

OSIRE® E3731i OSP
V1.1.0

Generated by Doxygen 1.9.5

1 OSIRE® E3731i OSP	1
2 Data Structure Index	3
2.1 Data Structures	3
3 File Index	5
3.1 File List	5
4 Data Structure Documentation	7
4.1 ComStatus_t Struct Reference	7
4.1.1 Detailed Description	7
4.1.2 Field Documentation	7
4.1.2.1 address	7
4.1.2.2	7
4.1.2.3 comStatus	7
4.1.2.4	8
4.1.2.5 reserved	8
4.1.2.6 sio1_state	8
4.1.2.7 sio2_state	8
4.2 InitRsp_t Struct Reference	8
4.2.1 Detailed Description	8
4.2.2 Field Documentation	8
4.2.2.1 address	8
4.2.2.2	9
4.2.2.3	9
4.2.2.4 rsp	9
4.2.2.5 status	9
4.2.2.6 temp	9
4.3 LedStatus_t Struct Reference	9
4.3.1 Detailed Description	9
4.3.2 Field Documentation	10
4.3.2.1 address	10
4.3.2.2	10
4.3.2.3 blue_open	10
4.3.2.4 blue_short	10
4.3.2.5	10
4.3.2.6 green_open	10
4.3.2.7 green_short	10
4.3.2.8 ledStatus	10
4.3.2.9 ledStatus_reserved_1	11
4.3.2.10 ledStatus_reserved_2	11
4.3.2.11 red_open	11
4.3.2.12 red_short	11

4.4 osireTemp_t Struct Reference	11
4.4.1 Detailed Description	11
4.4.2 Field Documentation	11
4.4.2.1 address	11
4.4.2.2	12
4.4.2.3 temp_value	12
4.5 ospCmd_t Struct Reference	12
4.5.1 Detailed Description	12
4.5.2 Field Documentation	12
4.5.2.1 inCmdId	12
4.5.2.2 inDeviceAddress	12
4.5.2.3 outCmdBufferLength	12
4.5.2.4 outResponseLength	13
4.5.2.5 outResponseMsg	13
4.5.2.6 p_inParameter	13
4.5.2.7 p_outCmdBuffer	13
4.6 ospHeader_t Union Reference	13
4.6.1 Detailed Description	13
4.6.2 Field Documentation	13
4.6.2.1 address	13
4.6.2.2	14
4.6.2.3 buf	14
4.6.2.4 command	14
4.6.2.5 preamble	14
4.6.2.6 psi	14
4.6.2.7 reserved	14
4.7 OtpData_t Struct Reference	14
4.7.1 Detailed Description	14
4.7.2 Field Documentation	15
4.7.2.1 address	15
4.7.2.2 byte	15
4.7.2.3	15
4.8 OtpDataComplete_t Struct Reference	15
4.8.1 Detailed Description	15
4.8.2 Field Documentation	15
4.8.2.1 address	15
4.8.2.2 byte	15
4.8.2.3	16
4.9 OtthData_t Struct Reference	16
4.9.1 Detailed Description	16
4.9.2 Field Documentation	16
4.9.2.1 address	16

4.9.2.2	16
4.9.2.3	16
4.9.2.4 or_cycle	16
4.9.2.5 ot_high_value	17
4.9.2.6 ot_low_value	17
4.9.2.7 otth_reserved	17
4.9.2.8 otthData	17
4.10 PwmData_t Struct Reference	17
4.10.1 Detailed Description	17
4.10.2 Field Documentation	17
4.10.2.1 address	18
4.10.2.2	18
4.10.2.3 blue_curr	18
4.10.2.4 blue_pwm	18
4.10.2.5	18
4.10.2.6 green_curr	18
4.10.2.7 green_pwm	18
4.10.2.8 pwmData	18
4.10.2.9 red_curr	18
4.10.2.10 red_pwm	19
4.11 SetSetupData_t Struct Reference	19
4.11.1 Detailed Description	19
4.11.2 Field Documentation	19
4.11.2.1 address	19
4.11.2.2	19
4.11.2.3 ce_fsave	19
4.11.2.4 com_inv	20
4.11.2.5 crc_en	20
4.11.2.6	20
4.11.2.7 fast_pwm	20
4.11.2.8 los_fsave	20
4.11.2.9 ot_fsave	20
4.11.2.10 setupData	20
4.11.2.11 tempck_sel	20
4.11.2.12 uv_fsave	20
4.12 Status_t Struct Reference	21
4.12.1 Detailed Description	21
4.12.2 Field Documentation	21
4.12.2.1 address	21
4.12.2.2	21
4.12.2.3 ce_flag	21
4.12.2.4 com_mode	21

4.12.2.5	22
4.12.2.6 <code>los_flag</code>	22
4.12.2.7 <code>ot_flag</code>	22
4.12.2.8 <code>otpcrc_flag</code>	22
4.12.2.9 <code>state</code>	22
4.12.2.10 <code>status</code>	22
4.12.2.11 <code>uv_flag</code>	22
4.13 <code>TempStatus_t</code> Struct Reference	22
4.13.1 Detailed Description	23
4.13.2 Field Documentation	23
4.13.2.1 <code>address</code>	23
4.13.2.2	23
4.13.2.3	23
4.13.2.4 <code>Status</code>	23
4.13.2.5 <code>Temp</code>	23
4.13.2.6 <code>tempStatus</code>	23
5 File Documentation	25
5.1 <code>C:/_uc-fpu/Documentation/mainpage_osp.md</code> File Reference	25
5.2 <code>OSP/inc/genericDevice.h</code> File Reference	25
5.2.1 Macro Definition Documentation	27
5.2.1.1 <code>BROADCAST_ADDRESS</code>	27
5.2.1.2 <code>FIRST_BYTE_PAYLOAD</code>	27
5.2.1.3 <code>LENGTH_CLR_ERROR_MSG</code>	27
5.2.1.4 <code>LENGTH_GO_ACTIVE_MSG</code>	27
5.2.1.5 <code>LENGTH_GO_DEEP_SLEEP_MSG</code>	27
5.2.1.6 <code>LENGTH_GO_SLEEP_MSG</code>	27
5.2.1.7 <code>LENGTH_INIT_MSG</code>	27
5.2.1.8 <code>LENGTH_INIT_RSP</code>	27
5.2.1.9 <code>LENGTH_NO_OSP_RSP</code>	27
5.2.1.10 <code>LENGTH_RESET_MSG</code>	28
5.2.1.11 <code>MAXIMUM_ADDRESS</code>	28
5.2.1.12 <code>NO_OSP_RSP</code>	28
5.2.1.13 <code>OSP_PROTOCOL_PREAMPLE</code>	28
5.2.1.14 <code>OSP_RSP</code>	28
5.2.2 Typedef Documentation	28
5.2.2.1 <code>ospInitRsp_t</code>	28
5.2.3 Enumeration Type Documentation	28
5.2.3.1 <code>OSP_ERROR_CODE</code>	28
5.2.3.2 <code>OSP_GENERIC_DEVICE_CMDS</code>	29
5.2.4 Function Documentation	29
5.2.4.1 <code>build_header()</code>	29

5.2.4.2 osp_go_active()	31
5.2.4.3 osp_go_deep_sleep()	32
5.2.4.4 osp_go_sleep()	33
5.2.4.5 osp_init_bidir()	34
5.2.4.6 osp_init_loop()	35
5.2.4.7 osp_osire_clr_error()	36
5.2.4.8 osp_reset()	37
5.3 genericDevice.h	38
5.4 OSP/inc/osireDevice.h File Reference	40
5.4.1 Macro Definition Documentation	43
5.4.1.1 LENGTH_P4ERROR_MSG	43
5.4.1.2 LENGTH_READ_COMSTATUS_MSG	43
5.4.1.3 LENGTH_READ_COMSTATUS_RSP	43
5.4.1.4 LENGTH_READ_LEDSTATUS_MSG	43
5.4.1.5 LENGTH_READ_LEDSTATUS_RSP	43
5.4.1.6 LENGTH_READ_OTP_MSG	43
5.4.1.7 LENGTH_READ_OTP_RSP	44
5.4.1.8 LENGTH_READ_OTTH_MSG	44
5.4.1.9 LENGTH_READ_OTTH_RSP	44
5.4.1.10 LENGTH_READ_PWM_MSG	44
5.4.1.11 LENGTH_READ_PWM_RSP	44
5.4.1.12 LENGTH_READ_SETUP_MSG	44
5.4.1.13 LENGTH_READ_SETUP_RSP	44
5.4.1.14 LENGTH_READ_STATUS_MSG	44
5.4.1.15 LENGTH_READ_STATUS_RSP	44
5.4.1.16 LENGTH_READ_TEMP_MSG	44
5.4.1.17 LENGTH_READ_TEMP_RSP	45
5.4.1.18 LENGTH_READ_TEMPSTATUS_MSG	45
5.4.1.19 LENGTH_READ_TEMPSTATUS_RSP	45
5.4.1.20 LENGTH_SET_OTTH_MSG	45
5.4.1.21 LENGTH_SET_PWM_MSG	45
5.4.1.22 LENGTH_SET_SETUP_MSG	45
5.4.2 Typedef Documentation	45
5.4.2.1 osireComStatus_t	45
5.4.2.2 osireLedStatus_t	45
5.4.2.3 osireOtpData_t	45
5.4.2.4 osireOtpDataComplete_t	46
5.4.2.5 osireOtthData_t	46
5.4.2.6 osirePwmData_t	46
5.4.2.7 osireSetSetupData_t	46
5.4.2.8 osireStatus_t	46
5.4.2.9 osireTemp_t	46

5.4.2.10 osireTempStatus_t	46
5.4.3 Enumeration Type Documentation	46
5.4.3.1 OSP_OSIRE_DEVICE_CMDS	46
5.4.4 Function Documentation	47
5.4.4.1 osp_go_active_and_sr()	47
5.4.4.2 osp_osire_clr_error_and_sr()	48
5.4.4.3 osp_osire_go_deep_sleep_and_sr()	49
5.4.4.4 osp_osire_go_sleep_and_sr()	50
5.4.4.5 osp_osire_p4error_bidir()	51
5.4.4.6 osp_osire_p4error_loop()	53
5.4.4.7 osp_osire_read_comstatus()	54
5.4.4.8 osp_osire_read_ledstatus()	55
5.4.4.9 osp_osire_read_otp_complete()	55
5.4.4.10 osp_osire_read_otth()	56
5.4.4.11 osp_osire_read_pwm()	57
5.4.4.12 osp_osire_read_setup()	58
5.4.4.13 osp_osire_read_status()	59
5.4.4.14 osp_osire_read_temp()	60
5.4.4.15 osp_osire_read_tempstatus()	61
5.4.4.16 osp_osire_set_otth()	62
5.4.4.17 osp_osire_set_otth_and_sr()	63
5.4.4.18 osp_osire_set_pwm()	64
5.4.4.19 osp_osire_set_pwm_and_sr()	65
5.4.4.20 osp_osire_set_setup()	66
5.4.4.21 osp_osire_set_setup_and_sr()	67
5.4.4.22 osp_read_otp()	68
5.5 osireDevice.h	69
5.6 OSP/inc/ospCmdBuffer.h File Reference	73
5.6.1 Typedef Documentation	74
5.6.1.1 ospCmdBuffer_t	74
5.7 ospCmdBuffer.h	74
5.8 OSP/src/genericDevice.c File Reference	75
5.8.1 Function Documentation	76
5.8.1.1 build_header()	76
5.8.1.2 osp_go_active()	78
5.8.1.3 osp_go_deep_sleep()	78
5.8.1.4 osp_go_sleep()	79
5.8.1.5 osp_init_bidir()	80
5.8.1.6 osp_init_loop()	81
5.8.1.7 osp_osire_clr_error()	82
5.8.1.8 osp_reset()	83
5.9 genericDevice.c	84

5.10 OSP/src/osireDevice.c File Reference	87
5.10.1 Function Documentation	89
5.10.1.1 osp_go_active_and_sr()	89
5.10.1.2 osp_osire_clr_error_and_sr()	90
5.10.1.3 osp_osire_go_deep_sleep_and_sr()	91
5.10.1.4 osp_osire_go_sleep_and_sr()	92
5.10.1.5 osp_osire_p4error_bidir()	94
5.10.1.6 osp_osire_p4error_loop()	95
5.10.1.7 osp_osire_read_comstatus()	96
5.10.1.8 osp_osire_read_ledstatus()	97
5.10.1.9 osp_osire_read_otp_complete()	98
5.10.1.10 osp_osire_read_otth()	99
5.10.1.11 osp_osire_read_pwm()	100
5.10.1.12 osp_osire_read_setup()	101
5.10.1.13 osp_osire_read_status()	101
5.10.1.14 osp_osire_read_temp()	102
5.10.1.15 osp_osire_read_tempstatus()	103
5.10.1.16 osp_osire_set_otth()	104
5.10.1.17 osp_osire_set_otth_and_sr()	105
5.10.1.18 osp_osire_set_pwm()	106
5.10.1.19 osp_osire_set_pwm_and_sr()	107
5.10.1.20 osp_osire_set_setup()	108
5.10.1.21 osp_osire_set_setup_and_sr()	109
5.10.1.22 osp_read_otp()	110
5.11 osireDevice.c	111
5.12 OSP/src/ospCmdBuffer.c File Reference	122
5.12.1 Macro Definition Documentation	123
5.12.1.1 MAX_CMD_BUFFER_SIZE	123
5.12.2 Function Documentation	123
5.12.2.1 osp_cmd_buffer()	123
5.13 ospCmdBuffer.c	134
6 Disclaimer	145
Index	146

Chapter 1

OSIRE® E3731i OSP

- OSIRE® E3731i EVK Firmware Version 2.1.0
- OSP - Open Standard Protocol Implementation V1.0.0
- S32 Design Studio for Arm 3.5 Project
- Generic OSP commands
 - osp_reset
 - osp_init_bidir
 - osp_go_active
 - osp_clr_error
 - osp_init_loop
 - osp_go_sleep
 - osp_go_deep_sleep
- RGBi OSP commands
 - osp_osire_set_setup
 - osp_osire_read_setup
 - osp_osire_read_tempstatus
 - osp_osire_set_pwm
 - osp_osire_read_pwm
 - osp_osire_read_otp
 - osp_osire_p4error_bidir
 - osp_osire_clr_error_sr
 - osp_osire_go_sleep_sr
 - osp_osire_go_active_sr
 - osp_osire_go_deep_sleep_sr
 - osp_osire_read_status
 - osp_osire_read_comstatus
 - osp_osire_read_ledstatus
 - osp_osire_read_temp
 - osp_osire_read_otth
 - osp_osire_set_otth
 - osp_osire_set_otth_sr
 - osp_osire_set_setup_sr
 - osp_osire_set_pwm_sr
 - osp_osire_p4error_loop

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

ComStatus_t	7
InitRsp_t	8
LedStatus_t	9
osireTemp_t	11
ospCmd_t	12
ospHeader_t	13
OtpData_t	14
OtpDataComplete_t	15
OtthData_t	16
PwmData_t	17
SetSetupData_t	19
Status_t	21
TempStatus_t	22

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

OSP/inc/ genericDevice.h	25
OSP/inc/ osireDevice.h	40
OSP/inc/ ospCmdBuffer.h	73
OSP/src/ genericDevice.c	75
OSP/src/ osireDevice.c	87
OSP/src/ ospCmdBuffer.c	122

Chapter 4

Data Structure Documentation

4.1 ComStatus_t Struct Reference

```
#include <osireDevice.h>
```

Data Fields

- union {
 - uint8_t [comStatus](#)
 - struct {
 - uint8_t [sio1_state](#):2
 - uint8_t [sio2_state](#):2
 - uint8_t [reserved](#):4
 - } [bit](#)
- uint16_t [address](#):10

4.1.1 Detailed Description

Definition at line [210](#) of file [osireDevice.h](#).

4.1.2 Field Documentation

4.1.2.1 address

```
uint16_t address
```

device address

Definition at line [222](#) of file [osireDevice.h](#).

4.1.2.2

```
struct { ... } bit
```

4.1.2.3 comStatus

```
uint8_t comStatus
```

COM STATUS register

Definition at line [214](#) of file [osireDevice.h](#).

4.1.2.4

```
union { ... } data
```

4.1.2.5 reserved

```
uint8_t reserved
```

reserved

Definition at line 219 of file [osireDevice.h](#).

4.1.2.6 sio1_state

```
uint8_t sio1_state
```

Communication mode of SIO1

Definition at line 217 of file [osireDevice.h](#).

4.1.2.7 sio2_state

```
uint8_t sio2_state
```

Communication mode of SIO2

Definition at line 218 of file [osireDevice.h](#).

The documentation for this struct was generated from the following file:

- [OSP/inc/osireDevice.h](#)

4.2 InitRsp_t Struct Reference

```
#include <genericDevice.h>
```

Data Fields

- union {
 - uint8_t [rsp](#) [10]
 - struct {
 - uint8_t [temp](#)
 - uint8_t [status](#)
 - uint16_t [address](#):10
 - } [bit](#)
- } [data](#)

4.2.1 Detailed Description

Enumeration OSP Generic Device error codes

Definition at line 109 of file [genericDevice.h](#).

4.2.2 Field Documentation

4.2.2.1 address

```
uint16_t address
```

device address

Definition at line 118 of file [genericDevice.h](#).

4.2.2.2

```
struct { ... } bit
```

4.2.2.3

```
union { ... } data
```

4.2.2.4 rsp

```
uint8_t rsp[10]
```

response buffer for OSP_INIT_BIDIR command

Definition at line 113 of file [genericDevice.h](#).

4.2.2.5 status

```
uint8_t status
```

current status

Definition at line 117 of file [genericDevice.h](#).

4.2.2.6 temp

```
uint8_t temp
```

temperature

Definition at line 116 of file [genericDevice.h](#).

The documentation for this struct was generated from the following file:

- [OSP/inc/genericDevice.h](#)

4.3 LedStatus_t Struct Reference

```
#include <osireDevice.h>
```

Data Fields

- union {
 - uint8_t [ledStatus](#)
 - struct {
 - uint8_t [blue_short](#):1
 - uint8_t [green_short](#):1
 - uint8_t [red_short](#):1
 - uint8_t [ledStatus_reserved_1](#):1
 - uint8_t [blue_open](#):1
 - uint8_t [green_open](#):1
 - uint8_t [red_open](#):1
 - uint8_t [ledStatus_reserved_2](#):1
 - } [bit](#)
 - } [data](#)
- uint16_t [address](#):10

4.3.1 Detailed Description

Definition at line 177 of file [osireDevice.h](#).

4.3.2 Field Documentation

4.3.2.1 address

uint16_t address

device address

Definition at line 194 of file [osireDevice.h](#).

4.3.2.2

struct { ... } bit

4.3.2.3 blue_open

uint8_t blue_open

BLUE channel open fault flag.

Definition at line 188 of file [osireDevice.h](#).

4.3.2.4 blue_short

uint8_t blue_short

BLUE channel short fault flag

Definition at line 184 of file [osireDevice.h](#).

4.3.2.5

union { ... } data

4.3.2.6 green_open

uint8_t green_open

GREEN channel open fault flag.

Definition at line 189 of file [osireDevice.h](#).

4.3.2.7 green_short

uint8_t green_short

GREEN channel short fault flag

Definition at line 185 of file [osireDevice.h](#).

4.3.2.8 ledStatus

uint8_t ledStatus

LED STATUS register. Cleared after readout.

Definition at line 181 of file [osireDevice.h](#).

4.3.2.9 ledStatus_reserved_1

uint8_t ledStatus_reserved_1
reserved
Definition at line 187 of file [osireDevice.h](#).

4.3.2.10 ledStatus_reserved_2

uint8_t ledStatus_reserved_2
reserved
Definition at line 191 of file [osireDevice.h](#).

4.3.2.11 red_open

uint8_t red_open
RED channel open fault flag.
Definition at line 190 of file [osireDevice.h](#).

4.3.2.12 red_short

uint8_t red_short
RED channel short fault flag
Definition at line 186 of file [osireDevice.h](#).
The documentation for this struct was generated from the following file:

- [OSP/inc/osireDevice.h](#)

4.4 osireTemp_t Struct Reference

```
#include <osireDevice.h>
```

Data Fields

- union {
 uint8_t temp_value
} data
- uint16_t address:10

4.4.1 Detailed Description

Definition at line 199 of file [osireDevice.h](#).

4.4.2 Field Documentation

4.4.2.1 address

uint16_t address
device address
Definition at line 205 of file [osireDevice.h](#).

4.4.2.2

```
union { ... } data
```

4.4.2.3 temp_value

```
uint8_t temp_value
```

TEMP register

Definition at line 203 of file [osireDevice.h](#).

The documentation for this struct was generated from the following file:

- [OSP/inc/osireDevice.h](#)

4.5 ospCmd_t Struct Reference

```
#include <ospCmdBuffer.h>
```

Data Fields

- `uint16_t` [inDeviceAddress](#)
- `uint8_t` [inCmdId](#)
- `void *` [p_inParameter](#)
- `uint8_t *` [p_outCmdBuffer](#)
- `uint8_t` [outCmdBufferLength](#)
- `uint8_t` [outResponseLength](#)
- `bool` [outResponseMsg](#)

4.5.1 Detailed Description

Definition at line 33 of file [ospCmdBuffer.h](#).

4.5.2 Field Documentation

4.5.2.1 inCmdId

```
uint8_t inCmdId
```

INPUT: OSP command identifier

Definition at line 36 of file [ospCmdBuffer.h](#).

4.5.2.2 inDeviceAddress

```
uint16_t inDeviceAddress
```

INPUT: device address

Definition at line 35 of file [ospCmdBuffer.h](#).

4.5.2.3 outCmdBufferLength

```
uint8_t outCmdBufferLength
```

OUTPUT: length of requested OSP sequence

Definition at line 39 of file [ospCmdBuffer.h](#).

4.5.2.4 outResponseLength

uint8_t outResponseLength

OUTPUT: length of the expected response

Definition at line 40 of file [ospCmdBuffer.h](#).

