meas_count_set, ITEM_SIZE_MEAS_COUNT_ALS);`
	`// get MEAS_COUNT-property`
	`as7352_get_item(DEV_ID, ITEM_ID_MEAS_COUNT_ALS, (void *)&meas_count_get, ITEM_SIZE_MEAS_COUNT_ALS);`
	
	Note
	
	For measurement mode MEASUREMENT_TYPE_SPECTRAL_FIFO both items ITEM_ID_MEAS_COUNT_ALS and ITEM_ID_MEAS_COUNT_FIFO can be set. Whichever value is reached first will cause the measurment to abort

Enumerator

ITEM_ID_LED_PATTERN	With the help of the structure as7352_led_pattern LEDs can be switched synchronously to the measurement. Inside the structure can be defined how long a specifc LED configuration shall be used.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_LED_PATTERN (size of structure as7352_led_pattern, 10x)
	• *Parameter range:* count: 0 - 255, config: see as7352_led_masks
	• *Default value:* count: 0, config: LED_MASK_OFF
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_pattern led_pattern[AS7352_LED_PATTERN_NUM] = {0, 0};`
	`// configure the LED pattern:`
	`// first two measurements:  LEDs off,`
	`led_pattern[0].config = LED_MASK_OFF;`
	`led_pattern[0].count = 2;`
	`// next three measurements:  internal LED activated`
	`led_pattern[1].config = LED_MASK_INTERN;`
	`led_pattern[1].count = 3;`
	`// next four measurements:  internal LED and external LED_0 are activated`
	`led_pattern[2].config = LED_MASK_EXT_0 | LED_MASK_INTERN;`
	`led_pattern[2].count = 4;`
	`// set LED_PATTERN-property`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_PATTERN, (void *)led_pattern, ITEM_SIZE_LED_PATTERN);`
	`// get LED_PATTERN-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_PATTERN, (void *)led_pattern, ITEM_SIZE_LED_PATTERN);`
	`\b Limits`
	`- GPIO3 controls the external LEDs, GPIO3 can only be used exclusive.`
	`- So, ext.  Leds cannot be used together with sync mode.`
	`- ITEM_ID_SYNC_MODE needs to be set to SYNC_MODE_DISABLED, which is default.`

Enumerator

ITEM_ID_LED_WAIT_TIME	Set warm up time in microseconds for synchronized LED switching.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_LED_WAIT_TIME (unsigned 32bit)
	• *Parameter range:* 0us - 10s
	• *Default value:* 100000 (100ms)
	• *Direction:* write/read
	
	Example
	
	`uint32_t warm_up_time_set, warm_up_time_get;`
	`// set LED_WAIT_TIME-property`
	`warm_up_time_set = 0;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_WAIT_TIME, (void *)&warm_up_time_set, ITEM_SIZE_LED_WAIT_TIME);`
	`// get LED_WAIT_TIME-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_WAIT_TIME, (void *)&warm_up_time_get, ITEM_SIZE_LED_WAIT_TIME);`

ITEM_ID_INTERRUPT_PIN	Unequal zero means that the interrupt pin will be used, otherwise the time of integration will be calculated internally.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL, MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_INTERRUPT_PIN (unsigned 8bit)
	• *Parameter range:* 0 - 1
	• *Default value:* 0
	• *Direction:* write/read
	
	Example
	
	`uint8_t interrupt_pin_set, interrupt_pin_get;`
	`// set INTERRUPT_PIN-property`
	`interrupt_pin_set = 1;`
	`as7352_set_item(DEV_ID, ITEM_ID_INTERRUPT_PIN, (void *)&interrupt_pin_set, ITEM_SIZE_INTERRUPT_PIN);`
	`// get INTERRUPT_PIN-property`
	`as7352_get_item(DEV_ID, ITEM_ID_INTERRUPT_PIN, (void *)&interrupt_pin_get, ITEM_SIZE_INTERRUPT_PIN);`

Enumerator

ITEM_ID_LED_INTERN	Configures the internal LED with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_INTERN (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_INTERN-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_INTERN, (void *)&config_set, ITEM_SIZE_LED_INTERN);`
	`// get LED_INTERN-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_INTERN, (void *)&config_get, ITEM_SIZE_LED_INTERN);`

ITEM_ID_LED_EXT_0	Configures the external LED 0 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_0 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_0-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_0, (void *)&config_set, ITEM_SIZE_LED_EXT_0);`
	`// get LED_EXT_0-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_0, (void *)&config_get, ITEM_SIZE_LED_EXT_0);`
	`\b Limits`
	`- GPIO3 controls the external LEDs, GPIO3 can only be used exclusive.`
	`- So, ext.  Leds cannot be used together with sync mode.`
	`- ITEM_ID_SYNC_MODE needs to be set to SYNC_MODE_DISABLED, which is default.`