4.5.2.5 outResponseMsg

bool outResponseMsg

OUTPUT: true if a response id expected

Definition at line 41 of file [ospCmdBuffer.h](#).

4.5.2.6 p_inParameter

void* p_inParameter

INPUT: pointer to parameter structure

Definition at line 37 of file [ospCmdBuffer.h](#).

4.5.2.7 p_outCmdBuffer

uint8_t* p_outCmdBuffer

OUTPUT: buffer with requested OSP sequence

Definition at line 38 of file [ospCmdBuffer.h](#).

The documentation for this struct was generated from the following file:

- [OSP/inc/ospCmdBuffer.h](#)

4.6 ospHeader_t Union Reference

```
#include <genericDevice.h>
```

Data Fields

- uint8_t [buf](#) [4]
 - struct {
 - uint32_t [reserved](#):8
 - uint32_t [command](#):7
 - uint32_t [psi](#):3
 - uint32_t [address](#):10
 - uint32_t [preamble](#):4
- } [bit](#)

4.6.1 Detailed Description

Definition at line 126 of file [genericDevice.h](#).

4.6.2 Field Documentation

4.6.2.1 address

uint32_t address

device address

Definition at line 134 of file [genericDevice.h](#).

4.6.2.2

```
struct { ... } bit
```

4.6.2.3 buf

```
uint8_t buf[4]
```

header buffer

Definition at line 128 of file [genericDevice.h](#).

4.6.2.4 command

```
uint32_t command
```

OSP or OSP_OSIRE command

Definition at line 132 of file [genericDevice.h](#).

4.6.2.5 preamble

```
uint32_t preamble
```

OSP_PROTOCOL_PREAMBLE 0x0A

Definition at line 135 of file [genericDevice.h](#).

4.6.2.6 psi

```
uint32_t psi
```

payload in bytes

Definition at line 133 of file [genericDevice.h](#).

4.6.2.7 reserved

```
uint32_t reserved
```

reserved

Definition at line 131 of file [genericDevice.h](#).

The documentation for this union was generated from the following file:

- [OSP/inc/genericDevice.h](#)

4.7 OtpData_t Struct Reference

```
#include <osireDevice.h>
```

Data Fields

- union {
 - uint8_t [byte](#) [8]
 } [data](#)
- uint16_t [address](#):10

4.7.1 Detailed Description

Definition at line 155 of file [osireDevice.h](#).

4.7.2 Field Documentation

4.7.2.1 address

uint16_t address

device address

Definition at line 161 of file [osireDevice.h](#).

4.7.2.2 byte

uint8_t byte[8]

buffer for one otp read block

Definition at line 159 of file [osireDevice.h](#).

4.7.2.3

union { ... } data

The documentation for this struct was generated from the following file:

- [OSP/inc/osireDevice.h](#)

4.8 OtpDataComplete_t Struct Reference

```
#include <osireDevice.h>
```

Data Fields

- union {
 uint8_t [byte](#) [32]
} [data](#)
- uint16_t [address](#):10

4.8.1 Detailed Description

Definition at line 166 of file [osireDevice.h](#).

4.8.2 Field Documentation

4.8.2.1 address

uint16_t address

device address

Definition at line 172 of file [osireDevice.h](#).

4.8.2.2 byte

uint8_t byte[32]

max. buffer for full otp read memory

Definition at line 170 of file [osireDevice.h](#).

4.8.2.3

```
union { ... } data
```

The documentation for this struct was generated from the following file:

- [OSP/inc/osireDevice.h](#)

4.9 OtthData_t Struct Reference

```
#include <osireDevice.h>
```

Data Fields

- union {
 - uint8_t [otthData](#) [3]
 - struct {
 - uint8_t [ot_high_value](#):8
 - uint8_t [ot_low_value](#):8
 - uint8_t [or_cycle](#):2
 - uint8_t [otth_reserved](#):6
 - } [bit](#)
- uint16_t [address](#):10

4.9.1 Detailed Description

Definition at line [263](#) of file [osireDevice.h](#).

4.9.2 Field Documentation

4.9.2.1 address

```
uint16_t address
```

device address

Definition at line [276](#) of file [osireDevice.h](#).

4.9.2.2

```
struct { ... } bit
```

4.9.2.3

```
union { ... } data
```

4.9.2.4 or_cycle

```
uint8_t or_cycle
```

overttemperature detection low pass filter cycle length

Definition at line [272](#) of file [osireDevice.h](#).

4.9.2.5 ot_high_value

uint8_t ot_high_value
 overtemperature fault threshold
 Definition at line 270 of file [osireDevice.h](#).

4.9.2.6 ot_low_value

uint8_t ot_low_value
 overtemperature fault release threshold
 Definition at line 271 of file [osireDevice.h](#).

4.9.2.7 otth_reserved

uint8_t otth_reserved
 reserved
 Definition at line 273 of file [osireDevice.h](#).

4.9.2.8 otthData

uint8_t otthData[3]
 STATUS register
 Definition at line 267 of file [osireDevice.h](#).
 The documentation for this struct was generated from the following file:

- [OSP/inc/osireDevice.h](#)

4.10 PwmData_t Struct Reference

```
#include <osireDevice.h>
```

Data Fields

- union {
 - uint8_t [pwmData](#) [6]
 - struct {
 - uint16_t [blue_pwm](#):15
 - uint16_t [blue_curr](#):1
 - uint16_t [green_pwm](#):15
 - uint16_t [green_curr](#):1
 - uint16_t [red_pwm](#):15
 - uint16_t [red_curr](#):1
 - } [bit](#)
 - } [data](#)
- uint16_t [address](#):10

4.10.1 Detailed Description

Definition at line 135 of file [osireDevice.h](#).

4.10.2 Field Documentation

4.10.2.1 address

uint16_t address

device address

Definition at line 150 of file [osireDevice.h](#).

4.10.2.2

```
struct { ... } bit
```

4.10.2.3 blue_curr

uint16_t blue_curr

day or night mode

Definition at line 143 of file [osireDevice.h](#).

4.10.2.4 blue_pwm

uint16_t blue_pwm

blue PWM value

Definition at line 142 of file [osireDevice.h](#).

4.10.2.5

```
union { ... } data
```

4.10.2.6 green_curr

uint16_t green_curr

day or night mode

Definition at line 145 of file [osireDevice.h](#).

4.10.2.7 green_pwm

uint16_t green_pwm

green PWM value

Definition at line 144 of file [osireDevice.h](#).

4.10.2.8 pwmData

uint8_t pwmData[6]

PWM data buffer

Definition at line 139 of file [osireDevice.h](#).

4.10.2.9 red_curr

uint16_t red_curr

day or night mode

Definition at line 147 of file [osireDevice.h](#).

4.10.2.10 red_pwm

uint16_t red_pwm

red PWM value

Definition at line 146 of file [osireDevice.h](#).

The documentation for this struct was generated from the following file:

- [OSP/inc/osireDevice.h](#)

4.11 SetSetupData_t Struct Reference

```
#include <osireDevice.h>
```

Data Fields

- union {
 - uint8_t [setupData](#)
 - struct {
 - uint8_t [uv_fsave](#):1
 - uint8_t [ot_fsave](#):1
 - uint8_t [los_fsave](#):1
 - uint8_t [ce_fsave](#):1
 - uint8_t [tempck_sel](#):1
 - uint8_t [crc_en](#):1
 - uint8_t [com_inv](#):1
 - uint8_t [fast_pwm](#):1
 - } [bit](#)
 - } [data](#)
- uint16_t [address](#):10

4.11.1 Detailed Description

Definition at line 113 of file [osireDevice.h](#).

4.11.2 Field Documentation

4.11.2.1 address

uint16_t address

device address

Definition at line 130 of file [osireDevice.h](#).

4.11.2.2

```
struct { ... } bit
```

4.11.2.3 ce_fsave

uint8_t ce_fsave

communication error is detected

Definition at line 123 of file [osireDevice.h](#).

4.11.2.4 com_inv

uint8_t com_inv

CLK polarity for MCU mode

Definition at line 126 of file [osireDevice.h](#).

4.11.2.5 crc_en

uint8_t crc_en

CRC check

Definition at line 125 of file [osireDevice.h](#).

4.11.2.6

union { ... } data

4.11.2.7 fast_pwm

uint8_t fast_pwm

PWM frequency and dynamic range

Definition at line 127 of file [osireDevice.h](#).

4.11.2.8 los_fsave

uint8_t los_fsave

open/short error is detected

Definition at line 122 of file [osireDevice.h](#).

4.11.2.9 ot_fsave

uint8_t ot_fsave

overtemperature error is detected

Definition at line 121 of file [osireDevice.h](#).

4.11.2.10 setupData

uint8_t setupData

SETUP DATA register

Definition at line 117 of file [osireDevice.h](#).

4.11.2.11 tempck_sel

uint8_t tempck_sel

Update rate of the temperature sensor

Definition at line 124 of file [osireDevice.h](#).

4.11.2.12 uv_fsave

uint8_t uv_fsave

undervoltage error is detected

Definition at line 120 of file [osireDevice.h](#).

The documentation for this struct was generated from the following file:

- [OSP/inc/osireDevice.h](#)

4.12 Status_t Struct Reference

```
#include <osireDevice.h>
```

Data Fields

- union {
 - uint8_t [status](#)
 - struct {
 - uint8_t [uv_flag](#):1
 - uint8_t [ot_flag](#):1
 - uint8_t [los_flag](#):1
 - uint8_t [ce_flag](#):1
 - uint8_t [com_mode](#):1
 - uint8_t [otpcrc_flag](#):1
 - uint8_t [state](#):2
 - } [bit](#)
 - } [data](#)
- uint16_t [address](#):10

4.12.1 Detailed Description

Definition at line [227](#) of file [osireDevice.h](#).

4.12.2 Field Documentation

4.12.2.1 address

uint16_t address
device address

Definition at line [243](#) of file [osireDevice.h](#).

4.12.2.2

```
struct { ... } bit
```

4.12.2.3 ce_flag

uint8_t ce_flag
communication fault flag

Definition at line [237](#) of file [osireDevice.h](#).

4.12.2.4 com_mode

uint8_t com_mode
communication direction

Definition at line [238](#) of file [osireDevice.h](#).

4.12.2.5

```
union { ... } data
```

4.12.2.6 los_flag

```
uint8_t los_flag
```

LED fault flag

Definition at line 236 of file [osireDevice.h](#).

4.12.2.7 ot_flag

```
uint8_t ot_flag
```

overtemperature fault flag

Definition at line 235 of file [osireDevice.h](#).

4.12.2.8 otpcrc_flag

```
uint8_t otpcrc_flag
```

OTP error flag

Definition at line 239 of file [osireDevice.h](#).

4.12.2.9 state

```
uint8_t state
```

device state

Definition at line 240 of file [osireDevice.h](#).

4.12.2.10 status

```
uint8_t status
```

STATUS register

Definition at line 231 of file [osireDevice.h](#).

4.12.2.11 uv_flag

```
uint8_t uv_flag
```

undervoltage fault flag

Definition at line 234 of file [osireDevice.h](#).

The documentation for this struct was generated from the following file:

- [OSP/inc/osireDevice.h](#)

4.13 TempStatus_t Struct Reference

```
#include <osireDevice.h>
```

Data Fields

- union {
 - uint8_t [tempStatus](#) [2]
 - struct {
 - uint8_t [Status](#)


```
    uint8_t Temp
} byte
} data
```

- uint16_t address:10

4.13.1 Detailed Description

Definition at line 248 of file [osireDevice.h](#).

4.13.2 Field Documentation

4.13.2.1 address

```
uint16_t address
```

device address

Definition at line 259 of file [osireDevice.h](#).

4.13.2.2

```
struct { ... } byte
```

4.13.2.3

```
union { ... } data
```

4.13.2.4 Status

```
uint8_t Status
```

Definition at line 255 of file [osireDevice.h](#).

4.13.2.5 Temp

```
uint8_t Temp
```

Definition at line 256 of file [osireDevice.h](#).

4.13.2.6 tempStatus

```
uint8_t tempStatus[2]
```

STATUS + TEMP registers

Definition at line 252 of file [osireDevice.h](#).

The documentation for this struct was generated from the following file:

- [OSP/inc/osireDevice.h](#)

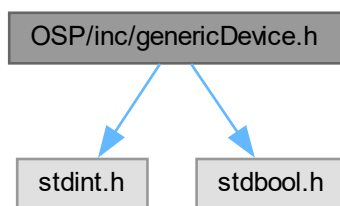
Chapter 5

File Documentation

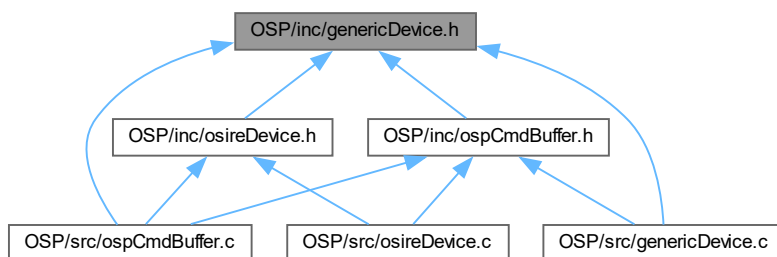
5.1 C:/_uc-fpu/Documentation/mainpage_osp.md File Reference

5.2 OSP/inc/genericDevice.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
Include dependency graph for genericDevice.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [InitRsp_t](#)
- union [ospHeader_t](#)

Macros

- `#define OSP_PROTOCOL_PREAMBLE 0x0A`
- `#define FIRST_BYTE_PAYLOAD 3`
- `#define BROADCAST_ADDRESS 0`
- `#define MAXIMUM_ADDRESS 1002`
- `#define LENGTH_INIT_MSG 4`
- `#define LENGTH_INIT_RSP 6`
- `#define LENGTH_RESET_MSG 4`
- `#define LENGTH_GO_ACTIVE_MSG 4`
- `#define LENGTH_GO_SLEEP_MSG 4`
- `#define LENGTH_GO_DEEP_SLEEP_MSG 4`
- `#define LENGTH_CLR_ERROR_MSG 4`
- `#define NO_OSP_RSP false`
- `#define OSP_RSP !NO_OSP_RSP`
- `#define LENGTH_NO_OSP_RSP 0`

Typedefs

- `typedef struct InitRsp_t ospInitRsp_t`

Enumerations

- `enum OSP_GENERIC_DEVICE_CMDS {
 OSP_RESET = 0x00 , OSP_CLR_ERROR = 0x01 , OSP_INIT_BIDIR = 0x02 , OSP_INIT_LOOP = 0x03 ,
 OSP_GO_SLEEP = 0x04 , OSP_GO_ACTIVE = 0x05 , OSP_GO_DEEP_SLEEP = 0x06 }`
- `enum OSP_ERROR_CODE {
 OSP_NO_ERROR = 0x00 , OSP_ADDRESS_ERROR , OSP_ERROR_INITIALIZATION , OSP_ERROR_CRC
 ,
 OSP_ERROR_SPI , OSP_ERROR_PARAMETER , OSP_ERROR_NOT_IMPLEMENTED }`

Functions

- `enum OSP_ERROR_CODE osp_init_bidir (uint16_t deviceAddress, ospInitRsp_t *p_rsp)`
OSP_INIT_BIDIR command Initiates the automatic addressing of the chain and sets the communication direction to bidirectional. The command shall be addressed to the first unit in the chain always with address 0x001. The last unit in the chain (indicated by the EOL mode) returns its address to the master.
- `enum OSP_ERROR_CODE osp_reset (uint16_t deviceAddress)`
OSP_RESET command Performs a complete reset of one or all devices. The effect is identical to a power cycle. All register values are set to their default values, all error flags are cleared, the communication mode detection is restarted, LED drivers are turned off, and the address is set to 0x3ff. The device enters the UNINITIALIZED mode.
- `enum OSP_ERROR_CODE osp_go_active (uint16_t deviceAddress)`
OSP_GO_ACTIVE command. Puts device into ACTIVE state: The LED drivers are enabled. The last PWM parameters are used to create LED PWM signals. An PWM parameter update via SETPWM() command is possible. During ACTIVE mode, diagnostic is possible and readable.
- `enum OSP_ERROR_CODE osp_init_loop (uint16_t deviceAddress, ospInitRsp_t *p_rsp)`
OSP_INIT_LOOP command Same as INITBIDIR but sets the communication mode to loop-back. The response to the master is sent in the forward direction.
- `enum OSP_ERROR_CODE osp_go_sleep (uint16_t deviceAddress)`
OSP_GO_SLEEP command Sends one or all devices into SLEEP state.
- `enum OSP_ERROR_CODE osp_go_deep_sleep (uint16_t deviceAddress)`
OSP_GO_DEEP_SLEEP command Sends one or all devices into DEEPSLEEP state.
- `enum OSP_ERROR_CODE osp_osire_clr_error (uint16_t deviceAddress)`
OSP_CLEAR_ERROR command This function will clear all error flags, if an error still exists for example short/open the error flag is set again.
- `void build_header (uint8_t *p_msg, uint16_t deviceAddress, uint8_t command, uint8_t lengthMsg)`
Internal function for OSP header creation.

5.2.1 Macro Definition Documentation

5.2.1.1 BROADCAST_ADDRESS

```
#define BROADCAST_ADDRESS 0
```

Definition at line 51 of file [genericDevice.h](#).

5.2.1.2 FIRST_BYTE_PAYLOAD

```
#define FIRST_BYTE_PAYLOAD 3
```

Definition at line 47 of file [genericDevice.h](#).

5.2.1.3 LENGTH_CLR_ERROR_MSG

```
#define LENGTH_CLR_ERROR_MSG 4
```

Definition at line 67 of file [genericDevice.h](#).

5.2.1.4 LENGTH_GO_ACTIVE_MSG

```
#define LENGTH_GO_ACTIVE_MSG 4
```

Definition at line 61 of file [genericDevice.h](#).

5.2.1.5 LENGTH_GO_DEEP_SLEEP_MSG

```
#define LENGTH_GO_DEEP_SLEEP_MSG 4
```

Definition at line 65 of file [genericDevice.h](#).

5.2.1.6 LENGTH_GO_SLEEP_MSG

```
#define LENGTH_GO_SLEEP_MSG 4
```

Definition at line 63 of file [genericDevice.h](#).

5.2.1.7 LENGTH_INIT_MSG

```
#define LENGTH_INIT_MSG 4
```

Definition at line 56 of file [genericDevice.h](#).

5.2.1.8 LENGTH_INIT_RSP

```
#define LENGTH_INIT_RSP 6
```

Definition at line 57 of file [genericDevice.h](#).

5.2.1.9 LENGTH_NO_OSP_RSP

```
#define LENGTH_NO_OSP_RSP 0
```

Definition at line 71 of file [genericDevice.h](#).

5.2.1.10 LENGTH_RESET_MSG

```
#define LENGTH_RESET_MSG 4
```

Definition at line 59 of file [genericDevice.h](#).

5.2.1.11 MAXIMUM_ADDRESS

```
#define MAXIMUM_ADDRESS 1002
```

Definition at line 52 of file [genericDevice.h](#).

5.2.1.12 NO_OSP_RSP

```
#define NO_OSP_RSP false
```

Definition at line 69 of file [genericDevice.h](#).

5.2.1.13 OSP_PROTOCOL_PREAMPLE

```
#define OSP_PROTOCOL_PREAMPLE 0x0A
```

Brief OSP Library - Generic Device

Customer API for devices that are communicating with OSP Generic Device Protocol

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf" Defines OSP Generic Device commands

Definition at line 46 of file [genericDevice.h](#).

5.2.1.14 OSP_RSP

```
#define OSP_RSP !NO_OSP_RSP
```

Definition at line 70 of file [genericDevice.h](#).

5.2.2 Typedef Documentation

5.2.2.1 osplnitRsp_t

```
typedef struct InitRsp_t ospInitRsp_t
```

Enumeration OSP Generic Device error codes

5.2.3 Enumeration Type Documentation

5.2.3.1 OSP_ERROR_CODE

```
enum OSP_ERROR_CODE
```

Enumeration OSP Generic Device error codes

Enumerator

OSP_NO_ERROR	no error
OSP_ADDRESS_ERROR	invalid device address
OSP_ERROR_INITIALIZATION	error while initializing
OSP_ERROR_CRC	incorrect CRC of OSP command
OSP_ERROR_SPI	SPI interface error
OSP_ERROR_PARAMETER	invalid parameter error
OSP_ERROR_NOT_IMPLEMENTED	CMD not implemented error

Definition at line 92 of file [genericDevice.h](#).

```
00093 {
00094     OSP_NO_ERROR = 0x00,
00095     OSP_ADDRESS_ERROR,
00096     OSP_ERROR_INITIALIZATION,
00097     OSP_ERROR_CRC,
00099     OSP_ERROR_SPI,
00100     OSP_ERROR_PARAMETER,
00101     OSP_ERROR_NOT_IMPLEMENTED
00102 };
```

5.2.3.2 OSP_GENERIC_DEVICE_CMDS

enum [OSP_GENERIC_DEVICE_CMDS](#)

Enumeration OSP Generic Device commands

Enumerator

OSP_RESET	implemented
OSP_CLR_ERROR	implemented
OSP_INIT_BIDIR	implemented
OSP_INIT_LOOP	implemented
OSP_GO_SLEEP	implemented
OSP_GO_ACTIVE	implemented
OSP_GO_DEEP_SLEEP	implemented

Definition at line 76 of file [genericDevice.h](#).

```
00077 {
00078     OSP_RESET = 0x00,
00079     OSP_CLR_ERROR = 0x01,
00080     OSP_INIT_BIDIR = 0x02,
00081     OSP_INIT_LOOP = 0x03,
00082     OSP_GO_SLEEP = 0x04,
00083     OSP_GO_ACTIVE = 0x05,
00084     OSP_GO_DEEP_SLEEP = 0x06,
00085 };
```

5.2.4 Function Documentation

5.2.4.1 build_header()

```
void build_header (
    uint8_t * p_msg,
    uint16_t deviceAddress,
    uint8_t command,
    uint8_t lengthMsg )
```

Internal function for OSP header creation.