Enumerator

ITEM_ID_LED_EXT_1	Configures the external LED 1 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_1 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_1-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_1, (void *)&config_set, ITEM_SIZE_LED_EXT_1);`
	`// get LED_EXT_1-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_1, (void *)&config_get, ITEM_SIZE_LED_EXT_1);`
	`\b Limits`
	`- GPIO3 controls the external LEDs, GPIO3 can only be used exclusive.`
	`- So, ext. Leds cannot be used together with sync mode.`
	`- ITEM_ID_SYNC_MODE needs to be set to SYNC_MODE_DISABLED, which is default.`

ITEM_ID_LED_EXT_2	Configures the external LED 2 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_2 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_2-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_2, (void *)&config_set, ITEM_SIZE_LED_EXT_2);`
	`// get LED_EXT_2-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_2, (void *)&config_get, ITEM_SIZE_LED_EXT_2);`
	`\b Limits`
	`- GPIO3 controls the external LEDs, GPIO3 can only be used exclusive.`
	`- So, ext. Leds cannot be used together with sync mode.`
	`- ITEM_ID_SYNC_MODE needs to be set to SYNC_MODE_DISABLED, which is default.`

Enumerator

ITEM_ID_LED_EXT_3	Configures the external LED 3 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_3 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_3-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_3, (void *)&config_set, ITEM_SIZE_LED_EXT_3);`
	`// get LED_EXT_3-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_3, (void *)&config_get, ITEM_SIZE_LED_EXT_3);`
	`\b Limits`
	`- GPIO3 controls the external LEDs, GPIO3 can only be used exclusive.`
	`- So, ext. Leds cannot be used together with sync mode.`
	`- ITEM_ID_SYNC_MODE needs to be set to SYNC_MODE_DISABLED, which is default.`

ITEM_ID_LED_EXT_4	Configures the external LED 4 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_4 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_4-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_4, (void *)&config_set, ITEM_SIZE_LED_EXT_4);`
	`// get LED_EXT_4-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_4, (void *)&config_get, ITEM_SIZE_LED_EXT_4);`
	`\b Limits`
	`- GPIO3 controls the external LEDs, GPIO3 can only be used exclusive.`
	`- So, ext. Leds cannot be used together with sync mode.`
	`- ITEM_ID_SYNC_MODE needs to be set to SYNC_MODE_DISABLED, which is default.`

Enumerator

ITEM_ID_LED_EXT_5	Configures the external LED 5 with help of structure as7352_led_config.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_LED_EXT_5 (twice unsigned 16bit)
	• *Parameter range:* see as7352_led_config
	• *Default value:* enable: 0, brightness: 100
	• *Direction:* write/read
	
	Example
	
	`struct as7352_led_config config_set, config_get;`
	`// set LED_EXT_5-property`
	`config_set.enable = 1;`
	`config_set.brightness = 1000;`
	`as7352_set_item(DEV_ID, ITEM_ID_LED_EXT_5, (void *)&config_set, ITEM_SIZE_LED_EXT_5);`
	`// get LED_EXT_5-property`
	`as7352_get_item(DEV_ID, ITEM_ID_LED_EXT_5, (void *)&config_get, ITEM_SIZE_LED_EXT_5);`
	`\b Limits`
	`- GPIO3 controls the external LEDs, GPIO3 can only be used exclusive.`
	`- So, ext. Leds cannot be used together with sync mode.`
	`- ITEM_ID_SYNC_MODE needs to be set to SYNC_MODE_DISABLED, which is default.`

ITEM_ID_OUTPUT	The output is configured as an open collector. On default the output is HI-Z and is disconnected. Unequal zero means that the output pin drives low.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_OUTPUT (unsigned 8bit)
	• *Parameter range:* 0 (HI-Z) - 1 (GND)
	• *Default value:* 0 (HI-Z)
	• *Direction:* write/read
	
	Example
	
	`uint8_t output_set, output_get;`
	`// set OUTPUT-property`
	`output_set = 1`
	`as7352_set_item(DEV_ID, ITEM_ID_OUTPUT, (void *)&output_set, ITEM_SIZE_OUTPUT);`
	`// get OUTPUT-property`
	`as7352_get_item(DEV_ID, ITEM_ID_OUTPUT, (void *)&output_get, ITEM_SIZE_OUTPUT);`

Enumerator

ITEM_ID_TEMP_EXT_0	Get the temperature of an external sensor 0 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_0 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_0-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_0, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_0);`

ITEM_ID_TEMP_EXT_1	Get the temperature of an external sensor 1 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_1 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_1-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_1, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_1);`

Enumerator

ITEM_ID_TEMP_EXT_2	Get the temperature of an external sensor 2 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_2 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_2-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_2, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_2);`

ITEM_ID_TEMP_EXT_3	Get the temperature of an external sensor 3 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_3 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_3-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_3, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_3);`

Enumerator

ITEM_ID_TEMP_EXT_4	Get the temperature of an external sensor 4 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_4 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_4-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_4, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_4);`

ITEM_ID_TEMP_EXT_5	Get the temperature of an external sensor 5 in millidegree.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TEMP_EXT_5 (signed 32bit)
	• *Parameter range:* depends on OSAL implementation (typical -40000 ... 125000)
	• *Default value:* -
	• *Direction:* read
	
	Example
	
	`int32_t millidegree_get;`
	`// get TEMP_EXT_5-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TEMP_EXT_5, (void *)&millidegree_get, ITEM_SIZE_TEMP_EXT_5);`

Enumerator

ITEM_ID_MEASURE_ITEMS	Item describes additional measurement data. The list can be fullfilled with any item which shall be readout synchronized to the measurement. Those item content will be transferred with the callback function.
	Note
	The maximum supported buffer size in callback function is AS7352_MAX_ITEM_BUFFER_SIZE.
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL, MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_MEASURE_ITEMS (10x unsigned 8bit)
	• *Parameter range:* 0 .. ITEM_ID_MAX - 1
	• *Default value:* ITEM_ID_RESERVED (10 times)
	• *Direction:* write/read
	Example
	`uint8_t measure_items[ITEM_SIZE_MEASURE_ITEMS] = {ITEM_ID_RESERVED};`
	`// set MEASURE_ITEMS-property: added items for temperature sensor`
	`// and gain to cyclic measurement data`
	`measure_items[0] = ITEM_ID_TEMP_EXT_0;`
	`measure_items[1] = ITEM_ID_AGAIN;`
	`as7352_set_item(DEV_ID, ITEM_ID_MEASURE_ITEMS, (void *)&measure_items, ITEM_SIZE_MEASURE_ITEMS);`
	`// get MEASURE_ITEMS-property`
	`as7352_get_item(DEV_ID, ITEM_ID_MEASURE_ITEMS (void *)&measure_items, ITEM_SIZE_MEASURE_ITEMS);`

Enumerator

ITEM_ID_FGAIN	Configures the GAIN for fast FIFO measurement mode. Internally, the register FD_TIME_2 is used.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_FGAIN (unsigned 8bit)
	• *Parameter range:* see as7352_gain
	• *Default value:* GAIN_16X
	• *Direction:* write/read
	
	Example
	
	`uint8_t fgain_set, fgain_get;`
	`// set FGAIN-property`
	`fgain_set = GAIN_256X;`
	`as7352_set_item(DEV_ID, ITEM_ID_FGAIN, (void *)&fgain_set,`
	`ITEM_SIZE_FGAIN);`
	`// get FGAIN-property`
	`as7352_get_item(DEV_ID, ITEM_ID_FGAIN, (void *)&fgain_get,`
	`ITEM_SIZE_FGAIN);`

Enumerator

ITEM_ID_FTIME	Configures the integration time for fast FIFO measurement mode: This parameter sets the I2C registers FD_TIME_1 and FD_TIME_2 directly.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_FTIME (unsigned 16bit)
	• *Parameter range:* 0 - 0x07FF
	• *Default value:* 359
	• *Direction:* write/read
	
	Note
	
	If FTIME is smaller or equal 255, the resulting ADC values also cannot be greater than 255. Therefore, the 8-Bit FIFO mode is automatically used. In this mode, the FIFO only stores 8-Bit values, which can be read out in half the time. Other than speed, the 8-Bit mode has no effect on data aquisition. The data will still be saved in the 16-Bit data buffer to remain compatible with the rest of the chip library.
	
	Example
	
	`uint16_t ftime_set, ftime_get;`
	`// set FTIME-property`
	`ftime_set = 179;`
	`as7352_set_item(DEV_ID, ITEM_ID_FTIME, (void *)&ftime_set,`
	`ITEM_SIZE_FTIME);`
	`// get FTIME-property`
	`as7352_get_item(DEV_ID, ITEM_ID_FTIME, (void *)&ftime_get,`
	`ITEM_SIZE_FTIME);`
	