Parameters

<i>p_msg</i>	message buffer to create the OSP header
<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
<i>command</i>	OSP or OSP_OSIRE command
<i>lengthMsg</i>	length of the buffer that is used lengthMsg

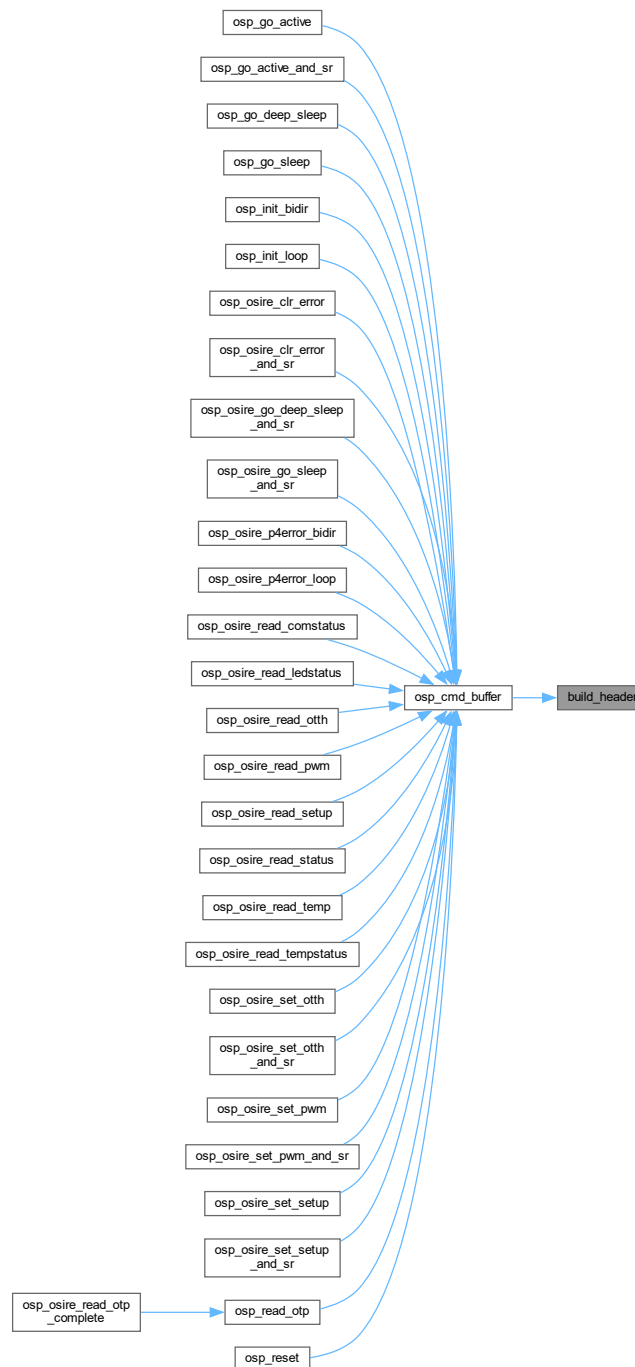
Returns

error communication or command parameter error

Definition at line 269 of file [genericDevice.c](#).

```
00271 {
00272     ospHeader_t hdr;
00273
00274     hdr.bit.preamble = OSP_PROTOCOL_PREAMPLE;
00275     hdr.bit.address = deviceAddress;
00276
00277     if (lengthMsg == 12)
00278     {
00279         hdr.bit.psi = 7;
00280     }
00281     else
00282     {
00283         hdr.bit.psi = lengthMsg - 4;
00284     }
00285     hdr.bit.command = command;
00286
00287     for (uint8_t i = 0; i < 3; i++)
00288     {
00289         *p_msg = hdr.buf[3 - i];
00290         p_msg++;
00291     }
00292 }
```


Here is the caller graph for this function:



5.2.4.2 osp_go_active()

```
enum OSP_ERROR_CODE osp_go_active (
    uint16_t deviceAddress )
```

OSP_GO_ACTIVE command. Puts device into ACTIVE state: The LED drivers are enabled. The last PWM parameters are used to create LED PWM signals. An PWM parameter update via SETPWM() command is possible. During ACTIVE mode, diagnostic is possible and readable.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
----------------------	--

Returns

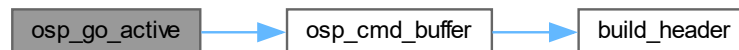
error communication or command parameter error

Definition at line 150 of file [genericDevice.c](#).

```

00151 {
00152     ospCmdBuffer_t ospCmd;
00153     enum OSP_ERROR_CODE ospErrorCode;
00154     errorSpi_t spiError;
00155
00156     ospCmd.inCmdId = OSP_GO_ACTIVE;
00157     ospCmd.inDeviceAddress = deviceAddress;
00158     ospCmd.p_inParameter = NULL;
00159
00160     ospErrorCode = osp_cmd_buffer (&ospCmd);
00161     if (ospErrorCode != OSP_NO_ERROR)
00162     {
00163         return ospErrorCode;
00164     }
00165
00166     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00167                                           ospCmd.outCmdBufferLength);
00168
00169     if (spiError != NO_ERROR_SPI)
00170     {
00171         return OSP_ERROR_SPI;
00172     }
00173
00174     return OSP_NO_ERROR;
00175 }
```

Here is the call graph for this function:



5.2.4.3 osp_go_deep_sleep()

```

enum OSP_ERROR_CODE osp_go_deep_sleep (
    uint16_t deviceAddress )
```

OSP_GO_DEEP_SLEEP command Sends one or all devices into DEEPSLEEP state.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
----------------------	--

Returns

error communication or command parameter error

Definition at line 209 of file [genericDevice.c](#).

```

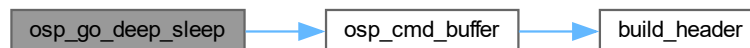
00210 {
00211     ospCmdBuffer_t ospCmd;
```

```

00212  enum OSP_ERROR_CODE ospErrorCode;
00213  errorSpi_t spiError;
00214
00215  ospCmd.inCmdId = OSP_GO_DEEP_SLEEP;
00216  ospCmd.inDeviceAddress = deviceAddress;
00217  ospCmd.p_inParameter = NULL;
00218
00219  ospErrorCode = osp_cmd_buffer (&ospCmd);
00220  if (ospErrorCode != OSP_NO_ERROR)
00221  {
00222      return ospErrorCode;
00223  }
00224
00225  spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00226                                         ospCmd.outCmdBufferLength);
00227
00228  if (spiError != NO_ERROR_SPI)
00229  {
00230      return OSP_ERROR_SPI;
00231  }
00232
00233  return OSP_NO_ERROR;
00234 }
00235 }

```

Here is the call graph for this function:



5.2.4.4 osp_go_sleep()

```

enum OSP_ERROR_CODE osp_go_sleep (
    uint16_t deviceAddress )

```

OSP_GO_SLEEP command Sends one or all devices into SLEEP state.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
----------------------	--

Returns

error communication or command parameter error

Definition at line 179 of file [genericDevice.c](#).

```

00180 {
00181     ospCmdBuffer_t ospCmd;
00182     enum OSP_ERROR_CODE ospErrorCode;
00183     errorSpi_t spiError;
00184
00185     ospCmd.inCmdId = OSP_GO_SLEEP;
00186     ospCmd.inDeviceAddress = deviceAddress;
00187     ospCmd.p_inParameter = NULL;
00188
00189     ospErrorCode = osp_cmd_buffer (&ospCmd);
00190     if (ospErrorCode != OSP_NO_ERROR)
00191     {
00192         return ospErrorCode;
00193     }
00194
00195     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00196                                             ospCmd.outCmdBufferLength);
00197
00198     if (spiError != NO_ERROR_SPI)
00199     {

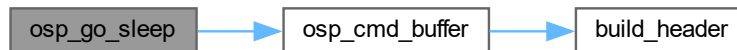
```

```

00200         return OSP_ERROR_SPI;
00201     }
00202
00203     return OSP_NO_ERROR;
00204
00205 }

```

Here is the call graph for this function:



5.2.4.5 osp_init_bidir()

```

enum OSP_ERROR_CODE osp_init_bidir (
    uint16_t deviceAddress,
    ospInitRsp_t * p_rsp )

```

OSP_INIT_BIDIR command Initiates the automatic addressing of the chain and sets the communication direction to bidirectional. The command shall be addressed to the first unit in the chain always with address 0x001. The last unit in the chain (indicated by the EOL mode) returns its address to the master.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	start address of 1st device, shall be 1
<i>p_rsp</i>	response data from device

Returns

error communication or command parameter error

Definition at line 30 of file [genericDevice.c](#).

```

00031 {
00032     uint8_t rspBuffer[LENGTH_INIT_RSP]; // response buffer
00033     ospCmdBuffer_t ospCmd;
00034     enum OSP_ERROR_CODE ospErrorCode;
00035     errorSpi_t spiError;
00036
00037     // clear response buffer
00038     memset (rspBuffer, 0, LENGTH_INIT_RSP);
00039
00040     ospCmd.inCmdId = OSP_INIT_BIDIR;
00041     ospCmd.inDeviceAddress = deviceAddress;
00042     ospCmd.p_inParameter = NULL;
00043
00044     ospErrorCode = osp_cmd_buffer (&ospCmd);
00045     if (ospErrorCode != OSP_NO_ERROR)
00046     {
00047         return ospErrorCode;
00048     }
00049
00050     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00051                                                         rspBuffer,
00052                                                         ospCmd.outCmdBufferLength,
00053                                                         ospCmd.outResponseLength);
00054
00055     if (spiError != NO_ERROR_SPI)
00056     {
00057         return OSP_ERROR_SPI;
00058     }
00059
00060     if (crc (rspBuffer, LENGTH_INIT_RSP) != 0)
00061     {

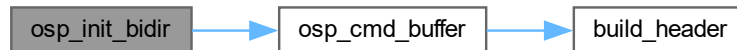
```

```

00062     return OSP_ERROR_CRC;
00063 }
00064
00065 p_rsp->data.bit.temp = rspBuffer[3];
00066 p_rsp->data.bit.status = rspBuffer[4];
00067 p_rsp->data.bit.address = ((rspBuffer[0] & 0x0F) << 6)
00068     | ((rspBuffer[1] >> 2) & 0x3F);
00069
00070 p_rsp->address = p_rsp->data.bit.address; // return address for all cmds with rsp
00071 return OSP_NO_ERROR;
00072 }

```

Here is the call graph for this function:



5.2.4.6 osp_init_loop()

```

enum OSP_ERROR_CODE osp_init_loop (
    uint16_t deviceAddress,
    ospInitRsp_t * p_rsp )

```

OSP_INIT_LOOP command Same as INITBIDIR but sets the communication mode to loop-back. The response to the master is sent in the forward direction.

For further details refer to "OSIRE_E3731i_Start_Up_Guide"

Parameters

<i>deviceAddress</i>	start address of 1st device, shall be 1
<i>p_rsp</i>	response data from device

Returns

error communication or command parameter error

Definition at line 76 of file [genericDevice.c](#).

```

00077 {
00078     uint8_t rspBuffer[LENGTH_INIT_RSP]; // response buffer
00079     ospCmdBuffer_t ospCmd;
00080     enum OSP_ERROR_CODE ospErrorCode;
00081     errorSpi_t spiError;
00082
00083     memset (rspBuffer, 0, LENGTH_INIT_RSP);
00084
00085     ospCmd.inCmdId = OSP_INIT_LOOP;
00086     ospCmd.inDeviceAddress = deviceAddress;
00087     ospCmd.p_inParameter = NULL;
00088
00089     ospErrorCode = osp_cmd_buffer (&ospCmd);
00090     if (ospErrorCode != OSP_NO_ERROR)
00091     {
00092         return ospErrorCode;
00093     }
00094
00095     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00096                                                         rspBuffer,
00097                                                         ospCmd.outCmdBufferLength,
00098                                                         ospCmd.outResponseLength);
00099
00100     if (spiError != NO_ERROR_SPI)
00101     {
00102         return OSP_ERROR_SPI;
00103     }

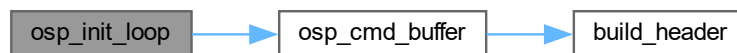
```

```

00104
00105     if (crc (rspBuffer, LENGTH_INIT_RSP) != 0)
00106     {
00107         return OSP_ERROR_CRC;
00108     }
00109
00110     p_rsp->data.bit.temp = rspBuffer[3];
00111     p_rsp->data.bit.status = rspBuffer[4];
00112     p_rsp->data.bit.address = ((rspBuffer[0] & 0x0F) << 6)
00113         | ((rspBuffer[1] >> 2) & 0x3F);
00114
00115     p_rsp->address = p_rsp->data.bit.address; // return address for all cmds with rsp
00116     return OSP_NO_ERROR;
00117 }

```

Here is the call graph for this function:



5.2.4.7 osp_osire_clr_error()

```

enum OSP_ERROR_CODE osp_osire_clr_error (
    uint16_t deviceAddress )

```

OSP_CLEAR_ERROR command This function will clear all error flags, if an error still exists for example short/open the error flag is set again.

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
----------------------	--

Returns

error communication or command parameter error

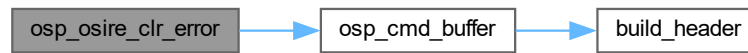
Definition at line 239 of file [genericDevice.c](#).

```

00240 {
00241     ospCmdBuffer_t ospCmd;
00242     enum OSP_ERROR_CODE ospErrorCode;
00243     errorSpi_t spiError;
00244
00245     ospCmd.inCmdId = OSP_CLR_ERROR;
00246     ospCmd.inDeviceAddress = deviceAddress;
00247     ospCmd.p_inParameter = NULL;
00248
00249     ospErrorCode = osp_cmd_buffer (&ospCmd);
00250     if (ospErrorCode != OSP_NO_ERROR)
00251     {
00252         return ospErrorCode;
00253     }
00254
00255     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00256                                           ospCmd.outCmdBufferLength);
00257
00258     if (spiError != NO_ERROR_SPI)
00259     {
00260         return OSP_ERROR_SPI;
00261     }
00262
00263     return OSP_NO_ERROR;
00264 }
00265 }

```

Here is the call graph for this function:



5.2.4.8 osp_reset()

```
enum OSP_ERROR_CODE osp_reset (
    uint16_t deviceAddress )
```

OSP_RESET command Performs a complete reset of one or all devices. The effect is identical to a power cycle. All register values are set to their default values, all error flags are cleared, the communication mode detection is restarted, LED drivers are turned off, and the address is set to 0x3ff. The device enters the UNINITIALIZED mode. For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceId</i>	0..1000 RGBi device address (0: broadcast)
-----------------	--

Returns

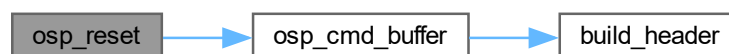
error communication or command parameter error

Definition at line 121 of file [genericDevice.c](#).

```

00122 {
00123     ospCmdBuffer_t ospCmd;
00124     enum OSP_ERROR_CODE ospErrorCode;
00125     errorSpi_t spiError;
00126
00127     ospCmd.inCmdId = OSP_RESET;
00128     ospCmd.inDeviceAddress = deviceAddress;
00129     ospCmd.p_inParameter = NULL;
00130
00131     ospErrorCode = osp_cmd_buffer (&ospCmd);
00132     if (ospErrorCode != OSP_NO_ERROR)
00133     {
00134         return ospErrorCode;
00135     }
00136
00137     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00138                                           ospCmd.outCmdBufferLength);
00139
00140     if (spiError != NO_ERROR_SPI)
00141     {
00142         return OSP_ERROR_SPI;
00143     }
00144
00145     return OSP_NO_ERROR;
00146 }
```

Here is the call graph for this function:



5.3 genericDevice.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002 * Copyright 2022 by ams OSRAM AG
00003 * All rights are reserved.
00004 *
00005 * IMPORTANT - PLEASE READ CAREFULLY BEFORE COPYING, INSTALLING OR USING
00006 * THE SOFTWARE.
00007 *
00008 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00009 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00010 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
00011 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00012 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00013 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00014 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00015 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00016 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00017 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00018 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00019 *****/
00020 #ifndef OSP_INC_GENERIC_DEVICE_H_
00021 #define OSP_INC_GENERIC_DEVICE_H_
00022
00023 #ifdef __cplusplus
00024 extern "C"
00025 {
00026 #endif
00027
00028 #include <stdint.h>
00029 #include <stdbool.h>
00030
00044 /*****/
00045 /*****/
00046 #define OSP_PROTOCOL_PREAMBLE 0x0A
00047 #define FIRST_BYTE_PAYLOAD 3
00048
00049 /*****/
00050 /*****/
00051 #define BROADCAST_ADDRESS 0
00052 #define MAXIMUM_ADDRESS 1002
00053
00054 /*****/
00055 /*****/
00056 #define LENGTH_INIT_MSG 4
00057 #define LENGTH_INIT_RSP 6
00058 /*****/
00059 #define LENGTH_RESET_MSG 4
00060 /*****/
00061 #define LENGTH_GO_ACTIVE_MSG 4
00062 /*****/
00063 #define LENGTH_GO_SLEEP_MSG 4
00064 /*****/
00065 #define LENGTH_GO_DEEP_SLEEP_MSG 4
00066 /*****/
00067 #define LENGTH_CLR_ERROR_MSG 4
00068 /*****/
00069 #define NO_OSP_RSP false
00070 #define OSP_RSP !NO_OSP_RSP
00071 #define LENGTH_NO_OSP_RSP 0
00072 /*****/
00076 enum OSP_GENERIC_DEVICE_CMDS
00077 {
00078     OSP_RESET = 0x00,
00079     OSP_CLR_ERROR = 0x01,
00080     OSP_INIT_BIDIR = 0x02,
00081     OSP_INIT_LOOP = 0x03,
00082     OSP_GO_SLEEP = 0x04,
00083     OSP_GO_ACTIVE = 0x05,
00084     OSP_GO_DEEP_SLEEP = 0x06,
00085 };
00086
00087 /*****/
00088 /*****/
00092 enum OSP_ERROR_CODE
00093 {
00094     OSP_NO_ERROR = 0x00,
00095     OSP_ADDRESS_ERROR,
00096     OSP_ERROR_INITIALIZATION,
00097     OSP_ERROR_CRC,
00099     OSP_ERROR_SPI,
00100     OSP_ERROR_PARAMETER,
00101     OSP_ERROR_NOT_IMPLEMENTED
00102 };

```



```

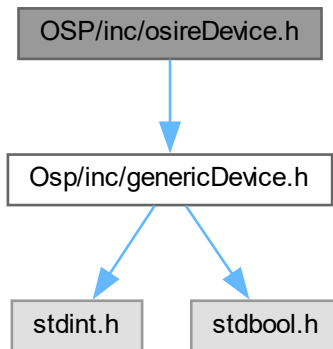
00103
00104 /*****
00105 /*****
00109 typedef struct InitRsp_t
00110 {
00111     union
00112     {
00113         uint8_t  rsp[10];
00114         struct
00115         {
00116             uint8_t temp;
00117             uint8_t status;
00118             uint16_t address :10;
00119         } bit;
00120     } data;
00121     uint16_t address :10;
00122 } ospInitRsp_t;
00123
00124 /*****
00125 /*****
00126 typedef union
00127 {
00128     uint8_t buf[4];
00129     struct
00130     {
00131         uint32_t reserved :8;
00132         uint32_t command :7;
00133         uint32_t psi :3;
00134         uint32_t address :10;
00135         uint32_t preamble :4;
00136     } bit;
00137 } ospHeader_t;
00138
00154 enum OSP_ERROR_CODE osp_init_bidir (uint16_t deviceAddress, ospInitRsp_t *p_rsp);
00155
00170 enum OSP_ERROR_CODE osp_reset (uint16_t deviceAddress);
00171
00185 enum OSP_ERROR_CODE osp_go_active (uint16_t deviceAddress);
00186
00199 enum OSP_ERROR_CODE osp_init_loop (uint16_t deviceAddress, ospInitRsp_t *p_rsp);
00200
00211 enum OSP_ERROR_CODE osp_go_sleep (uint16_t deviceAddress);
00212
00223 enum OSP_ERROR_CODE osp_go_deep_sleep (uint16_t deviceAddress);
00224
00234 enum OSP_ERROR_CODE osp_osire_clr_error (uint16_t deviceAddress);
00235
00247 void build_header (uint8_t *p_msg, uint16_t deviceAddress, uint8_t command,
00248                    uint8_t lengthMsg);
00249
00250 #ifdef __cplusplus
00251 }
00252 #endif
00253
00254 #endif // OSP_INC_GENERIC_DEVICE_H_

```

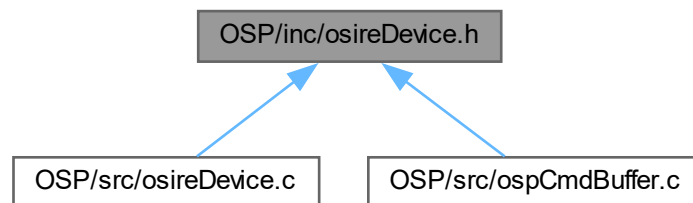
5.4 OSP/inc/osireDevice.h File Reference