	See also
	
	ITEM_ID_FTIME_US

Enumerator

ITEM_ID_FTIME_US	Configures the integration time for fast FIFO measurement mode in microseconds.
	Following formula describes the relationship:
	$t_{int} = (FD\_TIME + 1) * (2000/720)us$
	The maximum possible ADC fullscale range has a width of 16bit, but is limited by the integration time:
	$ADC_{fullscale} = FD\_TIME + 1$
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_FTIME_US (unsigned 32bit)
	• *Parameter range:* 3us - 5689us
	• *Default value:* 1000us
	• *Direction:* write/read
	Example
	`uint32_t ftime_us_set, ftime_us_get;`
	`// set FTIME_US-property`
	`ftime_set = 500; // set to 500us == 2kHz`
	`as7352_set_item(DEV_ID, ITEM_ID_FTIME_US, (void *)&ftime_us_set, ITEM_SIZE_FTIME_US);`
	`// get FTIME_US-property`
	`as7352_get_item(DEV_ID, ITEM_ID_FTIME_US, (void *)&ftime_us_get, ITEM_SIZE_FTIME_US);`

ITEM_ID_TIMESTAMP	Reads a microseconds timestamp. It can be used to get accurate time difference between to measurements.
	Properties
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_TIMESTAMP (unsigned 64bit)
	• *Parameter range:* full 64bit range
	• *Default value:* -
	• *Direction:* read only
	Example
	`uint64_t timestamp_get;`
	`// get TIMESTAMP-property`
	`as7352_get_item(DEV_ID, ITEM_ID_TIMESTAMP, (void *)&timestamp_get, ITEM_SIZE_TIMESTAMP);`

Enumerator

ITEM_ID_AUTO_GAIN_RANGE_ALS	Enables or disables the automatic gain control for spectral measurements. If lower and upper limit are different and the lower value is lower than the upper one, auto gain is enabled. In this case the gain will be changed automatically. The algorithm tries to map the highest ADC-value to 80% of the maximum. The configured default gain of item ITEM_ID_AGAIN will be used as start value. If this value is out of the configured range, then the lower limit will be used instead. Dependent on the current measurement mode, the item sets ITEM_ID_AGAIN or will be set internally.
	Note
	It's usefull to configure the item ITEM_ID_MEASURE_ITEMS with ITEM_ID_AGAIN to normalize the measured data in the application software.
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL, MEASUREMENT_TYPE_SPECTRAL_FIFO
	• *Payload size:* ITEM_SIZE_AUTO_GAIN_RANGE_ALS (see as7352_auto_gain)
	• *Parameter range:* see as7352_gain
	• *Default value:* GAIN_0_5X, GAIN_0_5X
	• *Direction:* write/read
	Example
	`struct as7352_auto_gain auto_gain;`
	`// Enable auto gain over the whole range.`
	`auto_gain.lower_limit = GAIN_0_5X;`
	`auto_gain.upper_limit = GAIN_5120X,`
	`as7352_set_item(DEV_ID, ITEM_ID_AUTO_GAIN_RANGE_ALS, (void *)&auto_gain, ITEM_SIZE_AUTO_GAIN_RANGE_ALS);`

Enumerator

ITEM_ID_AUTO_GAIN_RANGE_FIFO	Enables or disables the automatic gain control for FiFo measurements. If lower and upper limit are different and the lower value is lower than the upper one, auto gain is enabled. In this case the gain will be changed automatically. The algorithm tries to map the highest ADC-value to 80% of the maximum. The configured default gain of item ITEM_ID_FGAIN will be used as start value. If this value is out of the configured range, then the lower limit will be used instead. Dependent on the current measurement mode, the item sets ITEM_ID_AGAIN or ITEM_ID_FGAIN will be set internally.
	Note
	It's useful to configure the item ITEM_ID_MEASURE_ITEMS with ITEM_ID_FGAIN to normalize the measured data in the application software.
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL_FIFO, MEASUREMENT_TYPE_FIFO
	• *Payload size:* ITEM_SIZE_AUTO_GAIN_RANGE_FIFO (see as7352_auto_gain)
	• *Parameter range:* see as7352_gain
	• *Default value:* GAIN_0_5X, GAIN_0_5X
	• *Direction:* write/read
	Example
	`struct as7352_auto_gain auto_gain;`
	`// Enable auto gain over the whole range.`
	`auto_gain.lower_limit = GAIN_0_5X;`
	`auto_gain.upper_limit = GAIN_5120X,`
	`as7352_set_item(DEV_ID, ITEM_ID_AUTO_GAIN_RANGE_ALS, (void *)&auto_gain, ITEM_SIZE_AUTO_GAIN_RANGE_FIFO);`

Enumerator

ITEM_ID_GAIN_FACTORS	The sensor AS7352 has a gain error over the whole area. If the gain will be increased by one, the measured value is not exact the double. To compensate this behavior, the measured values will be multiplied with a correction factor. The default values are used from the datasheet but can be changed.
	The given factors will be divided by 10000 internally to get the right factor with four decimal places.
	Note
	The feature can be disabled by setting all factors to 10000.
	Properties
	• *Measurement mode:* MEASUREMENT_TYPE_SPECTRAL
	• *Payload size:* ITEM_SIZE_GAIN_FACTORS (15x unsigned short)
	• *Parameter range:* 1 - 20000
	• *Default value:* 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000 (gain GAIN_0_5X - GAIN_5120X)
	• *Direction:* write/read
	Note
	TODO: Gain factors are set to 100%. After validation these should be updated
	Example
	`uint16_t gain_factors[AS7352_GAIN_FACTOR_NUM];`
	`// Read gain factors`
	`as7352_get_item(DEV_ID, ITEM_ID_GAIN_FACTORS, (void *)gain_factors, ITEM_SIZE_GAIN_FACTORS);`

Enumerator

ITEM_ID_REG_READ	Read a register. This item implements a read register access as a two step process where the register address needs to written to the item first. The subsequent read of this item will return the register value. For register addresses < 0x80, the register bank switch is performed automatically, if necessary. After a successful read operation, the address pointer will be incremented by 1
	Note
	On initialization, the address pointer is at 0x00
	Properties
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_REG_READ (unsigned 8bit)
	• *Parameter range:* 0 - 255
	• *Default value:* 0
	• *Direction:* read / write
	Example
	`uint8_t reg_addr = AS7352_REGADDR_STATUS uint8_t value;`
	`// set address pointer to AS7352_REGADDR_STATUS`
	`as7352_set_item(DEV_ID, ITEM_ID_REG_READ, (void *)&reg_addr, sizeof(uint8_t));`
	`// read register AS7352_REGADDR_STATUS`
	`as7352_get_item(DEV_ID, ITEM_ID_REG_READ, (void *)&value, ITEM_SIZE_REG_READ);`

ITEM_ID_SYNC_MODE	Configures the sync mode of the sensor. When sync mode is enabled, the sensor triggers a measurement either on a rising or falling edge of the signal connected to the GPIO pin.
	Properties
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_SYNC_MODE (unsigned 8bit)
	• *Parameter range:* see as7352_sync_mode
	• *Default value:* SYNC_MODE_DISABLED
	• *Direction:* read / write
	Example
	`uint8_t sync_mode = SYNC_MODE_DISABLED;`
	`as7352_set_item(DEV_ID, ITEM_ID_SYNC_MODE, (void *)&sync_mode, ITEM_SIZE_SYNC_MODE);`
	`as7352_get_item(DEV_ID, ITEM_ID_SYNC_MODE, (void *)&sync_mode, ITEM_SIZE_SYNC_MODE);`
	`\b Limits`
	`- Sync mode is using GPIO3 and can only be used exclusive.`
	`- So, ext. Leds which are controlled by GPIO3 cannot be used together with sync mode.`
	`- ITEM_ID_LED_PATTERN for ext. LEDs and ITEM_ID_LED_EXT_[0..5] need to be disabled to use sync.`

Enumerator

ITEM_ID_FIFO_AGC_BLOCKSIZE	Configures the blocksize for fifo measurements, after every block a AGC cycle will be executed. So the application can ensure that it will receive at least one consistent block between 2 AGC cycles to calculate a FFT from it. So, a blocksize should be >= FFT window size. A blocksize of 0 will cause an AGC cycle at the beginning of measurement and will then run infinetely without any further AGC.
	
	Properties
	
	• *Measurement mode:* -
	• *Payload size:* ITEM_SIZE_FIFO_AGC_BLOCKSIZE (unsigned 8bit)
	• *Parameter range:* full 16bit range
	• *Default value:* 0
	• *Direction:* read / write
	
	Example
	
	`uint16_t block_size = 1024;`
	`as7352_set_item(DEV_ID, ITEM_ID_FIFO_AGC_BLOCKSIZE, (void *)&block_size, ITEM_SIZE_FIFO_AGC_BLOCKSIZE);`
	`as7352_get_item(DEV_ID, ITEM_ID_FIFO_AGC_BLOCKSIZE, (void *)&block_size, ITEM_SIZE_FIFO_AGC_BLOCKSIZE);`

ITEM_ID_FD_COEFF	Configure the number of datasets for the detection of the flicker frequency using onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_COEFF (unsigned 8bit)
	• *Parameter range:* 0-7
	• *Default value:* 4
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_coeff = 3;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_COEFF, (void *)&fd_coeff, ITEM_SIZE_FD_COEFF);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_COEFF, (void *)&fd_coeff, ITEM_SIZE_FD_COEFF);`