```
#include <Osp/inc/genericDevice.h>
```

Include dependency graph for osireDevice.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [SetSetupData_t](#)
- struct [PwmData_t](#)
- struct [OtpData_t](#)
- struct [OtpDataComplete_t](#)
- struct [LedStatus_t](#)
- struct [osireTemp_t](#)
- struct [ComStatus_t](#)
- struct [Status_t](#)
- struct [TempStatus_t](#)
- struct [OthData_t](#)

Macros

- `#define` [LENGTH_SET_SETUP_MSG](#) 5

- `#define LENGTH_SET_PWM_MSG 10`
- `#define LENGTH_SET_OTTH_MSG 7`
- `#define LENGTH_READ_PWM_MSG 4`
- `#define LENGTH_READ_PWM_RSP 10`
- `#define LENGTH_READ_OTP_MSG 5`
- `#define LENGTH_READ_OTP_RSP 12`
- `#define LENGTH_READ_SETUP_MSG 4`
- `#define LENGTH_READ_SETUP_RSP 5`
- `#define LENGTH_READ_TEMPSTATUS_MSG 4`
- `#define LENGTH_READ_TEMPSTATUS_RSP 6`
- `#define LENGTH_READ_TEMP_MSG 4`
- `#define LENGTH_READ_TEMP_RSP 5`
- `#define LENGTH_READ_OTTH_MSG 4`
- `#define LENGTH_READ_OTTH_RSP 7`
- `#define LENGTH_READ_COMSTATUS_MSG 4`
- `#define LENGTH_READ_COMSTATUS_RSP 5`
- `#define LENGTH_READ_STATUS_MSG 4`
- `#define LENGTH_READ_STATUS_RSP 5`
- `#define LENGTH_READ_LEDSTATUS_MSG 4`
- `#define LENGTH_READ_LEDSTATUS_RSP 5`
- `#define LENGTH_P4ERROR_MSG 4`

Typedefs

- `typedef struct SetSetupData_t osireSetSetupData_t`
- `typedef struct PwmData_t osirePwmData_t`
- `typedef struct OtpData_t osireOtpData_t`
- `typedef struct OtpDataComplete_t osireOtpDataComplete_t`
- `typedef struct LedStatus_t osireLedStatus_t`
- `typedef struct osireTemp_t osireTemp_t`
- `typedef struct ComStatus_t osireComStatus_t`
- `typedef struct Status_t osireStatus_t`
- `typedef struct TempStatus_t osireTempStatus_t`
- `typedef struct OtthData_t osireOtthData_t`

Enumerations

- `enum OSP_OSIRE_DEVICE_CMDS {`
`OSP_OSIRE_P4ERROR_BIDIR = 0x08 , OSP_OSIRE_CLR_ERROR_SR = 0x21 , OSP_OSIRE_GO_SLEEP_SR`
`= 0x24 , OSP_OSIRE_GO_ACTIVE_SR = 0x25 ,`
`OSP_OSIRE_GO_DEEP_SLEEP_SR = 0x26 , OSP_OSIRE_READ_STATUS = 0x40 , OSP_OSIRE_READ_TEMP_STATUS`
`= 0x42 , OSP_OSIRE_READ_COM_STATUS = 0x44 ,`
`OSP_OSIRE_READ_LED_STATUS = 0x46 , OSP_OSIRE_READ_TEMP = 0x48 , OSP_OSIRE_READ_OTTH`
`= 0x4A , OSP_OSIRE_SET_OTTH = 0x4B ,`
`OSP_OSIRE_SET_OTTH_SR = 0x6B , OSP_OSIRE_READ_SETUP = 0x4C , OSP_OSIRE_SET_SETUP`
`= 0x4D , OSP_OSIRE_SET_SETUP_SR = 0x6D ,`
`OSP_OSIRE_READ_PWM = 0x4E , OSP_OSIRE_SET_PWM = 0x4F , OSP_OSIRE_SET_PWM_SR =`
`0x6F , OSP_OSIRE_READ_OTP = 0x58 ,`
`OSP_OSIRE_P4ERROR_LOOP = 0x09 }`

Functions

- enum [OSP_ERROR_CODE](#) [osp_osire_set_setup](#) (uint16_t deviceAddress, [osireSetSetupData_t](#) data)
OSP_OSIRE_SET_SETUP command.
- enum [OSP_ERROR_CODE](#) [osp_osire_set_setup_and_sr](#) (uint16_t deviceAddress, [osireSetSetupData_t](#) data, [osireTempStatus_t](#) *p_data)
OSP_OSIRE_SET_SETUP_SR command and reads status and temperature.
- enum [OSP_ERROR_CODE](#) [osp_osire_set_pwm](#) (uint16_t deviceAddress, [osirePwmData_t](#) data)
OSP_OSIRE_SET_PWM command.
- enum [OSP_ERROR_CODE](#) [osp_osire_set_pwm_and_sr](#) (uint16_t deviceAddress, [osirePwmData_t](#) data, [osireTempStatus_t](#) *p_data)
OSP_OSIRE_SET_PWM_SR command and reads status and temperature.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_pwm](#) (uint16_t deviceAddress, [osirePwmData_t](#) *p_data)
OSP_OSIRE_READ_PWM command.
- enum [OSP_ERROR_CODE](#) [osp_read_otp](#) (uint16_t deviceAddress, uint8_t otpAddress, [osireOtpData_t](#) *p_data)
OSP_OSIRE_READ_OTP command Reads 8 bytes from OTP memory, from otpAddress. If readout address is beyond OTP address range, 0x00 will be delivered for these addresses.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_otp_complete](#) (uint16_t deviceAddress, [osireOtpDataComplete_t](#) *p_data)
Read complete OTP memory command Reads all bytes from OTP memory. By use of several consecutive OSP_OSIRE_READ_OTP commands.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_ledstatus](#) (uint16_t deviceAddress, [osireLedStatus_t](#) *p_data)
OSP_OSIRE_READ_LED_STATUS command This function reads the LED STATUS register of the device.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_tempstatus](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_data)
OSP_OSIRE_READ_TEMP_STATUS command This function reads the STATUS and TEMP register in a single 2-byte payload of the device.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_temp](#) (uint16_t deviceAddress, [osireTemp_t](#) *p_data)
OSP_OSIRE_READ_TEMP command This function reads LED TEMP register of the device.
- enum [OSP_ERROR_CODE](#) [osp_osire_set_otth](#) (uint16_t deviceAddress, [osireOtthData_t](#) data)
OSP_OSIRE_SET_OTTH command.
- enum [OSP_ERROR_CODE](#) [osp_osire_set_otth_and_sr](#) (uint16_t deviceAddress, [osireOtthData_t](#) data, [osireTempStatus_t](#) *p_data)
OSP_OSIRE_SET_OTTH_SR command and reads status and temperature.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_otth](#) (uint16_t deviceAddress, [osireOtthData_t](#) *p_data)
OSP_OSIRE_READ_OTTH command.
- enum [OSP_ERROR_CODE](#) [osp_osire_go_sleep_and_sr](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_data)
OSP_OSIRE_GO_SLEEP_SR command and read status and temperature This function sends one or all devices into SLEEP state Additionally the STATUS and TEMP register is read.
- enum [OSP_ERROR_CODE](#) [osp_osire_go_deep_sleep_and_sr](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_data)
OSP_OSIRE_GO_DEEP_SLEEP_SR command and read status and temperature This function sends one or all devices into DEEP_SLEEP state Additionally the STATUS and TEMP register is read.
- enum [OSP_ERROR_CODE](#) [osp_osire_clr_error_and_sr](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_data)
OSP_CLEAR_ERROR command and read status and temperature This function will clear all error flags, if an error still exists for example short/open the error flag is set again and LED TEMPSTAT Register will be sent.
- enum [OSP_ERROR_CODE](#) [osp_osire_p4error_bidir](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_data)
OSP_PING_FOR_ERROR_BIDIR command Addressed or initialized device (typ. 1st.), checks if error flag occurs. It will check only the selected flag bits which leads to SLEEP state (setup register). If yes: answer to master (state + temperature) if not: same command will be forward to next device with a new address field (STARTADDRESS+1). If no error flag bit in chain. Last device sends status + temperature register. Note: The command shall use the address of the first device only. Using an address from a unit in the middle of the chain might lead to unpredictable behavior of the chain if another unit saw a reset condition (e.g. power loss).

- enum [OSP_ERROR_CODE](#) [osp_osire_p4error_loop](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_data)
OSP_PING_FOR_ERROR_LOOP command Addressed or initialized device (typ. 1st.), checks if error flag occurs. It will check only the selected flag bits which leads to SLEEP state (setup register). If yes: answer to master (state + temperature) if not: same command will be forward to next device with a new address field (STARTADDRESS+1). If no error flag bit in chain. Last device sends status + temperature register. Note: The command shall use the address of the first device only. Using an address from a unit in the middle of the chain might lead to unpredictable behavior of the chain if another unit saw a reset condition (e.g. power loss).
- enum [OSP_ERROR_CODE](#) [osp_go_active_and_sr](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_rsp)
OSP_GO_ACTIVE command and read status and temperature Puts device into ACTIVE state: The LED drivers are enabled. The last PWM parameters are used to create LED PWM signals. An PWM parameter update via SETPWM() command is possible. During ACTIVE mode, diagnostic is possible and readable.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_setup](#) (uint16_t deviceAddress, [osireSetSetupData_t](#) *p_data)
OSP_OSIRE_READ_SETUP command This function reads SETUP register of the device.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_comstatus](#) (uint16_t deviceAddress, [osireComStatus_t](#) *p_data)
OSP_OSIRE_READ_COM_STATUS command This function reads COM STATUS register of the device.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_status](#) (uint16_t deviceAddress, [osireStatus_t](#) *p_data)
OSP_OSIRE_READ_STATUS command This function reads STATUS register of the device.

5.4.1 Macro Definition Documentation

5.4.1.1 LENGTH_P4ERROR_MSG

```
#define LENGTH_P4ERROR_MSG 4
```

Definition at line 78 of file [osireDevice.h](#).

5.4.1.2 LENGTH_READ_COMSTATUS_MSG

```
#define LENGTH_READ_COMSTATUS_MSG 4
```

Definition at line 68 of file [osireDevice.h](#).

5.4.1.3 LENGTH_READ_COMSTATUS_RSP

```
#define LENGTH_READ_COMSTATUS_RSP 5
```

Definition at line 69 of file [osireDevice.h](#).

5.4.1.4 LENGTH_READ_LEDSTATUS_MSG

```
#define LENGTH_READ_LEDSTATUS_MSG 4
```

Definition at line 74 of file [osireDevice.h](#).

5.4.1.5 LENGTH_READ_LEDSTATUS_RSP

```
#define LENGTH_READ_LEDSTATUS_RSP 5
```

Definition at line 75 of file [osireDevice.h](#).

5.4.1.6 LENGTH_READ_OTP_MSG

```
#define LENGTH_READ_OTP_MSG 5
```

Definition at line 53 of file [osireDevice.h](#).

5.4.1.7 LENGTH_READ_OTP_RSP

```
#define LENGTH_READ_OTP_RSP 12
```

Definition at line 54 of file [osireDevice.h](#).

5.4.1.8 LENGTH_READ_OTTH_MSG

```
#define LENGTH_READ_OTTH_MSG 4
```

Definition at line 65 of file [osireDevice.h](#).

5.4.1.9 LENGTH_READ_OTTH_RSP

```
#define LENGTH_READ_OTTH_RSP 7
```

Definition at line 66 of file [osireDevice.h](#).

5.4.1.10 LENGTH_READ_PWM_MSG

```
#define LENGTH_READ_PWM_MSG 4
```

Definition at line 50 of file [osireDevice.h](#).

5.4.1.11 LENGTH_READ_PWM_RSP

```
#define LENGTH_READ_PWM_RSP 10
```

Definition at line 51 of file [osireDevice.h](#).

5.4.1.12 LENGTH_READ_SETUP_MSG

```
#define LENGTH_READ_SETUP_MSG 4
```

Definition at line 56 of file [osireDevice.h](#).

5.4.1.13 LENGTH_READ_SETUP_RSP

```
#define LENGTH_READ_SETUP_RSP 5
```

Definition at line 57 of file [osireDevice.h](#).

5.4.1.14 LENGTH_READ_STATUS_MSG

```
#define LENGTH_READ_STATUS_MSG 4
```

Definition at line 71 of file [osireDevice.h](#).

5.4.1.15 LENGTH_READ_STATUS_RSP

```
#define LENGTH_READ_STATUS_RSP 5
```

Definition at line 72 of file [osireDevice.h](#).

5.4.1.16 LENGTH_READ_TEMP_MSG

```
#define LENGTH_READ_TEMP_MSG 4
```

Definition at line 62 of file [osireDevice.h](#).

5.4.1.17 LENGTH_READ_TEMP_RSP

```
#define LENGTH_READ_TEMP_RSP 5
```

Definition at line 63 of file [osireDevice.h](#).

5.4.1.18 LENGTH_READ_TEMPSTATUS_MSG

```
#define LENGTH_READ_TEMPSTATUS_MSG 4
```

Definition at line 59 of file [osireDevice.h](#).

5.4.1.19 LENGTH_READ_TEMPSTATUS_RSP

```
#define LENGTH_READ_TEMPSTATUS_RSP 6
```

Definition at line 60 of file [osireDevice.h](#).

5.4.1.20 LENGTH_SET_OTTH_MSG

```
#define LENGTH_SET_OTTH_MSG 7
```

Definition at line 48 of file [osireDevice.h](#).

5.4.1.21 LENGTH_SET_PWM_MSG

```
#define LENGTH_SET_PWM_MSG 10
```

Definition at line 46 of file [osireDevice.h](#).

5.4.1.22 LENGTH_SET_SETUP_MSG

```
#define LENGTH_SET_SETUP_MSG 5
```

Brief OSP Library
Customer API for RGBi LED Stripe communication
For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf" Include OSP generic definitions and data structures
Definition at line 44 of file [osireDevice.h](#).

5.4.2 Typedef Documentation

5.4.2.1 osireComStatus_t

```
typedef struct ComStatus_t osireComStatus_t
```

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

5.4.2.2 osireLedStatus_t

```
typedef struct LedStatus_t osireLedStatus_t
```

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

5.4.2.3 osireOtpData_t

```
typedef struct OtpData_t osireOtpData_t
```

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

5.4.2.4 osireOtpDataComplete_t

```
typedef struct OtpDataComplete_t osireOtpDataComplete_t
```

5.4.2.5 osireOtthData_t

```
typedef struct OtthData_t osireOtthData_t
```

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

5.4.2.6 osirePwmData_t

```
typedef struct PwmData_t osirePwmData_t
```

5.4.2.7 osireSetSetupData_t

```
typedef struct SetSetupData_t osireSetSetupData_t
```

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

5.4.2.8 osireStatus_t

```
typedef struct Status_t osireStatus_t
```

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

5.4.2.9 osireTemp_t

```
typedef struct osireTemp_t osireTemp_t
```

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

5.4.2.10 osireTempStatus_t

```
typedef struct TempStatus_t osireTempStatus_t
```

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

5.4.3 Enumeration Type Documentation

5.4.3.1 OSP_OSIRE_DEVICE_CMDS

```
enum OSP_OSIRE_DEVICE_CMDS
```

Enumeration OSP OSire Device commands

Enumerator

OSP_OSIRE_P4ERROR_BIDIR	implemented
OSP_OSIRE_CLR_ERROR_SR	implemented
OSP_OSIRE_GO_SLEEP_SR	implemented
OSP_OSIRE_GO_ACTIVE_SR	implemented
OSP_OSIRE_GO_DEEP_SLEEP_SR	implemented
OSP_OSIRE_READ_STATUS	implemented
OSP_OSIRE_READ_TEMP_STATUS	implemented
OSP_OSIRE_READ_COM_STATUS	implemented
OSP_OSIRE_READ_LED_STATUS	implemented
OSP_OSIRE_READ_TEMP	implemented
OSP_OSIRE_READ_OTTH	implemented
OSP_OSIRE_SET_OTTH	implemented

Enumerator

OSP_OSIRE_SET_OTTH_SR	implemented
OSP_OSIRE_READ_SETUP	implemented
OSP_OSIRE_SET_SETUP	implemented
OSP_OSIRE_SET_SETUP_SR	implemented
OSP_OSIRE_READ_PWM	implemented
OSP_OSIRE_SET_PWM	implemented
OSP_OSIRE_SET_PWM_SR	implemented
OSP_OSIRE_READ_OTP	implemented
OSP_OSIRE_P4ERROR_LOOP	implemented; precondition: OSP_INIT_LOOP

Definition at line 85 of file [osireDevice.h](#).

```

00086 {
00087     OSP_OSIRE_P4ERROR_BIDIR = 0x08,
00088     OSP_OSIRE_CLR_ERROR_SR = 0x21,
00089     OSP_OSIRE_GO_SLEEP_SR = 0x24,
00090     OSP_OSIRE_GO_ACTIVE_SR = 0x25,
00091     OSP_OSIRE_GO_DEEP_SLEEP_SR = 0x26,
00092     OSP_OSIRE_READ_STATUS = 0x40,
00093     OSP_OSIRE_READ_TEMP_STATUS = 0x42,
00094     OSP_OSIRE_READ_COM_STATUS = 0x44,
00095     OSP_OSIRE_READ_LED_STATUS = 0x46,
00096     OSP_OSIRE_READ_TEMP = 0x48,
00097     OSP_OSIRE_READ_OTTH = 0x4A,
00098     OSP_OSIRE_SET_OTTH = 0x4B,
00099     OSP_OSIRE_SET_OTTH_SR = 0x6B,
00100     OSP_OSIRE_READ_SETUP = 0x4C,
00101     OSP_OSIRE_SET_SETUP = 0x4D,
00102     OSP_OSIRE_SET_SETUP_SR = 0x6D,
00103     OSP_OSIRE_READ_PWM = 0x4E,
00104     OSP_OSIRE_SET_PWM = 0x4F,
00105     OSP_OSIRE_SET_PWM_SR = 0x6F,
00106     OSP_OSIRE_READ_OTP = 0x58,
00108     OSP_OSIRE_P4ERROR_LOOP = 0x09
00109 };

```

5.4.4 Function Documentation

5.4.4.1 osp_go_active_and_sr()

```

enum OSP_ERROR_CODE osp_go_active_and_sr (
    uint16_t deviceAddress,
    osireTempStatus_t * p_rsp )

```

OSP_GO_ACTIVE command and read status and temperature Puts device into ACTIVE state: The LED drivers are enabled. The last PWM parameters are used to create LED PWM signals. An PWM parameter update via SETPWM() command is possible. During ACTIVE mode, diagnostic is possible and readable.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

Definition at line 563 of file [osireDevice.c](#).

```

00565 {
00566     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00567     ospCmdBuffer_t ospCmd;
00568     enum OSP_ERROR_CODE ospErrorCode;
00569     errorSpi_t spiError;

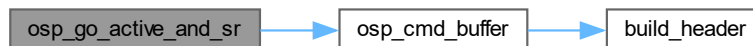
```

```

00570
00571  ospCmd.inCmdId = OSP_OSIRE_GO_ACTIVE_SR;
00572  ospCmd.inDeviceAddress = deviceAddress;
00573  ospCmd.p_inParameter = NULL;
00574
00575  ospErrorCode = osp_cmd_buffer (&ospCmd);
00576  if (ospErrorCode != OSP_NO_ERROR)
00577  {
00578      return ospErrorCode;
00579  }
00580
00581  spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00582                                                    rspBuffer,
00583                                                    ospCmd.outCmdBufferLength,
00584                                                    ospCmd.outResponseLength);
00585
00586  if (spiError != NO_ERROR_SPI)
00587  {
00588      return OSP_ERROR_SPI;
00589  }
00590
00591  if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00592  {
00593      return OSP_ERROR_CRC;
00594  }
00595
00596  for (uint8_t i = 0; i < 2; i++)
00597  {
00598      p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00599  }
00600
00601  p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00602  return OSP_NO_ERROR;
00603 }

```

Here is the call graph for this function:



5.4.4.2 osp_osire_clr_error_and_sr()

```

enum OSP_ERROR_CODE osp_osire_clr_error_and_sr (
    uint16_t deviceAddress,
    osireTempStatus_t * p_data )

```

OSP_CLEAR_ERROR command and read status and temperature This function will clear all error flags, if an error still exists for example short/open the error flag is set again and LED TEMPSTAT Register will be sent.

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

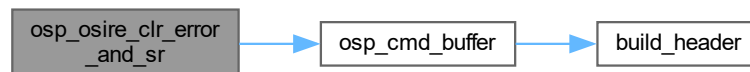
Definition at line 699 of file `osireDevice.c`.

```

00701 {
00702     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // message buffer
00703     ospCmdBuffer_t ospCmd;
00704     enum OSP_ERROR_CODE ospErrorCode;
00705     errorSpi_t spiError;
00706
00707     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00708
00709     ospCmd.inCmdId = OSP_OSIRE_CLR_ERROR_SR;
00710     ospCmd.inDeviceAddress = deviceAddress;
00711     ospCmd.p_inParameter = NULL;
00712
00713     ospErrorCode = osp_cmd_buffer (&ospCmd);
00714     if (ospErrorCode != OSP_NO_ERROR)
00715     {
00716         return ospErrorCode;
00717     }
00718
00719     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00720                                                         rspBuffer,
00721                                                         ospCmd.outCmdBufferLength,
00722                                                         ospCmd.outResponseLength);
00723
00724     if (spiError != NO_ERROR_SPI)
00725     {
00726         return OSP_ERROR_SPI;
00727     }
00728
00729     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00730     {
00731         return OSP_ERROR_CRC;
00732     }
00733
00734     for (uint8_t i = 0; i < 2; i++)
00735     {
00736         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00737     }
00738
00739     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00740     return OSP_NO_ERROR;
00741 }

```

Here is the call graph for this function:

**5.4.4.3 osp_osire_go_deep_sleep_and_sr()**

```

enum OSP_ERROR_CODE osp_osire_go_deep_sleep_and_sr (
    uint16_t deviceAddress,
    osireTempStatus_t * p_data )

```

OSP_OSIRE_GO_DEEP_SLEEP_SR command and read status and temperature This function sends one or all devices into DEEP_SLEEP state Additionally the STATUS and TEMP register is read.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

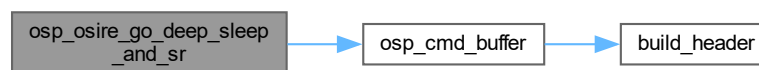
Definition at line 653 of file `osireDevice.c`.

```

00655 {
00656     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00657     ospCmdBuffer_t ospCmd;
00658     enum OSP_ERROR_CODE ospErrorCode;
00659     errorSpi_t spiError;
00660
00661     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00662
00663     ospCmd.inCmdId = OSP_OSIRE_GO_DEEP_SLEEP_SR;
00664     ospCmd.inDeviceAddress = deviceAddress;
00665     ospCmd.p_inParameter = NULL;
00666
00667     ospErrorCode = osp_cmd_buffer (&ospCmd);
00668     if (ospErrorCode != OSP_NO_ERROR)
00669     {
00670         return ospErrorCode;
00671     }
00672
00673     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00674                                                         rspBuffer,
00675                                                         ospCmd.outCmdBufferLength,
00676                                                         ospCmd.outResponseLength);
00677
00678     if (spiError != NO_ERROR_SPI)
00679     {
00680         return OSP_ERROR_SPI;
00681     }
00682
00683     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00684     {
00685         return OSP_ERROR_CRC;
00686     }
00687
00688     for (uint8_t i = 0; i < 2; i++)
00689     {
00690         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00691     }
00692
00693     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00694     return OSP_NO_ERROR;
00695 }

```

Here is the call graph for this function:



5.4.4.4 osp_osire_go_sleep_and_sr()

```

enum OSP_ERROR_CODE osp_osire_go_sleep_and_sr (
    uint16_t deviceAddress,
    osireTempStatus_t * p_data )

```

OSP_OSIRE_GO_SLEEP_SR command and read status and temperature This function sends one or all devices into SLEEP state Additionally the STATUS and TEMP register is read.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

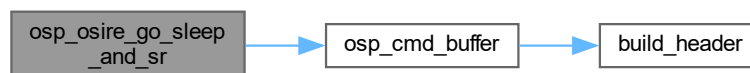
Definition at line 607 of file `osireDevice.c`.

```

00609 {
00610     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00611     ospCmdBuffer_t ospCmd;
00612     enum OSP_ERROR_CODE ospErrorCode;
00613     errorSpi_t spiError;
00614
00615     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00616
00617     ospCmd.inCmdId = OSP_OSIRE_GO_SLEEP_SR;
00618     ospCmd.inDeviceAddress = deviceAddress;
00619     ospCmd.p_inParameter = NULL;
00620
00621     ospErrorCode = osp_cmd_buffer (&ospCmd);
00622     if (ospErrorCode != OSP_NO_ERROR)
00623     {
00624         return ospErrorCode;
00625     }
00626
00627     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00628                                                         rspBuffer,
00629                                                         ospCmd.outCmdBufferLength,
00630                                                         ospCmd.outResponseLength);
00631
00632     if (spiError != NO_ERROR_SPI)
00633     {
00634         return OSP_ERROR_SPI;
00635     }
00636
00637     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00638     {
00639         return OSP_ERROR_CRC;
00640     }
00641
00642     for (uint8_t i = 0; i < 2; i++)
00643     {
00644         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00645     }
00646
00647     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00648     return OSP_NO_ERROR;
00649 }

```

Here is the call graph for this function:



5.4.4.5 osp_osire_p4error_bidir()

```

enum OSP_ERROR_CODE osp_osire_p4error_bidir (
    uint16_t deviceAddress,
    osireTempStatus_t * p_data )

```

OSP_PING_FOR_ERROR_BIDIR command Addressed or initialized device (typ. 1st.), checks if error flag occurs. It will check only the selected flag bits which leads to SLEEP state (setup register). If yes: answer to master (state + temperature) if not: same command will be forward to next device with a new address field (STARTADDRESS+1). If no error flag bit in chain. Last device sends status + temperature register. Note: The command shall use the address of the first device only. Using an address from a unit in the middle of the chain might lead to unpredictable behavior of the chain if another unit saw a reset condition (e.g. power loss).

Parameters

<i>deviceAddress</i>	of first RGBi device (typ. 1)
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

Parameters

<i>deviceAddress</i>	of first RGBi device (typ. 1)
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

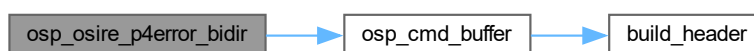
Definition at line 745 of file `osireDevice.c`.

```

00747 {
00748     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // message buffer
00749     ospCmdBuffer_t ospCmd;
00750     enum OSP_ERROR_CODE ospErrorCode;
00751     errorSpi_t spiError;
00752
00753     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00754
00755     ospCmd.inCmdId = OSP_OSIRE_P4ERROR_BIDIR;
00756     ospCmd.inDeviceAddress = deviceAddress;
00757     ospCmd.p_inParameter = NULL;
00758
00759     ospErrorCode = osp_cmd_buffer (&ospCmd);
00760     if (ospErrorCode != OSP_NO_ERROR)
00761     {
00762         return ospErrorCode;
00763     }
00764
00765     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00766                                                         rspBuffer,
00767                                                         ospCmd.outCmdBufferLength,
00768                                                         ospCmd.outResponseLength);
00769
00770     if (spiError != NO_ERROR_SPI)
00771     {
00772         return OSP_ERROR_SPI;
00773     }
00774
00775     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00776     {
00777         return OSP_ERROR_CRC;
00778     }
00779
00780     for (uint8_t i = 0; i < 2; i++)
00781     {
00782         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00783     }
00784
00785     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00786     return OSP_NO_ERROR;
00787 }

```

Here is the call graph for this function:



5.4.4.6 osp_osire_p4error_loop()

```
enum OSP_ERROR_CODE osp_osire_p4error_loop (
    uint16_t deviceAddress,
    osireTempStatus_t * p_data )
```

OSP_PING_FOR_ERROR_LOOP command Addressed or initialized device (typ. 1st.), checks if error flag occurs. It will check only the selected flag bits which leads to SLEEP state (setup register). If yes: answer to master (state + temperature) if not: same command will be forward to next device with a new address field (STARTADDRESS+1). If no error flag bit in chain. Last device sends status + temperature register. Note: The command shall use the address of the first device only. Using an address from a unit in the middle of the chain might lead to unpredictable behavior of the chain if another unit saw a reset condition (e.g. power loss).

Parameters

<i>deviceAddress</i>	of first RGBi device (typ. 1)
<i>p_data,pointer</i>	to response data from RGBi LED

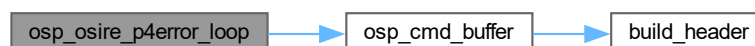
Returns

error communication or command parameter error

Definition at line 790 of file [osireDevice.c](#).

```
00792 {
00793     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // message buffer
00794     ospCmdBuffer_t ospCmd;
00795     enum OSP_ERROR_CODE ospErrorCode;
00796     errorSpi_t spiError;
00797
00798     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00799
00800     ospCmd.inCmdId = OSP_OSIRE_P4ERROR_LOOP;
00801     ospCmd.inDeviceAddress = deviceAddress;
00802     ospCmd.p_inParameter = NULL;
00803
00804     ospErrorCode = osp_cmd_buffer (&ospCmd);
00805     if (ospErrorCode != OSP_NO_ERROR)
00806     {
00807         return ospErrorCode;
00808     }
00809
00810     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00811                                                         rspBuffer,
00812                                                         ospCmd.outCmdBufferLength,
00813                                                         ospCmd.outResponseLength);
00814
00815     if (spiError != NO_ERROR_SPI)
00816     {
00817         return OSP_ERROR_SPI;
00818     }
00819
00820     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00821     {
00822         return OSP_ERROR_CRC;
00823     }
00824
00825     for (uint8_t i = 0; i < 2; i++)
00826     {
00827         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00828     }
00829
00830     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00831     return OSP_NO_ERROR;
00832 }
```

Here is the call graph for this function:



5.4.4.7 osp_osire_read_comstatus()

```
enum OSP_ERROR_CODE osp_osire_read_comstatus (
    uint16_t deviceAddress,
    osireComStatus_t * p_data )
```

OSP_OSIRE_READ_COM_STATUS command This function reads COM STATUS register of the device.
For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	pointer to response data from RGBi LED

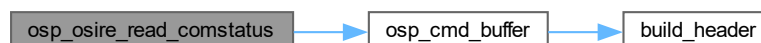
Returns

error COM STATUS register response data

Definition at line 878 of file `osireDevice.c`.

```
00880 {
00881     uint8_t rspBuffer[LENGTH_READ_COMSTATUS_RSP]; // response buffer
00882     ospCmdBuffer_t ospCmd;
00883     enum OSP_ERROR_CODE ospErrorCode;
00884     errorSpi_t spiError;
00885
00886     memset (rspBuffer, 0, LENGTH_READ_COMSTATUS_RSP);
00887
00888     ospCmd.inCmdId = OSP_OSIRE_READ_COM_STATUS;
00889     ospCmd.inDeviceAddress = deviceAddress;
00890     ospCmd.p_inParameter = NULL;
00891
00892     ospErrorCode = osp_cmd_buffer (&ospCmd);
00893     if (ospErrorCode != OSP_NO_ERROR)
00894     {
00895         return ospErrorCode;
00896     }
00897
00898     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00899                                                         rspBuffer,
00900                                                         ospCmd.outCmdBufferLength,
00901                                                         ospCmd.outResponseLength);
00902
00903     if (spiError != NO_ERROR_SPI)
00904     {
00905         return OSP_ERROR_SPI;
00906     }
00907
00908     if (crc (rspBuffer, LENGTH_READ_COMSTATUS_RSP) != 0)
00909     {
00910         return OSP_ERROR_CRC;
00911     }
00912
00913     p_rsp->data.comStatus = rspBuffer[FIRST_BYTE_PAYLOAD];
00914
00915     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00916     return OSP_NO_ERROR;
00917 }
```

Here is the call graph for this function:



5.4.4.8 osp_osire_read_ledstatus()

```
enum OSP_ERROR_CODE osp_osire_read_ledstatus (
    uint16_t deviceAddress,
    osireLedStatus_t * p_data )
```

OSP_OSIRE_READ_LED_STATUS command This function reads the LED STATUS register of the device. For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	LED STATUS register response data

Returns

error communication or command parameter error

Definition at line 308 of file `osireDevice.c`.

```
00310 {
00311     uint8_t rspBuffer[LENGTH_READ_LEDSTATUS_RSP]; // response buffer
00312     ospCmdBuffer_t ospCmd;
00313     enum OSP_ERROR_CODE ospErrorCode;
00314     errorSpi_t spiError;
00315
00316     memset (rspBuffer, 0, LENGTH_READ_LEDSTATUS_RSP);
00317
00318     ospCmd.inCmdId = OSP_OSIRE_READ_LED_STATUS;
00319     ospCmd.inDeviceAddress = deviceAddress;
00320     ospCmd.p_inParameter = NULL;
00321
00322     ospErrorCode = osp_cmd_buffer (&ospCmd);
00323     if (ospErrorCode != OSP_NO_ERROR)
00324     {
00325         return ospErrorCode;
00326     }
00327
00328     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00329                                                         rspBuffer,
00330                                                         ospCmd.outCmdBufferLength,
00331                                                         ospCmd.outResponseLength);
00332
00333     if (spiError != NO_ERROR_SPI)
00334     {
00335         return OSP_ERROR_SPI;
00336     }
00337
00338     if (crc (rspBuffer, LENGTH_READ_LEDSTATUS_RSP) != 0)
00339     {
00340         return OSP_ERROR_CRC;
00341     }
00342
00343     p_rsp->data.ledStatus = rspBuffer[3];
00344
00345     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00346     return OSP_NO_ERROR;
00347 }
```

Here is the call graph for this function:



5.4.4.9 osp_osire_read_otp_complete()

```
enum OSP_ERROR_CODE osp_osire_read_otp_complete (
```

```
uint16_t deviceAddress,
osireOtpDataComplete_t * p_data )
```

Read complete OTP memory command Reads all bytes from OTP memory. By use of several consecutive OSP↔_OSIRE_READ_OTP commands.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	full OTP memory data

Returns

error communication or command parameter error

Definition at line 282 of file `osireDevice.c`.

```
00284 {
00285     osireOtpData_t opt; // OTP buffer
00286
00287     for (uint8_t i = 0; i < 0x1F; i = i + 8)
00288     {
00289         enum OSP_ERROR_CODE errorCode = osp_read_otp (deviceAddress, i, &opt);
00290
00291         if (errorCode != OSP_NO_ERROR)
00292         {
00293             return errorCode;
00294         }
00295
00296         for (uint8_t j = 0; j < 8; j++)
00297         {
00298             p_rsp->data.byte[i + j] = opt.data.byte[j];
00299         }
00300     }
00301
00302     p_rsp->address = opt.address;
00303     return OSP_NO_ERROR;
00304 }
```

Here is the call graph for this function:



5.4.4.10 osp_osire_read_otth()

```
enum OSP_ERROR_CODE osp_osire_read_otth (
    uint16_t deviceAddress,
    osireOtthData_t * p_data )
```

OSP_OSIRE_READ_OTTH command.

This function reads the OTTH register of the device

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	PWM register response data

Returns

error communication or command parameter error

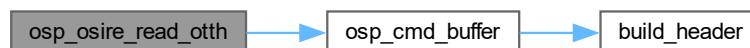
Definition at line 517 of file `osireDevice.c`.

```

00519 {
00520     uint8_t rspBuffer[LENGTH_READ_OTTH_RSP]; // response buffer
00521     ospCmdBuffer_t ospCmd;
00522     enum OSP_ERROR_CODE ospErrorCode;
00523     errorSpi_t spiError;
00524
00525     memset (rspBuffer, 0, LENGTH_READ_OTTH_RSP);
00526
00527     ospCmd.inCmdId = OSP_OSIRE_READ_OTTH;
00528     ospCmd.inDeviceAddress = deviceAddress;
00529     ospCmd.p_inParameter = NULL;
00530
00531     ospErrorCode = osp_cmd_buffer (&ospCmd);
00532     if (ospErrorCode != OSP_NO_ERROR)
00533     {
00534         return ospErrorCode;
00535     }
00536
00537     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00538                                                         rspBuffer,
00539                                                         ospCmd.outCmdBufferLength,
00540                                                         ospCmd.outResponseLength);
00541
00542     if (spiError != NO_ERROR_SPI)
00543     {
00544         return OSP_ERROR_SPI;
00545     }
00546
00547     if (crc (rspBuffer, LENGTH_READ_OTTH_RSP) != 0)
00548     {
00549         return OSP_ERROR_CRC;
00550     }
00551
00552     for (uint8_t i = 0; i < 3; i++)
00553     {
00554         p_rsp->data.otthData[i] = rspBuffer[5 - i];
00555     }
00556
00557     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00558     return OSP_NO_ERROR;
00559 }

```

Here is the call graph for this function:

**5.4.4.11 osp_osire_read_pwm()**

```

enum OSP_ERROR_CODE osp_osire_read_pwm (
    uint16_t deviceAddress,
    osirePwmData_t * p_data )

```

OSP_OSIRE_READ_PWM command.

This function reads the PWM register of the device

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	PWM register response data

Returns

error communication or command parameter error

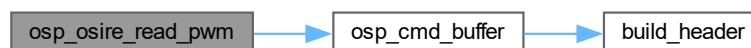
Definition at line 190 of file `osireDevice.c`.

```

00192 {
00193     uint8_t rspBuffer[LENGTH_READ_PWM_RSP]; // response buffer
00194     ospCmdBuffer_t ospCmd;
00195     enum OSP_ERROR_CODE ospErrorCode;
00196     errorSpi_t spiError;
00197
00198     memset (rspBuffer, 0, LENGTH_READ_PWM_RSP);
00199
00200     ospCmd.inCmdId = OSP_OSIRE_READ_PWM;
00201     ospCmd.inDeviceAddress = deviceAddress;
00202     ospCmd.p_inParameter = NULL;
00203
00204     ospErrorCode = osp_cmd_buffer (&ospCmd);
00205     if (ospErrorCode != OSP_NO_ERROR)
00206     {
00207         return ospErrorCode;
00208     }
00209
00210     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00211                                                         rspBuffer,
00212                                                         ospCmd.outCmdBufferLength,
00213                                                         ospCmd.outResponseLength);
00214
00215     if (spiError != NO_ERROR_SPI)
00216     {
00217         return OSP_ERROR_SPI;
00218     }
00219
00220     if (crc (rspBuffer, LENGTH_READ_PWM_RSP) != 0)
00221     {
00222         return OSP_ERROR_CRC;
00223     }
00224
00225     for (uint8_t i = 0; i < 6; i++)
00226     {
00227         p_rsp->data.pwmData[i] = rspBuffer[8 - i];
00228     }
00229
00230     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00231     return OSP_NO_ERROR;
00232 }

```

Here is the call graph for this function:



5.4.4.12 osp_osire_read_setup()

```

enum OSP_ERROR_CODE osp_osire_read_setup (
    uint16_t deviceAddress,
    osireSetSetupData_t * p_data )

```

OSP_OSIRE_READ_SETUP command This function reads SETUP register of the device.
For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	SETUP register response data

Returns

error communication or command parameter error

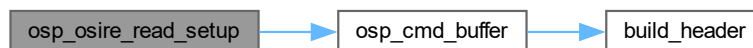
Definition at line 835 of file `osireDevice.c`.

```

00837 {
00838     uint8_t rspBuffer[LENGTH_READ_SETUP_RSP]; // response buffer
00839     ospCmdBuffer_t ospCmd;
00840     enum OSP_ERROR_CODE ospErrorCode;
00841     errorSpi_t spiError;
00842
00843     memset (rspBuffer, 0, LENGTH_READ_SETUP_RSP);
00844
00845     ospCmd.inCmdId = OSP_OSIRE_READ_SETUP;
00846     ospCmd.inDeviceAddress = deviceAddress;
00847     ospCmd.p_inParameter = NULL;
00848
00849     ospErrorCode = osp_cmd_buffer (&ospCmd);
00850     if (ospErrorCode != OSP_NO_ERROR)
00851     {
00852         return ospErrorCode;
00853     }
00854
00855     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00856                                                         rspBuffer,
00857                                                         ospCmd.outCmdBufferLength,
00858                                                         ospCmd.outResponseLength);
00859
00860     if (spiError != NO_ERROR_SPI)
00861     {
00862         return OSP_ERROR_SPI;
00863     }
00864
00865     if (crc (rspBuffer, LENGTH_READ_SETUP_RSP) != 0)
00866     {
00867         return OSP_ERROR_CRC;
00868     }
00869
00870     p_rsp->data.setupData = rspBuffer[FIRST_BYTE_PAYLOAD];
00871
00872     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00873     return OSP_NO_ERROR;
00874 }

```

Here is the call graph for this function:

**5.4.4.13 osp_osire_read_status()**

```

enum OSP_ERROR_CODE osp_osire_read_status (
    uint16_t deviceAddress,
    osireStatus_t * p_data )

```

OSP_OSIRE_READ_STATUS command This function reads STATUS register of the device.
For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	STATUS register response data

Returns

error communication or command parameter error

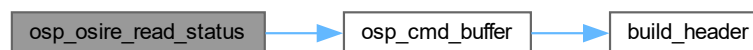
Definition at line 921 of file `osireDevice.c`.

```

00923 {
00924     uint8_t rspBuffer[LENGTH_READ_STATUS_RSP]; // response buffer
00925     ospCmdBuffer_t ospCmd;
00926     enum OSP_ERROR_CODE ospErrorCode;
00927     errorSpi_t spiError;
00928
00929     memset (rspBuffer, 0, LENGTH_READ_STATUS_RSP);
00930
00931     ospCmd.inCmdId = OSP_OSIRE_READ_STATUS;
00932     ospCmd.inDeviceAddress = deviceAddress;
00933     ospCmd.p_inParameter = NULL;
00934
00935     ospErrorCode = osp_cmd_buffer (&ospCmd);
00936     if (ospErrorCode != OSP_NO_ERROR)
00937     {
00938         return ospErrorCode;
00939     }
00940
00941     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00942                                                         rspBuffer,
00943                                                         ospCmd.outCmdBufferLength,
00944                                                         ospCmd.outResponseLength);
00945
00946     if (spiError != NO_ERROR_SPI)
00947     {
00948         return OSP_ERROR_SPI;
00949     }
00950
00951     if (crc (rspBuffer, LENGTH_READ_STATUS_RSP) != 0)
00952     {
00953         return OSP_ERROR_CRC;
00954     }
00955
00956     p_rsp->data.status = rspBuffer[FIRST_BYTE_PAYLOAD];
00957
00958     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00959     return OSP_NO_ERROR;
00960 }

```

Here is the call graph for this function:



5.4.4.14 osp_osire_read_temp()

```

enum OSP_ERROR_CODE osp_osire_read_temp (
    uint16_t deviceAddress,
    osireTemp_t * p_data )

```

OSP_OSIRE_READ_TEMP command This function reads LED TEMP register of the device.
For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	LED TEMP register response data

Returns

error communication or command parameter error

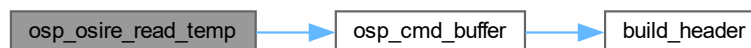
Definition at line 397 of file `osireDevice.c`.

```

00399 {
00400     uint8_t rspBuffer[LENGTH_READ_TEMP_RSP]; // response buffer
00401     ospCmdBuffer_t ospCmd;
00402     enum OSP_ERROR_CODE ospErrorCode;
00403     errorSpi_t spiError;
00404
00405     memset (rspBuffer, 0, LENGTH_READ_TEMP_RSP);
00406
00407     ospCmd.inCmdId = OSP_OSIRE_READ_TEMP;
00408     ospCmd.inDeviceAddress = deviceAddress;
00409     ospCmd.p_inParameter = NULL;
00410
00411     ospErrorCode = osp_cmd_buffer (&ospCmd);
00412     if (ospErrorCode != OSP_NO_ERROR)
00413     {
00414         return ospErrorCode;
00415     }
00416
00417     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00418                                                         rspBuffer,
00419                                                         ospCmd.outCmdBufferLength,
00420                                                         ospCmd.outResponseLength);
00421
00422     if (spiError != NO_ERROR_SPI)
00423     {
00424         return OSP_ERROR_SPI;
00425     }
00426
00427     if (crc (rspBuffer, LENGTH_READ_TEMP_RSP) != 0)
00428     {
00429         return OSP_ERROR_CRC;
00430     }
00431
00432     p_rsp->data.temp_value = rspBuffer[FIRST_BYTE_PAYLOAD];
00433
00434     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00435     return OSP_NO_ERROR;
00436 }

```

Here is the call graph for this function:

**5.4.4.15 osp_osire_read_tempstatus()**

```

enum OSP_ERROR_CODE osp_osire_read_tempstatus (
    uint16_t deviceAddress,
    osireTempStatus_t * p_data )

```

OSP_OSIRE_READ_TEMP_STATUS command This function reads the STATUS and TEMP register in a single 2-byte payload of the device.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

Definition at line 351 of file `osireDevice.c`.