Enumerator

ITEM_ID_FD_DCR	Configure the window size of the DC removal for the detection of the flicker frequency using onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_DCR (unsigned 8bit)
	• *Parameter range:* 0-7
	• *Default value:* 3
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_dcr = 3;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_DCR, (void *)&fd_dcr,`
	`ITEM_SIZE_FD_DCR);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_DCR, (void *)&fd_dcr,`
	`ITEM_SIZE_FD_DCR);`

ITEM_ID_FD_PERS	Configure the number of consecutive flicker frequency detections, so that the detection is accepted to be valid at onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_PERS (unsigned 8bit)
	• *Parameter range:* 0-7
	• *Default value:* 2
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_pers = 3;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_PERS, (void *)&fd_pers,`
	`ITEM_SIZE_FD_PERS);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_PERS, (void *)&fd_pers,`
	`ITEM_SIZE_FD_PERS);`

Enumerator

ITEM_ID_FD_AGC_DISABLE	Disable (1) or enable (0) the automatic gain control for detection of flicker frequencies at onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_AGC_DISABLE (unsigned 8bit)
	• *Parameter range:* 0-1
	• *Default value:* 0
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_agc_disable = 1;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_AGC_DISABLE, (void *)&fd_agc_disable, ITEM_SIZE_FD_AGC_DISABLE);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_AGC_DISABLE, (void *)&fd_agc_disable, ITEM_SIZE_FD_AGC_DISABLE);`

ITEM_ID_FD_MODCLK	Configure the clock speeds for detection of the flicker frequency using onchip Goertzel algorithm.
	
	• Bits 0-1: Modclk speed dur-ing flicker integration.
	• Bits 2-3: Modclk speed dur-ing flicker residual measurements
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_MODCLK (unsigned 8bit)
	• *Parameter range:* Bits[0:1] 0-3 ; Bits[2:3] 0-3
	• *Default value:* Bits[0:1] 0 ; Bits[2:3] 0
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_mod_clk = 3;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_MODCLK, (void *)&fd_mod_clk, ITEM_SIZE_FD_MODCLK);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_MODCLK, (void *)&fd_mod_clk, ITEM_SIZE_FD_MODCLK);`

Enumerator

ITEM_ID_FD_SAMPLES	Configure the number of samples used for detection of the flicker frequency using onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_SAMPLES (unsigned 8bit)
	• *Parameter range:* 0-7
	• *Default value:* 4
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_samples = 6;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_SAMPLES, (void *)&fd_samples, ITEM_SIZE_FD_SAMPLES);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_SAMPLES, (void *)&fd_samples, ITEM_SIZE_FD_SAMPLES);`

ITEM_ID_FD_COMPARE_VALUE	Configure the compare limit for the detection of the flick-er frequency using onchip Goertzel algorithm.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_GOERTZEL
	• *Payload size:* ITEM_SIZE_FD_COMPARE_VALUE (unsigned 8bit)
	• *Parameter range:* 0-7
	• *Default value:* 1
	• *Direction:* read / write
	
	Example
	
	`uint16_t fd_compare_value = 3;`
	`as7352_set_item(DEV_ID, ITEM_ID_FD_COMPARE_VALUE, (void *)&fd_compare_value, ITEM_SIZE_FD_COMPARE_VALUE);`
	`as7352_get_item(DEV_ID, ITEM_ID_FD_COMPARE_VALUE, (void *)&fd_compare_value, ITEM_SIZE_FD_COMPARE_VALUE);`

Enumerator

ITEM_ID_MEAS_COUNT_FIFO	Configure the number of measurements in a buffered measurement, 0 means continuous measurement.
	
	Properties
	
	• *Measurement mode:* MEASUREMENT_TYPE_FIFO, MEASUREMENT_TYPE_SPECTRAL_FIFO
	• *Payload size:* ITEM_SIZE_MEAS_COUNT_FIFO (unsigned 32bit)
	• *Parameter range:* 0 - 0xFFFFFFFF
	• *Default value:* 0 (continuous measurement)
	• *Direction:* write/read
	
	Example
	
	`uint32_t meas_count_set, meas_count_get;`
	`// set MEAS_COUNT-property`
	`meas_count_set = 20000;`
	`as7352_set_item(DEV_ID, ITEM_ID_MEAS_COUNT_FIFO, (void *)&meas_count_set, ITEM_SIZE_MEAS_COUNT_FIFO);`
	`// get MEAS_COUNT-property`
	`as7352_get_item(DEV_ID, ITEM_ID_MEAS_COUNT_FIFO, (void *)&meas_count_get, ITEM_SIZE_MEAS_COUNT_FIFO);`
	
	Note
	
	For measurement mode MEASUREMENT_TYPE_SPECTRAL_FIFO both items ITEM_ID_MEAS_COUNT_ALS and ITEM_ID_MEAS_COUNT_FIFO can be set. Whichever value is reached first will cause the measurement to abort

ITEM_ID_MAX	Internal definition of maximum supported items.

6.4.4.8 as7352_item_sizes `enum as7352_item_sizes`

List, which describes the supported length of every item.

Enumerator

ITEM_SIZE_RESERVED	size in bytes of item ITEM_ID_RESERVED
ITEM_SIZE_ASTEP	size in bytes of item ITEM_ID_ASTEP
ITEM_SIZE_ATIME	size in bytes of item ITEM_ID_ATIME
ITEM_SIZE_ITIME	size in bytes of item ITEM_ID_ITIME
ITEM_SIZE_AGAIN	size in bytes of item ITEM_ID_AGAIN
ITEM_SIZE_MEAS_TYPE	size in bytes of item ITEM_ID_MEAS_TYPE
ITEM_SIZE_BREAK	size in bytes of item ITEM_ID_BREAK
ITEM_SIZE_CHANNELS_MAX	size in bytes of item ITEM_ID_CHANNELS

Enumerator

ITEM_SIZE_VERSION	size in bytes of item ITEM_ID_VERSION
ITEM_SIZE_SERIAL	size in bytes of item ITEM_ID_SERIAL
ITEM_SIZE_SATURATION	size in bytes of item ITEM_ID_SATURATION
ITEM_SIZE_AUTOZERO	size in bytes of item ITEM_ID_AUTOZERO
ITEM_SIZE_MEAS_COUNT_ALS	size in bytes of item ITEM_ID_MEAS_COUNT_ALS
ITEM_SIZE_MEAS_COUNT_FIFO	size in bytes of item ITEM_ID_MEAS_COUNT_FIFO
ITEM_SIZE_LED_PATTERN	size in bytes of item ITEM_ID_LED_PATTERN
ITEM_SIZE_LED_WAIT_TIME	size in bytes of item ITEM_ID_LED_WAIT_TIME
ITEM_SIZE_INTERRUPT_PIN	size in bytes of item ITEM_ID_INTERRUPT_PIN
ITEM_SIZE_LED_INTERN	size in bytes of item ITEM_ID_LED_INTERN
ITEM_SIZE_LED_EXT_0	size in bytes of item ITEM_ID_LED_EXT_0
ITEM_SIZE_LED_EXT_1	size in bytes of item ITEM_ID_LED_EXT_1
ITEM_SIZE_LED_EXT_2	size in bytes of item ITEM_ID_LED_EXT_2
ITEM_SIZE_LED_EXT_3	size in bytes of item ITEM_ID_LED_EXT_3
ITEM_SIZE_LED_EXT_4	size in bytes of item ITEM_ID_LED_EXT_4
ITEM_SIZE_LED_EXT_5	size in bytes of item ITEM_ID_LED_EXT_5
ITEM_SIZE_OUTPUT	size in bytes of item ITEM_ID_OUTPUT
ITEM_SIZE_TEMP_EXT_0	size in bytes of item ITEM_ID_TEMP_EXT_0
ITEM_SIZE_TEMP_EXT_1	size in bytes of item ITEM_ID_TEMP_EXT_1
ITEM_SIZE_TEMP_EXT_2	size in bytes of item ITEM_ID_TEMP_EXT_2
ITEM_SIZE_TEMP_EXT_3	size in bytes of item ITEM_ID_TEMP_EXT_3
ITEM_SIZE_TEMP_EXT_4	size in bytes of item ITEM_ID_TEMP_EXT_4
ITEM_SIZE_TEMP_EXT_5	size in bytes of item ITEM_ID_TEMP_EXT_5
ITEM_SIZE_MEASURE_ITEMS	size in bytes of item ITEM_ID_MEASURE_ITEMS
ITEM_SIZE_FGAIN	size in bytes of item ITEM_ID_FGAIN
ITEM_SIZE_FTIME	size in bytes of item ITEM_ID_FTIME
ITEM_SIZE_FTIME_US	size in bytes of item ITEM_ID_FTIME_US
ITEM_SIZE_TIMESTAMP	size in bytes of item ITEM_ID_TIMESTAMP
ITEM_SIZE_AUTO_GAIN_RANGE_ALS	size in bytes of item ITEM_ID_AUTO_GAIN_ALS
ITEM_SIZE_AUTO_GAIN_RANGE_FIFO	size in bytes of item ITEM_ID_AUTO_GAIN_FIFO
ITEM_SIZE_GAIN_FACTORS	size in bytes of item ITEM_ID_GAIN_FACTOR
ITEM_SIZE_REG_READ	size in bytes of item ITEM_ID_REG_READ
ITEM_SIZE_SYNC_MODE	size in bytes of item ITEM_ID_SYNC_MODE
ITEM_SIZE_FIFO_AGC_BLOCKSIZE	size in bytes of item ITEM_ID_FIFO_AGC_BLOCKSIZE
ITEM_SIZE_FD_COEFF	size in bytes of item ITEM_ID_FD_COEFF
ITEM_SIZE_FD_DCR	size in bytes of item ITEM_ID_FD_DCR
ITEM_SIZE_FD_PERS	size in bytes of item ITEM_ID_FD_PERS
ITEM_SIZE_FD_AGC_DISABLE	size in bytes of item ITEM_ID_FD_AGC_DISABLE
ITEM_SIZE_FD_MODCLK	size in bytes of item ITEM_ID_FD_MODCLK
ITEM_SIZE_FD_SAMPLES	size in bytes of item ITEM_ID_FD_SAMPLES
ITEM_SIZE_FD_COMPARE_VALUE	size in bytes of item ITEM_ID_FD_COMPARE_VALUE