```

00353 {
00354     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00355     ospCmdBuffer_t ospCmd;
00356     enum OSP_ERROR_CODE ospErrorCode;
00357     errorSpi_t spiError;
00358
00359     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00360
00361     ospCmd.inCmdId = OSP_OSIRE_READ_TEMP_STATUS;
00362     ospCmd.inDeviceAddress = deviceAddress;
00363     ospCmd.p_inParameter = NULL;
00364
00365     ospErrorCode = osp_cmd_buffer (&ospCmd);
00366     if (ospErrorCode != OSP_NO_ERROR)
00367     {
00368         return ospErrorCode;
00369     }
00370
00371     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00372                                                         rspBuffer,
00373                                                         ospCmd.outCmdBufferLength,
00374                                                         ospCmd.outResponseLength);
00375
00376     if (spiError != NO_ERROR_SPI)
00377     {
00378         return OSP_ERROR_SPI;
00379     }
00380
00381     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00382     {
00383         return OSP_ERROR_CRC;
00384     }
00385
00386     for (uint8_t i = 0; i < 2; i++)
00387     {
00388         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00389     }
00390
00391     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00392     return OSP_NO_ERROR;
00393 }

```

Here is the call graph for this function:



5.4.4.16 osp_osire_set_otth()

```

enum OSP_ERROR_CODE osp_osire_set_otth (
    uint16_t deviceAddress,
    osireOtthData_t data )

```

OSP_OSIRE_SET_OTTH command.

Writes the OTTH register. See READOTTH for the payload format.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
<i>data</i>	PWM register value

Returns

error, communication or command parameter error

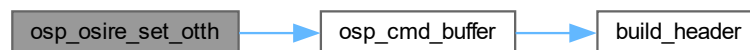
Definition at line 440 of file `osireDevice.c`.

```

00442 {
00443     ospCmdBuffer_t ospCmd;
00444     enum OSP_ERROR_CODE ospErrorCode;
00445     errorSpi_t spiError;
00446
00447     ospCmd.inCmdId = OSP_OSIRE_SET_OTTH;
00448     ospCmd.inDeviceAddress = deviceAddress;
00449     ospCmd.p_inParameter = &data.data.otthData;
00450
00451     ospErrorCode = osp_cmd_buffer (&ospCmd);
00452     if (ospErrorCode != OSP_NO_ERROR)
00453     {
00454         return ospErrorCode;
00455     }
00456
00457     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00458                                           ospCmd.outCmdBufferLength);
00459
00460     if (spiError != NO_ERROR_SPI)
00461     {
00462         return OSP_ERROR_SPI;
00463     }
00464
00465     return OSP_NO_ERROR;
00466 }

```

Here is the call graph for this function:

**5.4.4.17 osp_osire_set_otth_and_sr()**

```

enum OSP_ERROR_CODE osp_osire_set_otth_and_sr (
    uint16_t deviceAddress,
    osireOtthData_t data,
    osireTempStatus_t * p_data )

```

OSP_OSIRE_SET_OTTH_SR command and reads status and temperature.

Writes the OTTH register. See READOTTH for the payload format. Additionally the STATUS and TEMP register is read.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>data</i>	PWM register value
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

Definition at line 470 of file `osireDevice.c`.

```

00473 {
00474     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00475     ospCmdBuffer_t ospCmd;
00476     enum OSP_ERROR_CODE ospErrorCode;

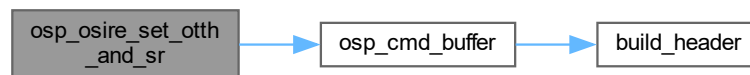
```

```

00477     errorSpi_t spiError;
00478
00479     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00480
00481     ospCmd.inCmdId = OSP_OSIRE_SET_OTTH_SR;
00482     ospCmd.inDeviceAddress = deviceAddress;
00483     ospCmd.p_inParameter = &data.data.otthData;
00484
00485     ospErrorCode = osp_cmd_buffer (&ospCmd);
00486     if (ospErrorCode != OSP_NO_ERROR)
00487     {
00488         return ospErrorCode;
00489     }
00490
00491     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00492                                                         rspBuffer,
00493                                                         ospCmd.outCmdBufferLength,
00494                                                         ospCmd.outResponseLength);
00495
00496     if (spiError != NO_ERROR_SPI)
00497     {
00498         return OSP_ERROR_SPI;
00499     }
00500
00501     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00502     {
00503         return OSP_ERROR_CRC;
00504     }
00505
00506     for (uint8_t i = 0; i < 2; i++)
00507     {
00508         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00509     }
00510
00511     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00512     return OSP_NO_ERROR;
00513 }

```

Here is the call graph for this function:



5.4.4.18 osp_osire_set_pwm()

```

enum OSP_ERROR_CODE osp_osire_set_pwm (
    uint16_t deviceAddress,
    osirePwmData_t data )

```

OSP_OSIRE_SET_PWM command.

Writes the PWM register. See READPWM for the payload format.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
<i>data</i>	PWM register value

Returns

error, communication or command parameter error

Definition at line 112 of file [osireDevice.c](#).

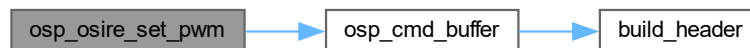
```
00114 {
```

```

00115     ospCmdBuffer_t ospCmd;
00116     enum OSP_ERROR_CODE ospErrorCode;
00117     errorSpi_t spiError;
00118
00119     ospCmd.inCmdId = OSP_OSIRE_SET_PWM;
00120     ospCmd.inDeviceAddress = deviceAddress;
00121     ospCmd.p_inParameter = &data.data.pwmData;
00122
00123     ospErrorCode = osp_cmd_buffer (&ospCmd);
00124     if (ospErrorCode != OSP_NO_ERROR)
00125     {
00126         return ospErrorCode;
00127     }
00128
00129     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00130                                           ospCmd.outCmdBufferLength);
00131
00132     if (spiError != NO_ERROR_SPI)
00133     {
00134         return OSP_ERROR_SPI;
00135     }
00136
00137     return OSP_NO_ERROR;
00138 }

```

Here is the call graph for this function:



5.4.4.19 osp_osire_set_pwm_and_sr()

```

enum OSP_ERROR_CODE osp_osire_set_pwm_and_sr (
    uint16_t deviceAddress,
    osirePwmData_t data,
    osireTempStatus_t * p_data )

```

OSP_OSIRE_SET_PWM_SR command and reads status and temperature.

Writes the PWM register. See READPWM for the payload format. Additionally the STATUS and TEMP register is read.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
<i>data</i>	PWM register value
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

Definition at line 143 of file [osireDevice.c](#).

```

00146 {
00147     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00148     ospCmdBuffer_t ospCmd;
00149     enum OSP_ERROR_CODE ospErrorCode;
00150     errorSpi_t spiError;
00151
00152     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00153
00154     ospCmd.inCmdId = OSP_OSIRE_SET_PWM_SR;
00155     ospCmd.inDeviceAddress = deviceAddress;

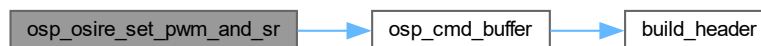
```

```

00156     ospCmd.p_inParameter = &data.data.pwmData;
00157
00158     ospErrorCode = osp_cmd_buffer (&ospCmd);
00159     if (ospErrorCode != OSP_NO_ERROR)
00160     {
00161         return ospErrorCode;
00162     }
00163
00164     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00165                                                         rspBuffer,
00166                                                         ospCmd.outCmdBufferLength,
00167                                                         ospCmd.outResponseLength);
00168
00169     if (spiError != NO_ERROR_SPI)
00170     {
00171         return OSP_ERROR_SPI;
00172     }
00173
00174     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00175     {
00176         return OSP_ERROR_CRC;
00177     }
00178
00179     for (uint8_t i = 0; i < 2; i++)
00180     {
00181         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00182     }
00183
00184     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00185     return OSP_NO_ERROR;
00186 }

```

Here is the call graph for this function:



5.4.4.20 osp_osire_set_setup()

```

enum OSP_ERROR_CODE osp_osire_set_setup (
    uint16_t deviceAddress,
    osireSetSetupData_t data )

```

OSP_OSIRE_SET_SETUP command.

Writes the SETUP register. See READSETUP for the payload format.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
<i>data</i>	SETUP register values

Returns

error communication or command parameter error

Include OSP Led Definitions and Data structures

Definition at line 36 of file [osireDevice.c](#).

```

00038 {
00039     ospCmdBuffer_t ospCmd;
00040     enum OSP_ERROR_CODE ospErrorCode;
00041     errorSpi_t spiError;
00042
00043     ospCmd.inCmdId = OSP_OSIRE_SET_SETUP;
00044     ospCmd.inDeviceAddress = deviceAddress;

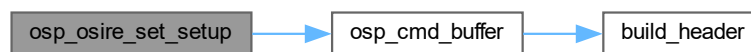
```

```

00045     ospCmd.p_inParameter = &data.data.setupData;
00046
00047     ospErrorCode = osp_cmd_buffer (&ospCmd);
00048     if (ospErrorCode != OSP_NO_ERROR)
00049     {
00050         return ospErrorCode;
00051     }
00052
00053     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00054                                           ospCmd.outCmdBufferLength);
00055
00056     if (spiError != NO_ERROR_SPI)
00057     {
00058         return OSP_ERROR_SPI;
00059     }
00060
00061     return OSP_NO_ERROR;
00062 }

```

Here is the call graph for this function:



5.4.4.21 osp_osire_set_setup_and_sr()

```

enum OSP_ERROR_CODE osp_osire_set_setup_and_sr (
    uint16_t deviceAddress,
    osireSetSetupData_t data,
    osireTempStatus_t * p_data )

```

OSP_OSIRE_SET_SETUP_SR command and reads status and temperature.

Writes the SETUP register. See READSETUP for the payload format. Additionally the STATUS and TEMP register is read.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress, 0..1000</i>	RGBi device address (0: broadcast)
<i>data</i>	SETUP register values
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

Definition at line 66 of file [osireDevice.c](#).

```

00069 {
00070     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00071     ospCmdBuffer_t ospCmd;
00072     enum OSP_ERROR_CODE ospErrorCode;
00073     errorSpi_t spiError;
00074
00075     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00076
00077     ospCmd.inCmdId = OSP_OSIRE_SET_SETUP_SR;
00078     ospCmd.inDeviceAddress = deviceAddress;
00079     ospCmd.p_inParameter = &data.data.setupData;
00080
00081     ospErrorCode = osp_cmd_buffer (&ospCmd);
00082     if (ospErrorCode != OSP_NO_ERROR)
00083     {
00084         return ospErrorCode;

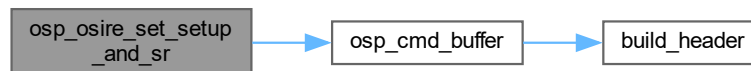
```

```

00085     }
00086
00087     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00088                                                         rspBuffer,
00089                                                         ospCmd.outCmdBufferLength,
00090                                                         ospCmd.outResponseLength);
00091
00092     if (spiError != NO_ERROR_SPI)
00093     {
00094         return OSP_ERROR_SPI;
00095     }
00096
00097     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00098     {
00099         return OSP_ERROR_CRC;
00100     }
00101
00102     for (uint8_t i = 0; i < 2; i++)
00103     {
00104         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00105     }
00106     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00107     return OSP_NO_ERROR;
00108 }

```

Here is the call graph for this function:



5.4.4.22 osp_read_otp()

```

enum OSP_ERROR_CODE osp_read_otp (
    uint16_t deviceAddress,
    uint8_t otpAddress,
    osireOtpData_t * p_data )

```

OSP_OSIRE_READ_OTP command Reads 8 bytes from OTP memory, from otpAddress. If readout address is beyond OTP address range, 0x00 will be delivered for these addresses.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>otpAddress</i>	address of the first read byte of OTP
<i>p_data</i>	OTP response data

Returns

error communication or command parameter error

Definition at line 236 of file `osireDevice.c`.

```

00238 {
00239     uint8_t rspBuffer[LENGTH_READ_OTP_RSP]; // response buffer
00240     ospCmdBuffer_t ospCmd;
00241     enum OSP_ERROR_CODE ospErrorCode;
00242     errorSpi_t spiError;
00243
00244     memset (rspBuffer, 0, LENGTH_READ_OTP_RSP);
00245
00246     ospCmd.inCmdId = OSP_OSIRE_READ_OTP;
00247     ospCmd.inDeviceAddress = deviceAddress;
00248     ospCmd.p_inParameter = &otpAddress;

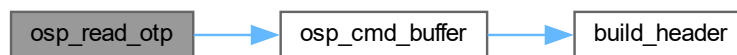
```

```

00249
00250     ospErrorCode = osp_cmd_buffer (&ospCmd);
00251     if (ospErrorCode != OSP_NO_ERROR)
00252     {
00253         return ospErrorCode;
00254     }
00255
00256     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00257                                                         rspBuffer,
00258                                                         ospCmd.outCmdBufferLength,
00259                                                         ospCmd.outResponseLength);
00260
00261     if (spiError != NO_ERROR_SPI)
00262     {
00263         return OSP_ERROR_SPI;
00264     }
00265
00266     if (crc (rspBuffer, LENGTH_READ_OTP_RSP) != 0)
00267     {
00268         return OSP_ERROR_CRC;
00269     }
00270
00271     for (uint8_t i = 0; i < 8; i++)
00272     {
00273         p_rsp->data.byte[i] = rspBuffer[10 - i];
00274     }
00275
00276     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00277     return OSP_NO_ERROR;
00278 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.5 osireDevice.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright 2022 by ams OSRAM AG
00003  * All rights are reserved.
00004  *
00005  * IMPORTANT - PLEASE READ CAREFULLY BEFORE COPYING, INSTALLING OR USING
00006  * THE SOFTWARE.
00007  *
00008  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00009  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00010  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
00011  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00012  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00013  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00014  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

```

```

00015 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY      *
00016 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT          *
00017 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE        *
00018 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.        *
00019 *****/
00020 #ifndef OSP_INC_OSIRE_DEVICE_H_
00021 #define OSP_INC_OSIRE_DEVICE_H_
00022
00023 #ifdef __cplusplus
00024 extern "C"
00025 {
00026 #endif
00027
00040 #include <Osp/inc/genericDevice.h>
00041
00042 /*****/
00043 /*****/
00044 #define LENGTH_SET_SETUP_MSG 5
00045 /*****/
00046 #define LENGTH_SET_PWM_MSG 10
00047 /*****/
00048 #define LENGTH_SET_OTTH_MSG 7
00049 /*****/
00050 #define LENGTH_READ_PWM_MSG 4
00051 #define LENGTH_READ_PWM_RSP 10
00052 /*****/
00053 #define LENGTH_READ_OTP_MSG 5
00054 #define LENGTH_READ_OTP_RSP 12
00055 /*****/
00056 #define LENGTH_READ_SETUP_MSG 4
00057 #define LENGTH_READ_SETUP_RSP 5
00058 /*****/
00059 #define LENGTH_READ_TEMPSTATUS_MSG 4
00060 #define LENGTH_READ_TEMPSTATUS_RSP 6
00061 /*****/
00062 #define LENGTH_READ_TEMP_MSG 4
00063 #define LENGTH_READ_TEMP_RSP 5
00064 /*****/
00065 #define LENGTH_READ_OTTH_MSG 4
00066 #define LENGTH_READ_OTTH_RSP 7
00067 /*****/
00068 #define LENGTH_READ_COMSTATUS_MSG 4
00069 #define LENGTH_READ_COMSTATUS_RSP 5
00070 /*****/
00071 #define LENGTH_READ_STATUS_MSG 4
00072 #define LENGTH_READ_STATUS_RSP 5
00073 /*****/
00074 #define LENGTH_READ_LEDSTATUS_MSG 4
00075 #define LENGTH_READ_LEDSTATUS_RSP 5
00076
00077 /*****/
00078 #define LENGTH_P4ERROR_MSG 4
00079
00080 /*****/
00081 /*****/
00085 enum OSP_OSIRE_DEVICE_CMDS
00086 {
00087     OSP_OSIRE_P4ERROR_BIDIR = 0x08,
00088     OSP_OSIRE_CLR_ERROR_SR = 0x21,
00089     OSP_OSIRE_GO_SLEEP_SR = 0x24,
00090     OSP_OSIRE_GO_ACTIVE_SR = 0x25,
00091     OSP_OSIRE_GO_DEEP_SLEEP_SR = 0x26,
00092     OSP_OSIRE_READ_STATUS = 0x40,
00093     OSP_OSIRE_READ_TEMP_STATUS = 0x42,
00094     OSP_OSIRE_READ_COM_STATUS = 0x44,
00095     OSP_OSIRE_READ_LED_STATUS = 0x46,
00096     OSP_OSIRE_READ_TEMP = 0x48,
00097     OSP_OSIRE_READ_OTTH = 0x4A,
00098     OSP_OSIRE_SET_OTTH = 0x4B,
00099     OSP_OSIRE_SET_OTTH_SR = 0x6B,
00100     OSP_OSIRE_READ_SETUP = 0x4C,
00101     OSP_OSIRE_SET_SETUP = 0x4D,
00102     OSP_OSIRE_SET_SETUP_SR = 0x6D,
00103     OSP_OSIRE_READ_PWM = 0x4E,
00104     OSP_OSIRE_SET_PWM = 0x4F,
00105     OSP_OSIRE_SET_PWM_SR = 0x6F,
00106     OSP_OSIRE_READ_OTP = 0x58,
00108     OSP_OSIRE_P4ERROR_LOOP = 0x09
00109 };
00110
00111 /*****/
00112 /*****/
00113 typedef struct SetSetupData_t
00114 {
00115     union
00116     {
00117         uint8_t setupData;

```



```

00118     struct
00119     {
00120         uint8_t uv_fsave :1;
00121         uint8_t ot_fsave :1;
00122         uint8_t los_fsave :1;
00123         uint8_t ce_fsave :1;
00124         uint8_t tempck_sel :1;
00125         uint8_t crc_en :1;
00126         uint8_t com_inv :1;
00127         uint8_t fast_pwm :1;
00128     } bit;
00129     } data;
00130     uint16_t address :10;
00131 } osireSetSetupData_t;
00132 /*****
00133 /*****
00134 /*****
00135 typedef struct PwmData_t
00136 {
00137     union
00138     {
00139         uint8_t pwmData[6];
00140         struct
00141         {
00142             uint16_t blue_pwm :15;
00143             uint16_t blue_curr :1;
00144             uint16_t green_pwm :15;
00145             uint16_t green_curr :1;
00146             uint16_t red_pwm :15;
00147             uint16_t red_curr :1;
00148         } bit;
00149     } data;
00150     uint16_t address :10;
00151 } osirePwmData_t;
00152
00153 /*****
00154 /*****
00155 typedef struct OtpData_t
00156 {
00157     union
00158     {
00159         uint8_t byte[8];
00160     } data;
00161     uint16_t address :10;
00162 } osireOtpData_t;
00163 /*****
00164 /*****
00165 /*****
00166 typedef struct OtpDataComplete_t
00167 {
00168     union
00169     {
00170         uint8_t byte[32];
00171     } data;
00172     uint16_t address :10;
00173 } osireOtpDataComplete_t;
00174
00175 /*****
00176 /*****
00177 typedef struct LedStatus_t
00178 {
00179     union
00180     {
00181         uint8_t ledStatus;
00182         struct
00183         {
00184             uint8_t blue_short :1;
00185             uint8_t green_short :1;
00186             uint8_t red_short :1;
00187             uint8_t ledStatus_reserved_1 :1;
00188             uint8_t blue_open :1;
00189             uint8_t green_open :1;
00190             uint8_t red_open :1;
00191             uint8_t ledStatus_reserved_2 :1;
00192         } bit;
00193     } data;
00194     uint16_t address :10;
00195 } osireLedStatus_t;
00196 /*****
00197 /*****
00198 /*****
00199 typedef struct osireTemp_t
00200 {
00201     union
00202     {
00203         uint8_t temp_value;
00204     } data;
00205     uint16_t address :10;
00206 } osireTemp_t;
00207 /*****

```

```

00209 /*****/
00210 typedef struct ComStatus_t
00211 {
00212     union
00213     {
00214         uint8_t comStatus;
00215         struct
00216         {
00217             uint8_t sio1_state :2;
00218             uint8_t sio2_state :2;
00219             uint8_t reserved :4;
00220         } bit;
00221     } data;
00222     uint16_t address :10;
00223 } osireComStatus_t;
00225 /*****/
00226 /*****/
00227 typedef struct Status_t
00228 {
00229     union
00230     {
00231         uint8_t status;
00232         struct
00233         {
00234             uint8_t uv_flag :1;
00235             uint8_t ot_flag :1;
00236             uint8_t los_flag :1;
00237             uint8_t ce_flag :1;
00238             uint8_t com_mode :1;
00239             uint8_t otpcrc_flag :1;
00240             uint8_t state :2;
00241         } bit;
00242     } data;
00243     uint16_t address :10;
00244 } osireStatus_t;
00246 /*****/
00247 /*****/
00248 typedef struct TempStatus_t
00249 {
00250     union
00251     {
00252         uint8_t tempStatus[2];
00253         struct
00254         {
00255             uint8_t Status;
00256             uint8_t Temp;
00257         } byte;
00258     } data;
00259     uint16_t address :10;
00260 } osireTempStatus_t;
00261 /*****/
00262 /*****/
00263 typedef struct OtthData_t
00264 {
00265     union
00266     {
00267         uint8_t otthData[3];
00268         struct
00269         {
00270             uint8_t ot_high_value :8;
00271             uint8_t ot_low_value :8;
00272             uint8_t or_cycle :2;
00273             uint8_t otth_reserved :6;
00274         } bit;
00275     } data;
00276     uint16_t address :10;
00277 } osireOtthData_t;
00278 /*****/
00279 /*****/
00280
00293 enum OSP_ERROR_CODE osp_osire_set_setup (uint16_t deviceAddress,
00294                                         osireSetSetupData_t data);
00295
00310 enum OSP_ERROR_CODE osp_osire_set_setup_and_sr (uint16_t deviceAddress,
00311                                                  osireSetSetupData_t data,
00312                                                  osireTempStatus_t *p_data);
00313
00326 enum OSP_ERROR_CODE osp_osire_set_pwm (uint16_t deviceAddress,
00327                                         osirePwmData_t data);
00328
00343 enum OSP_ERROR_CODE osp_osire_set_pwm_and_sr (uint16_t deviceAddress,
00344                                                osirePwmData_t data,
00345                                                osireTempStatus_t *p_data);
00346
00359 enum OSP_ERROR_CODE osp_osire_read_pwm (uint16_t deviceAddress,
00360                                         osirePwmData_t *p_data);
00361

```