6.4.4.9 as7352_states enum as7352_states

This enumeration describes the state of the measurement engine

Enumerator

STATE_CONFIG	No measurement is running.
STATE_MEASURE	Measurement is active.

7 Data Structure Documentation

7.1 as7352_auto_gain Struct Reference

Data Fields

- uint8_t lower_limit
- uint8_t upper_limit

7.1.1 Detailed Description

Configuration of the auto gain algorithm

7.1.2 Field Documentation

7.1.2.1 lower_limit `uint8_t as7352_auto_gain::lower_limit`

See enumeration as7352_gain

7.1.2.2 upper_limit `uint8_t as7352_auto_gain::upper_limit`

See enumeration as7352_gain

7.2 as7352_led_config Struct Reference

Data Fields

- uint16_t enable
- uint16_t brightness

7.2.1 Detailed Description

Configuration of the LEDs with help of ITEM_ID_LED_INTERN, ITEM_ID_LED_EXT_0 ...

7.2.2 Field Documentation

7.2.2.1 enable `uint16_t as7352_led_config::enable`

Unequal zero means that LED shall be enabled, otherwise disabled.

7.2.2.2 brightness `uint16_t as7352_led_config::brightness`

Brightness value in per mile (0 - 1000)

7.3 as7352_led_pattern Struct Reference

Data Fields

- uint8_t count
- uint8_t config

7.3.1 Detailed Description

Definition for ITEM_ID_LED_PATTERN. This is used to switch the LEDs synchronized to the spectral measurement. During the measurement, the LED can only be switched on or off. The current must be configured with as7352_led_config.

7.3.2 Field Documentation

7.3.2.1 count `uint8_t as7352_led_pattern::count`

Defines how often the current LED configuration will be used.

7.3.2.2 config `uint8_t as7352_led_pattern::config`

Select the LEDs for the next measurement cycles. See as7352_led_masks.

7.4 as7352_serial Struct Reference

Data Fields

- uint32_t timestamp
- uint32_t id

7.4.1 Detailed Description

Payload definition of item ITEM_ID_SERIAL

7.4.2 Field Documentation

7.4.2.1 timestamp `uint32_t as7352_serial::timestamp`

unix timestamp of manufacturing time

7.4.2.2 id `uint32_t as7352_serial::id`

For parallel production, this ID makes the timestamp unique.

7.5 as7352_version Struct Reference

Data Fields

- uint8_t major
- uint8_t minor
- uint8_t patch
- uint8_t build

7.5.1 Detailed Description

Payload definition of item ITEM_ID_VERSION

7.5.2 Field Documentation

7.5.2.1 major `uint8_t as7352_version::major`

first position of version data: major version number

7.5.2.2 minor `uint8_t as7352_version::minor`

second position of version data: minor version number

7.5.2.3 patch `uint8_t as7352_version::patch`

third position of version data: patch version number

7.5.2.4 build `uint8_t as7352_version::build`

fourth position of version data: build version number

7.6 spectral_osal_id Struct Reference

Data Fields

- uint16_t chip
- uint8_t dev

7.6.1 Detailed Description

Specifies the device type and id to link to the right driver.

7.6.2 Field Documentation

7.6.2.1 chip `uint16_t spectral_osal_id::chip`

The numeric description of the used sensor, e.g. AS1234 -> chip = 1234.

7.6.2.2 dev `uint8_t spectral_osal_id::dev`

Defines which number of device shall be used [0 .. NUM_SUPPORTED_DEVICES-1].

Index