```

00376 enum OSP_ERROR_CODE osp_read_otp (uint16_t deviceAddress, uint8_t otpAddress,
00377                                     osireOtpData_t *p_data);
00378
00391 enum OSP_ERROR_CODE osp_osire_read_otp_complete (uint16_t deviceAddress,
00392                                                    osireOtpDataComplete_t *p_data);
00393
00405 enum OSP_ERROR_CODE osp_osire_read_ledstatus (uint16_t deviceAddress,
00406                                                 osireLedStatus_t *p_data);
00407
00420 enum OSP_ERROR_CODE osp_osire_read_tempstatus (uint16_t deviceAddress,
00421                                                  osireTempStatus_t *p_data);
00422
00434 enum OSP_ERROR_CODE osp_osire_read_temp (uint16_t deviceAddress,
00435                                             osireTemp_t *p_data);
00436
00449 enum OSP_ERROR_CODE osp_osire_set_otth (uint16_t deviceAddress,
00450                                           osireOtthData_t data);
00451
00466 enum OSP_ERROR_CODE osp_osire_set_otth_and_sr (uint16_t deviceAddress,
00467                                                  osireOtthData_t data,
00468                                                  osireTempStatus_t *p_data);
00469
00482 enum OSP_ERROR_CODE osp_osire_read_otth (uint16_t deviceAddress,
00483                                            osireOtthData_t *p_data);
00484
00498 enum OSP_ERROR_CODE osp_osire_go_sleep_and_sr (uint16_t deviceAddress,
00499                                                  osireTempStatus_t *p_data);
00500
00514 enum OSP_ERROR_CODE osp_osire_go_deep_sleep_and_sr (uint16_t deviceAddress,
00515                                                       osireTempStatus_t *p_data);
00516
00527 enum OSP_ERROR_CODE osp_osire_clr_error_and_sr (uint16_t deviceAddress,
00528                                                  osireTempStatus_t *p_data);
00529
00547 enum OSP_ERROR_CODE osp_osire_p4error_bidir (uint16_t deviceAddress,
00548                                                  osireTempStatus_t *p_data);
00549
00567 enum OSP_ERROR_CODE osp_osire_p4error_loop (uint16_t deviceAddress,
00568                                                osireTempStatus_t *p_data);
00569
00584 enum OSP_ERROR_CODE osp_go_active_and_sr (uint16_t deviceAddress,
00585                                              osireTempStatus_t *p_rsp);
00586
00597 enum OSP_ERROR_CODE osp_osire_clr_error_and_sr (uint16_t deviceAddress,
00598                                                  osireTempStatus_t *p_data);
00599
00617 enum OSP_ERROR_CODE osp_osire_p4error_bidir (uint16_t deviceAddress,
00618                                                  osireTempStatus_t *p_data);
00619
00631 enum OSP_ERROR_CODE osp_osire_read_setup (uint16_t deviceAddress,
00632                                             osireSetSetupData_t *p_data);
00633
00645 enum OSP_ERROR_CODE osp_osire_read_comstatus (uint16_t deviceAddress,
00646                                                 osireComStatus_t *p_data);
00647
00659 enum OSP_ERROR_CODE osp_osire_read_status (uint16_t deviceAddress,
00660                                              osireStatus_t *p_data);
00661
00662 #ifdef __cplusplus
00663 }
00664 #endif
00665
00666 #endif // OSP_INC_OSIRE_DEVICE_H_

```

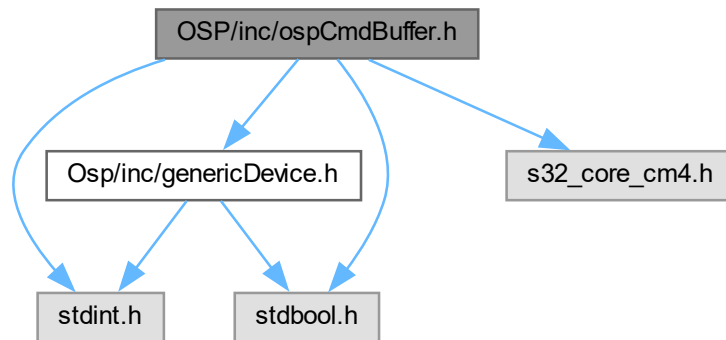
5.6 OSP/inc/ospCmdBuffer.h File Reference

```

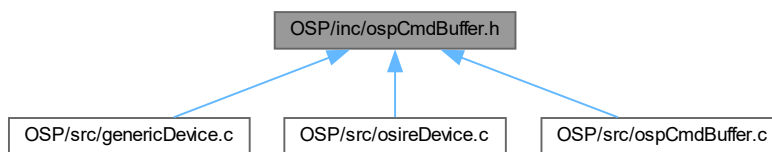
#include <stdint.h>
#include <stdbool.h>
#include <Osp/inc/genericDevice.h>
#include <s32_core_cm4.h>

```

Include dependency graph for ospCmdBuffer.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `ospCmd_t`

Typedefs

- typedef struct `ospCmd_t` `ospCmdBuffer_t`

5.6.1 Typedef Documentation

5.6.1.1 ospCmdBuffer_t

```
typedef struct ospCmd_t ospCmdBuffer_t
```

5.7 ospCmdBuffer.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright 2022 by ams OSRAM AG
00003  * All rights are reserved.
00004  *
00005  * IMPORTANT - PLEASE READ CAREFULLY BEFORE COPYING, INSTALLING OR USING
00006  * THE SOFTWARE.
  
```

```

00007 *
00008 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS *
00009 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT *
00010 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS *
00011 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT *
00012 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
00013 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT *
00014 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, *
00015 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY *
00016 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT *
00017 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE *
00018 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. *
00019 *****/
00020 #ifndef OSP_INC_OSPCMDBUFFER_H_
00021 #define OSP_INC_OSPCMDBUFFER_H_
00022
00023 #include <stdint.h>
00024 #include <stdbool.h>
00025 #include <Osp/inc/genericDevice.h>
00026 #include <s32_core_cm4.h>
00027
00028 /*****
00029 *****/
00030
00031 /*****
00032 *****/
00033 typedef struct ospCmd_t
00034 {
00035     uint16_t inDeviceAddress;
00036     uint8_t inCmdId;
00037     void *p_inParameter;
00038     uint8_t *p_outCmdBuffer;
00039     uint8_t outCmdBufferLength;
00040     uint8_t outResponseLength;
00041     bool outResponseMsg;
00042 } ospCmdBuffer_t;
00043
00044 /*****
00045 *****/
00046 START_FUNCTION_DECLARATION_RAMSECTION
00047 enum OSP_ERROR_CODE osp_cmd_buffer (ospCmdBuffer_t *p_cmdInfo)
00048 END_FUNCTION_DECLARATION_RAMSECTION
00049
00050 #endif /* OSP_INC_OSPCMDBUFFER_H_ */

```

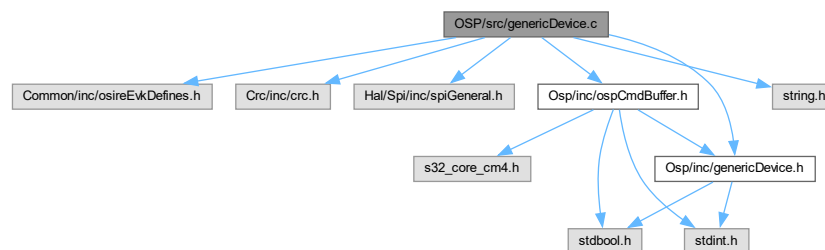
5.8 OSP/src/genericDevice.c File Reference

```

#include <Common/inc/osireEvkDefines.h>
#include <Crc/inc/crc.h>
#include <Hal/Spi/inc/spiGeneral.h>
#include <Osp/inc/genericDevice.h>
#include <Osp/inc/ospCmdBuffer.h>
#include <string.h>

```

Include dependency graph for genericDevice.c:



Functions

- enum [OSP_ERROR_CODE](#) [osp_init_bidir](#) (uint16_t deviceAddress, [ospInitRsp_t](#) *p_rsp)

OSP_INIT_BIDIR command Initiates the automatic addressing of the chain and sets the communication direction to bidirectional. The command shall be addressed to the first unit in the chain always with address 0x001. The last unit

- in the chain (indicated by the EOL mode) returns its address to the master.
- enum [OSP_ERROR_CODE osp_init_loop](#) (uint16_t deviceAddress, [ospInitRsp_t](#) *p_rsp)
OSP_INIT_LOOP command Same as INITBIDIR but sets the communication mode to loop-back. The response to the master is sent in the forward direction.
 - enum [OSP_ERROR_CODE osp_reset](#) (uint16_t deviceAddress)
OSP_RESET command Performs a complete reset of one or all devices. The effect is identical to a power cycle. All register values are set to their default values, all error flags are cleared, the communication mode detection is restarted, LED drivers are turned off, and the address is set to 0x3ff. The device enters the UNINITIALIZED mode.
 - enum [OSP_ERROR_CODE osp_go_active](#) (uint16_t deviceAddress)
OSP_GO_ACTIVE command. Puts device into ACTIVE state: The LED drivers are enabled. The last PWM parameters are used to create LED PWM signals. An PWM parameter update via SETPWM() command is possible. During ACTIVE mode, diagnostic is possible and readable.
 - enum [OSP_ERROR_CODE osp_go_sleep](#) (uint16_t deviceAddress)
OSP_GO_SLEEP command Sends one or all devices into SLEEP state.
 - enum [OSP_ERROR_CODE osp_go_deep_sleep](#) (uint16_t deviceAddress)
OSP_GO_DEEP_SLEEP command Sends one or all devices into DEEPSLEEP state.
 - enum [OSP_ERROR_CODE osp_osire_clr_error](#) (uint16_t deviceAddress)
OSP_CLEAR_ERROR command This function will clear all error flags, if an error still exists for example short/open the error flag is set again.
 - void [build_header](#) (uint8_t *p_msg, uint16_t deviceAddress, uint8_t command, uint8_t lengthMsg)
Internal function for OSP header creation.

5.8.1 Function Documentation

5.8.1.1 build_header()

```
void build_header (
    uint8_t * p_msg,
    uint16_t deviceAddress,
    uint8_t command,
    uint8_t lengthMsg )
```

Internal function for OSP header creation.

Parameters

<i>p_msg</i>	message buffer to create the OSP header
<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
<i>command</i>	OSP or OSP_OSIRE command
<i>lengthMsg</i>	length of the buffer that is used lengthMsg

Returns

error communication or command parameter error

Definition at line 269 of file [genericDevice.c](#).

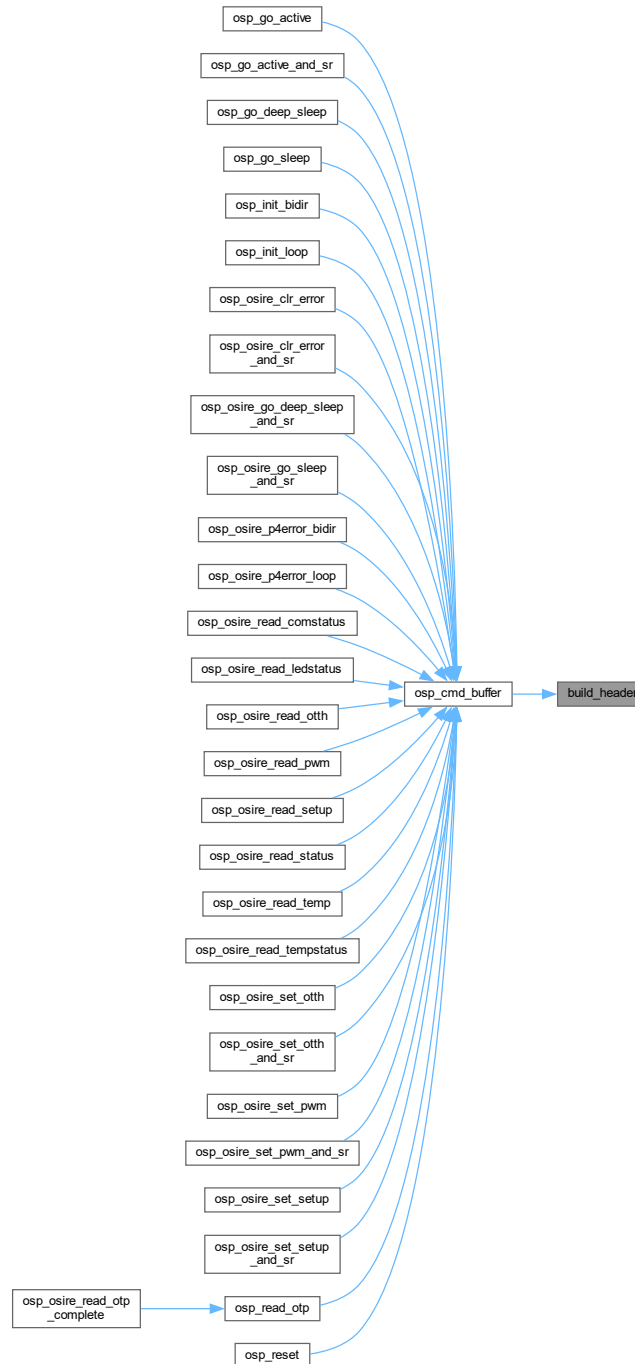
```
00271 {
00272     ospHeader\_t hdr;
00273
00274     hdr.bit.preamble = OSP\_PROTOCOL\_PREAMBLE;
00275     hdr.bit.address = deviceAddress;
00276
00277     if (lengthMsg == 12)
00278     {
00279         hdr.bit.psi = 7;
00280     }
00281     else
00282     {
00283         hdr.bit.psi = lengthMsg - 4;
00284     }
```

```

00285     hdr.bit.command = command;
00286
00287     for (uint8_t i = 0; i < 3; i++)
00288     {
00289         *p_msg = hdr.buf[3 - i];
00290         p_msg++;
00291     }
00292 }

```

Here is the caller graph for this function:



5.8.1.2 osp_go_active()

```
enum OSP_ERROR_CODE osp_go_active (
    uint16_t deviceAddress )
```

OSP_GO_ACTIVE command. Puts device into ACTIVE state: The LED drivers are enabled. The last PWM parameters are used to create LED PWM signals. An PWM parameter update via SETPWM() command is possible. During ACTIVE mode, diagnostic is possible and readable.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
----------------------	--

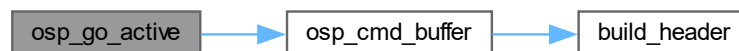
Returns

error communication or command parameter error

Definition at line 150 of file [genericDevice.c](#).

```
00151 {
00152     ospCmdBuffer_t ospCmd;
00153     enum OSP_ERROR_CODE ospErrorCode;
00154     errorSpi_t spiError;
00155
00156     ospCmd.inCmdId = OSP_GO_ACTIVE;
00157     ospCmd.inDeviceAddress = deviceAddress;
00158     ospCmd.p_inParameter = NULL;
00159
00160     ospErrorCode = osp_cmd_buffer (&ospCmd);
00161     if (ospErrorCode != OSP_NO_ERROR)
00162     {
00163         return ospErrorCode;
00164     }
00165
00166     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00167                                           ospCmd.outCmdBufferLength);
00168
00169     if (spiError != NO_ERROR_SPI)
00170     {
00171         return OSP_ERROR_SPI;
00172     }
00173
00174     return OSP_NO_ERROR;
00175 }
```

Here is the call graph for this function:



5.8.1.3 osp_go_deep_sleep()

```
enum OSP_ERROR_CODE osp_go_deep_sleep (
    uint16_t deviceAddress )
```

OSP_GO_DEEP_SLEEP command Sends one or all devices into DEEPSLEEP state.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
----------------------	--

Returns

error communication or command parameter error

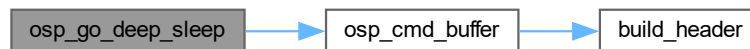
Definition at line 209 of file [genericDevice.c](#).

```

00210 {
00211     ospCmdBuffer_t ospCmd;
00212     enum OSP_ERROR_CODE ospErrorCode;
00213     errorSpi_t spiError;
00214
00215     ospCmd.inCmdId = OSP_GO_DEEP_SLEEP;
00216     ospCmd.inDeviceAddress = deviceAddress;
00217     ospCmd.p_inParameter = NULL;
00218
00219     ospErrorCode = osp_cmd_buffer (&ospCmd);
00220     if (ospErrorCode != OSP_NO_ERROR)
00221     {
00222         return ospErrorCode;
00223     }
00224
00225     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00226                                           ospCmd.outCmdBufferLength);
00227
00228     if (spiError != NO_ERROR_SPI)
00229     {
00230         return OSP_ERROR_SPI;
00231     }
00232
00233     return OSP_NO_ERROR;
00234 }
00235 }

```

Here is the call graph for this function:

**5.8.1.4 osp_go_sleep()**

```

enum OSP_ERROR_CODE osp_go_sleep (
    uint16_t deviceAddress )

```

OSP_GO_SLEEP command Sends one or all devices into SLEEP state.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
----------------------	--

Returns

error communication or command parameter error

Definition at line 179 of file [genericDevice.c](#).

```

00180 {
00181     ospCmdBuffer_t ospCmd;
00182     enum OSP_ERROR_CODE ospErrorCode;
00183     errorSpi_t spiError;
00184
00185     ospCmd.inCmdId = OSP_GO_SLEEP;
00186     ospCmd.inDeviceAddress = deviceAddress;
00187     ospCmd.p_inParameter = NULL;
00188
00189     ospErrorCode = osp_cmd_buffer (&ospCmd);
00190     if (ospErrorCode != OSP_NO_ERROR)
00191     {

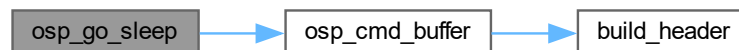
```

```

00192     return ospErrorCode;
00193 }
00194
00195 spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00196                                         ospCmd.outCmdBufferLength);
00197
00198 if (spiError != NO_ERROR_SPI)
00199 {
00200     return OSP_ERROR_SPI;
00201 }
00202
00203 return OSP_NO_ERROR;
00204 }
00205 }

```

Here is the call graph for this function:



5.8.1.5 osp_init_bidir()

```

enum OSP_ERROR_CODE osp_init_bidir (
    uint16_t deviceAddress,
    ospInitRsp_t * p_rsp )

```

OSP_INIT_BIDIR command Initiates the automatic addressing of the chain and sets the communication direction to bidirectional. The command shall be addressed to the first unit in the chain always with address 0x001. The last unit in the chain (indicated by the EOL mode) returns its address to the master.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	start address of 1st device, shall be 1
<i>p_rsp</i>	response data from device

Returns

error communication or command parameter error

Definition at line 30 of file [genericDevice.c](#).

```

00031 {
00032     uint8_t rspBuffer[LENGTH_INIT_RSP]; // response buffer
00033     ospCmdBuffer_t ospCmd;
00034     enum OSP_ERROR_CODE ospErrorCode;
00035     errorSpi_t spiError;
00036
00037     // clear response buffer
00038     memset (rspBuffer, 0, LENGTH_INIT_RSP);
00039
00040     ospCmd.inCmdId = OSP_INIT_BIDIR;
00041     ospCmd.inDeviceAddress = deviceAddress;
00042     ospCmd.p_inParameter = NULL;
00043
00044     ospErrorCode = osp_cmd_buffer (&ospCmd);
00045     if (ospErrorCode != OSP_NO_ERROR)
00046     {
00047         return ospErrorCode;
00048     }
00049
00050     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00051                                                         rspBuffer,
00052                                                         ospCmd.outCmdBufferLength,
00053                                                         ospCmd.outResponseLength);

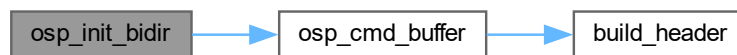
```

```

00054
00055     if (spiError != NO_ERROR_SPI)
00056     {
00057         return OSP_ERROR_SPI;
00058     }
00059
00060     if (crc (rspBuffer, LENGTH_INIT_RSP) != 0)
00061     {
00062         return OSP_ERROR_CRC;
00063     }
00064
00065     p_rsp->data.bit.temp = rspBuffer[3];
00066     p_rsp->data.bit.status = rspBuffer[4];
00067     p_rsp->data.bit.address = ((rspBuffer[0] & 0x0F) << 6)
00068         | ((rspBuffer[1] >> 2) & 0x3F);
00069
00070     p_rsp->address = p_rsp->data.bit.address; // return address for all cmds with rsp
00071     return OSP_NO_ERROR;
00072 }

```

Here is the call graph for this function:



5.8.1.6 osp_init_loop()

```

enum OSP_ERROR_CODE osp_init_loop (
    uint16_t deviceAddress,
    ospInitRsp_t * p_rsp )

```

OSP_INIT_LOOP command Same as INITBIDIR but sets the communication mode to loop-back. The response to the master is sent in the forward direction.

For further details refer to "OSIRE_E3731i_Start_Up_Guide"

Parameters

<i>deviceAddress</i>	start address of 1st device, shall be 1
<i>p_rsp</i>	response data from device

Returns

error communication or command parameter error

Definition at line 76 of file [genericDevice.c](#).

```

00077 {
00078     uint8_t rspBuffer[LENGTH_INIT_RSP]; // response buffer
00079     ospCmdBuffer_t ospCmd;
00080     enum OSP_ERROR_CODE ospErrorCode;
00081     errorSpi_t spiError;
00082
00083     memset (rspBuffer, 0, LENGTH_INIT_RSP);
00084
00085     ospCmd.inCmdId = OSP_INIT_LOOP;
00086     ospCmd.inDeviceAddress = deviceAddress;
00087     ospCmd.p_inParameter = NULL;
00088
00089     ospErrorCode = osp_cmd_buffer (&ospCmd);
00090     if (ospErrorCode != OSP_NO_ERROR)
00091     {
00092         return ospErrorCode;
00093     }
00094
00095     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,

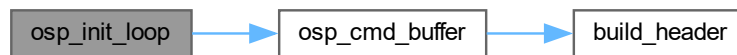
```

```

00096                                     rspBuffer,
00097                                     ospCmd.outCmdBufferLength,
00098                                     ospCmd.outResponseLength);
00099
00100     if (spiError != NO_ERROR_SPI)
00101     {
00102         return OSP_ERROR_SPI;
00103     }
00104
00105     if (crc (rspBuffer, LENGTH_INIT_RSP) != 0)
00106     {
00107         return OSP_ERROR_CRC;
00108     }
00109
00110     p_rsp->data.bit.temp = rspBuffer[3];
00111     p_rsp->data.bit.status = rspBuffer[4];
00112     p_rsp->data.bit.address = ((rspBuffer[0] & 0x0F) << 6)
00113         | ((rspBuffer[1] >> 2) & 0x3F);
00114
00115     p_rsp->address = p_rsp->data.bit.address; // return address for all cmds with rsp
00116     return OSP_NO_ERROR;
00117 }

```

Here is the call graph for this function:



5.8.1.7 osp_osire_clr_error()

```

enum OSP_ERROR_CODE osp_osire_clr_error (
    uint16_t deviceAddress )

```

OSP_CLEAR_ERROR command This function will clear all error flags, if an error still exists for example short/open the error flag is set again.

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
----------------------	--

Returns

error communication or command parameter error

Definition at line 239 of file [genericDevice.c](#).

```

00240 {
00241     ospCmdBuffer_t ospCmd;
00242     enum OSP_ERROR_CODE ospErrorCode;
00243     errorSpi_t spiError;
00244
00245     ospCmd.inCmdId = OSP_CLR_ERROR;
00246     ospCmd.inDeviceAddress = deviceAddress;
00247     ospCmd.p_inParameter = NULL;
00248
00249     ospErrorCode = osp_cmd_buffer (&ospCmd);
00250     if (ospErrorCode != OSP_NO_ERROR)
00251     {
00252         return ospErrorCode;
00253     }
00254
00255     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00256                                           ospCmd.outCmdBufferLength);
00257
00258     if (spiError != NO_ERROR_SPI)
00259     {
00260         return OSP_ERROR_SPI;

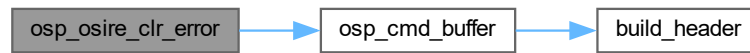
```

```

00261     }
00262
00263     return OSP_NO_ERROR;
00264
00265 }

```

Here is the call graph for this function:



5.8.1.8 osp_reset()

```

enum OSP_ERROR_CODE osp_reset (
    uint16_t deviceAddress )

```

OSP_RESET command Performs a complete reset of one or all devices. The effect is identical to a power cycle. All register values are set to their default values, all error flags are cleared, the communication mode detection is restarted, LED drivers are turned off, and the address is set to 0x3ff. The device enters the UNINITIALIZED mode. For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceId</i>	0..1000 RGBi device address (0: broadcast)
-----------------	--

Returns

error communication or command parameter error

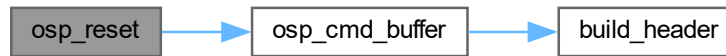
Definition at line 121 of file [genericDevice.c](#).

```

00122 {
00123     ospCmdBuffer_t ospCmd;
00124     enum OSP_ERROR_CODE ospErrorCode;
00125     errorSpi_t spiError;
00126
00127     ospCmd.inCmdId = OSP_RESET;
00128     ospCmd.inDeviceAddress = deviceAddress;
00129     ospCmd.p_inParameter = NULL;
00130
00131     ospErrorCode = osp_cmd_buffer (&ospCmd);
00132     if (ospErrorCode != OSP_NO_ERROR)
00133     {
00134         return ospErrorCode;
00135     }
00136
00137     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00138                                           ospCmd.outCmdBufferLength);
00139
00140     if (spiError != NO_ERROR_SPI)
00141     {
00142         return OSP_ERROR_SPI;
00143     }
00144
00145     return OSP_NO_ERROR;
00146 }

```

Here is the call graph for this function:



5.9 genericDevice.c

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright 2022 by ams OSRAM AG
00003  * All rights are reserved.
00004  *
00005  * IMPORTANT - PLEASE READ CAREFULLY BEFORE COPYING, INSTALLING OR USING
00006  * THE SOFTWARE.
00007  *
00008  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00009  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00010  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
00011  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00012  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00013  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00014  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00015  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00016  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00017  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00018  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00019  *****/
00020
00021 #include <Common/inc/osireEvkDefines.h>
00022 #include <Crc/inc/crc.h>
00023 #include <Hal/Spi/inc/spiGeneral.h>
00024 #include <Osp/inc/genericDevice.h>
00025 #include <Osp/inc/ospCmdBuffer.h>
00026 #include <string.h>
00027
00028 /*****
00029  *****/
00030 enum OSP_ERROR_CODE osp_init_bidir (uint16_t deviceAddress, ospInitRsp_t *p_rsp)
00031 {
00032     uint8_t rspBuffer[LENGTH_INIT_RSP]; // response buffer
00033     ospCmdBuffer_t ospCmd;
00034     enum OSP_ERROR_CODE ospErrorCode;
00035     errorSpi_t spiError;
00036
00037     // clear response buffer
00038     memset (rspBuffer, 0, LENGTH_INIT_RSP);
00039
00040     ospCmd.inCmdId = OSP_INIT_BIDIR;
00041     ospCmd.inDeviceAddress = deviceAddress;
00042     ospCmd.p_inParameter = NULL;
00043
00044     ospErrorCode = osp_cmd_buffer (&ospCmd);
00045     if (ospErrorCode != OSP_NO_ERROR)
00046     {
00047         return ospErrorCode;
00048     }
00049
00050     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00051                                                         rspBuffer,
00052                                                         ospCmd.outCmdBufferLength,
00053                                                         ospCmd.outResponseLength);
00054
00055     if (spiError != NO_ERROR_SPI)
00056     {
00057         return OSP_ERROR_SPI;
00058     }
00059
00060     if (crc (rspBuffer, LENGTH_INIT_RSP) != 0)
00061     {
00062         return OSP_ERROR_CRC;
00063     }

```

```

00064
00065 p_rsp->data.bit.temp = rspBuffer[3];
00066 p_rsp->data.bit.status = rspBuffer[4];
00067 p_rsp->data.bit.address = ((rspBuffer[0] & 0x0F) << 6)
00068 | ((rspBuffer[1] >> 2) & 0x3F);
00069
00070 p_rsp->address = p_rsp->data.bit.address; // return address for all cmds with rsp
00071 return OSP_NO_ERROR;
00072 }
00073
00074 /*****
00075 /*****
00076 enum OSP_ERROR_CODE osp_init_loop (uint16_t deviceAddress, ospInitRsp_t *p_rsp)
00077 {
00078     uint8_t rspBuffer[LENGTH_INIT_RSP]; // response buffer
00079     ospCmdBuffer_t ospCmd;
00080     enum OSP_ERROR_CODE ospErrorCode;
00081     errorSpi_t spiError;
00082
00083     memset (rspBuffer, 0, LENGTH_INIT_RSP);
00084
00085     ospCmd.inCmdId = OSP_INIT_LOOP;
00086     ospCmd.inDeviceAddress = deviceAddress;
00087     ospCmd.p_inParameter = NULL;
00088
00089     ospErrorCode = osp_cmd_buffer (&ospCmd);
00090     if (ospErrorCode != OSP_NO_ERROR)
00091     {
00092         return ospErrorCode;
00093     }
00094
00095     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00096                                                         rspBuffer,
00097                                                         ospCmd.outCmdBufferLength,
00098                                                         ospCmd.outResponseLength);
00099
00100     if (spiError != NO_ERROR_SPI)
00101     {
00102         return OSP_ERROR_SPI;
00103     }
00104
00105     if (crc (rspBuffer, LENGTH_INIT_RSP) != 0)
00106     {
00107         return OSP_ERROR_CRC;
00108     }
00109
00110     p_rsp->data.bit.temp = rspBuffer[3];
00111     p_rsp->data.bit.status = rspBuffer[4];
00112     p_rsp->data.bit.address = ((rspBuffer[0] & 0x0F) << 6)
00113 | ((rspBuffer[1] >> 2) & 0x3F);
00114
00115     p_rsp->address = p_rsp->data.bit.address; // return address for all cmds with rsp
00116     return OSP_NO_ERROR;
00117 }
00118
00119 /*****
00120 /*****
00121 enum OSP_ERROR_CODE osp_reset (uint16_t deviceAddress)
00122 {
00123     ospCmdBuffer_t ospCmd;
00124     enum OSP_ERROR_CODE ospErrorCode;
00125     errorSpi_t spiError;
00126
00127     ospCmd.inCmdId = OSP_RESET;
00128     ospCmd.inDeviceAddress = deviceAddress;
00129     ospCmd.p_inParameter = NULL;
00130
00131     ospErrorCode = osp_cmd_buffer (&ospCmd);
00132     if (ospErrorCode != OSP_NO_ERROR)
00133     {
00134         return ospErrorCode;
00135     }
00136
00137     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00138                                             ospCmd.outCmdBufferLength);
00139
00140     if (spiError != NO_ERROR_SPI)
00141     {
00142         return OSP_ERROR_SPI;
00143     }
00144
00145     return OSP_NO_ERROR;
00146 }
00147
00148 /*****
00149 /*****
00150 enum OSP_ERROR_CODE osp_go_active (uint16_t deviceAddress)

```

```

00151 {
00152     ospCmdBuffer_t ospCmd;
00153     enum OSP_ERROR_CODE ospErrorCode;
00154     errorSpi_t spiError;
00155
00156     ospCmd.inCmdId = OSP_GO_ACTIVE;
00157     ospCmd.inDeviceAddress = deviceAddress;
00158     ospCmd.p_inParameter = NULL;
00159
00160     ospErrorCode = osp_cmd_buffer (&ospCmd);
00161     if (ospErrorCode != OSP_NO_ERROR)
00162     {
00163         return ospErrorCode;
00164     }
00165
00166     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00167                                           ospCmd.outCmdBufferLength);
00168
00169     if (spiError != NO_ERROR_SPI)
00170     {
00171         return OSP_ERROR_SPI;
00172     }
00173
00174     return OSP_NO_ERROR;
00175 }
00176
00177 /*****
00178 /*****
00179 enum OSP_ERROR_CODE osp_go_sleep (uint16_t deviceAddress)
00180 {
00181     ospCmdBuffer_t ospCmd;
00182     enum OSP_ERROR_CODE ospErrorCode;
00183     errorSpi_t spiError;
00184
00185     ospCmd.inCmdId = OSP_GO_SLEEP;
00186     ospCmd.inDeviceAddress = deviceAddress;
00187     ospCmd.p_inParameter = NULL;
00188
00189     ospErrorCode = osp_cmd_buffer (&ospCmd);
00190     if (ospErrorCode != OSP_NO_ERROR)
00191     {
00192         return ospErrorCode;
00193     }
00194
00195     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00196                                           ospCmd.outCmdBufferLength);
00197
00198     if (spiError != NO_ERROR_SPI)
00199     {
00200         return OSP_ERROR_SPI;
00201     }
00202
00203     return OSP_NO_ERROR;
00204 }
00205
00206 /*****
00207 /*****
00209 enum OSP_ERROR_CODE osp_go_deep_sleep (uint16_t deviceAddress)
00210 {
00211     ospCmdBuffer_t ospCmd;
00212     enum OSP_ERROR_CODE ospErrorCode;
00213     errorSpi_t spiError;
00214
00215     ospCmd.inCmdId = OSP_GO_DEEP_SLEEP;
00216     ospCmd.inDeviceAddress = deviceAddress;
00217     ospCmd.p_inParameter = NULL;
00218
00219     ospErrorCode = osp_cmd_buffer (&ospCmd);
00220     if (ospErrorCode != OSP_NO_ERROR)
00221     {
00222         return ospErrorCode;
00223     }
00224
00225     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00226                                           ospCmd.outCmdBufferLength);
00227
00228     if (spiError != NO_ERROR_SPI)
00229     {
00230         return OSP_ERROR_SPI;
00231     }
00232
00233     return OSP_NO_ERROR;
00234 }
00235
00236 /*****
00237

```



```

00238 /*****/
00239 enum OSP_ERROR_CODE osp_osire_clr_error (uint16_t deviceAddress)
00240 {
00241     ospCmdBuffer_t ospCmd;
00242     enum OSP_ERROR_CODE ospErrorCode;
00243     errorSpi_t spiError;
00244
00245     ospCmd.inCmdId = OSP_CLR_ERROR;
00246     ospCmd.inDeviceAddress = deviceAddress;
00247     ospCmd.p_inParameter = NULL;
00248
00249     ospErrorCode = osp_cmd_buffer (&ospCmd);
00250     if (ospErrorCode != OSP_NO_ERROR)
00251     {
00252         return ospErrorCode;
00253     }
00254
00255     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00256                                           ospCmd.outCmdBufferLength);
00257
00258     if (spiError != NO_ERROR_SPI)
00259     {
00260         return OSP_ERROR_SPI;
00261     }
00262
00263     return OSP_NO_ERROR;
00264 }
00265
00266 /*****/
00267 /*****/
00268 void build_header (uint8_t *p_msg, uint16_t deviceAddress, uint8_t command,
00269                  uint8_t lengthMsg)
00270 {
00271     {
00272         ospHeader_t hdr;
00273
00274         hdr.bit.preamble = OSP_PROTOCOL_PREAMBLE;
00275         hdr.bit.address = deviceAddress;
00276
00277         if (lengthMsg == 12)
00278         {
00279             hdr.bit.psi = 7;
00280         }
00281         else
00282         {
00283             hdr.bit.psi = lengthMsg - 4;
00284         }
00285         hdr.bit.command = command;
00286
00287         for (uint8_t i = 0; i < 3; i++)
00288         {
00289             *p_msg = hdr.buf[3 - i];
00290             p_msg++;
00291         }
00292     }

```

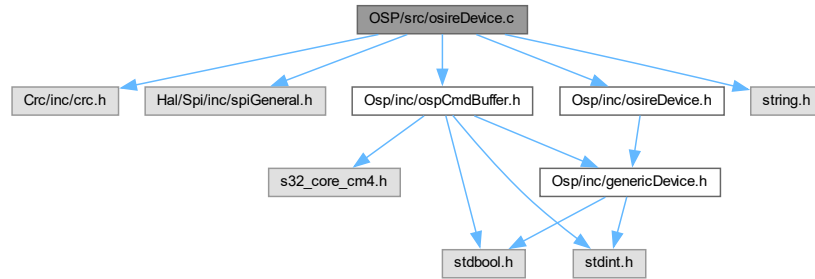
5.10 OSP/src/osireDevice.c File Reference

```

#include <Crc/inc/crc.h>
#include <Hal/Spi/inc/spiGeneral.h>
#include <Osp/inc/osireDevice.h>
#include <Osp/inc/ospCmdBuffer.h>
#include <string.h>

```

Include dependency graph for osireDevice.c:



Functions

- enum [OSP_ERROR_CODE](#) [osp_osire_set_setup](#) (uint16_t deviceAddress, [osireSetSetupData_t](#) data)
OSP_OSIRE_SET_SETUP command.
- enum [OSP_ERROR_CODE](#) [osp_osire_set_setup_and_sr](#) (uint16_t deviceAddress, [osireSetSetupData_t](#) data, [osireTempStatus_t](#) *p_rsp)
OSP_OSIRE_SET_SETUP_SR command and reads status and temperature.
- enum [OSP_ERROR_CODE](#) [osp_osire_set_pwm](#) (uint16_t deviceAddress, [osirePwmData_t](#) data)
OSP_OSIRE_SET_PWM command.
- enum [OSP_ERROR_CODE](#) [osp_osire_set_pwm_and_sr](#) (uint16_t deviceAddress, [osirePwmData_t](#) data, [osireTempStatus_t](#) *p_rsp)
OSP_OSIRE_SET_PWM_SR command and reads status and temperature.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_pwm](#) (uint16_t deviceAddress, [osirePwmData_t](#) *p_rsp)
OSP_OSIRE_READ_PWM command.
- enum [OSP_ERROR_CODE](#) [osp_read_otp](#) (uint16_t deviceAddress, uint8_t otpAddress, [osireOtpData_t](#) *p_rsp)
OSP_OSIRE_READ_OTP command Reads 8 bytes from OTP memory, from otpAddress. If readout address is beyond OTP address range, 0x00 will be delivered for these addresses.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_otp_complete](#) (uint16_t deviceAddress, [osireOtpDataComplete_t](#) *p_rsp)
Read complete OTP memory command Reads all bytes from OTP memory. By use of several consecutive OSP_↔ OSIRE_READ_OTP commands.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_ledstatus](#) (uint16_t deviceAddress, [osireLedStatus_t](#) *p_rsp)
OSP_OSIRE_READ_LED_STATUS command This function reads the LED STATUS register of the device.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_tempstatus](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_↔rsp)
OSP_OSIRE_READ_TEMP_STATUS command This function reads the STATUS and TEMP register in a single 2-byte payload of the device.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_temp](#) (uint16_t deviceAddress, [osireTemp_t](#) *p_rsp)
OSP_OSIRE_READ_TEMP command This function reads LED TEMP register of the device.
- enum [OSP_ERROR_CODE](#) [osp_osire_set_otth](#) (uint16_t deviceAddress, [osireOtthData_t](#) data)
OSP_OSIRE_SET_OTTH command.
- enum [OSP_ERROR_CODE](#) [osp_osire_set_otth_and_sr](#) (uint16_t deviceAddress, [osireOtthData_t](#) data, [osireTempStatus_t](#) *p_rsp)
OSP_OSIRE_SET_OTTH_SR command and reads status and temperature.
- enum [OSP_ERROR_CODE](#) [osp_osire_read_otth](#) (uint16_t deviceAddress, [osireOtthData_t](#) *p_rsp)
OSP_OSIRE_READ_OTTH command.
- enum [OSP_ERROR_CODE](#) [osp_go_active_and_sr](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_rsp)

- OSP_GO_ACTIVE command and read status and temperature Puts device into ACTIVE state: The LED drivers are enabled. The last PWM parameters are used to create LED PWM signals. An PWM parameter update via SETPWM() command is possible. During ACTIVE mode, diagnostic is possible and readable.*
- enum [OSP_ERROR_CODE osp_osire_go_sleep_and_sr](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_rsp)

OSP_OSIRE_GO_SLEEP_SR command and read status and temperature This function sends one or all devices into SLEEP state Additionally the STATUS and TEMP register is read.
 - enum [OSP_ERROR_CODE osp_osire_go_deep_sleep_and_sr](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_rsp)

OSP_OSIRE_GO_DEEP_SLEEP_SR command and read status and temperature This function sends one or all devices into DEEP_SLEEP state Additionally the STATUS and TEMP register is read.
 - enum [OSP_ERROR_CODE osp_osire_clr_error_and_sr](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_rsp)

OSP_CLEAR_ERROR command and read status and temperature This function will clear all error flags, if an error still exists for example short/open the error flag is set again and LED TEMPSTAT Register will be sent.
 - enum [OSP_ERROR_CODE osp_osire_p4error_bidir](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_rsp)

OSP_PING_FOR_ERROR_BIDIR command Addressed or initialized device (typ. 1st.), checks if error flag occurs. It will check only the selected flag bits which leads to SLEEP state (setup register). If yes: answer to master (state + temperature) if not: same command will be forward to next device with a new address field (STARTADDRESS+1). If no error flag bit in chain. Last device sends status + temperature register. Note: The command shall use the address of the first device only. Using an address from a unit in the middle of the chain might lead to unpredictable behavior of the chain if another unit saw a reset condition (e.g. power loss).
 - enum [OSP_ERROR_CODE osp_osire_p4error_loop](#) (uint16_t deviceAddress, [osireTempStatus_t](#) *p_rsp)

OSP_PING_FOR_ERROR_LOOP command Addressed or initialized device (typ. 1st.), checks if error flag occurs. It will check only the selected flag bits which leads to SLEEP state (setup register). If yes: answer to master (state + temperature) if not: same command will be forward to next device with a new address field (STARTADDRESS+1). If no error flag bit in chain. Last device sends status + temperature register. Note: The command shall use the address of the first device only. Using an address from a unit in the middle of the chain might lead to unpredictable behavior of the chain if another unit saw a reset condition (e.g. power loss).
 - enum [OSP_ERROR_CODE osp_osire_read_setup](#) (uint16_t deviceAddress, [osireSetSetupData_t](#) *p_rsp)

OSP_OSIRE_READ_SETUP command This function reads SETUP register of the device.
 - enum [OSP_ERROR_CODE osp_osire_read_comstatus](#) (uint16_t deviceAddress, [osireComStatus_t](#) *p_rsp)

OSP_OSIRE_READ_COM_STATUS command This function reads COM STATUS register of the device.
 - enum [OSP_ERROR_CODE osp_osire_read_status](#) (uint16_t deviceAddress, [osireStatus_t](#) *p_rsp)

OSP_OSIRE_READ_STATUS command This function reads STATUS register of the device.

5.10.1 Function Documentation

5.10.1.1 osp_go_active_and_sr()

```
enum OSP\_ERROR\_CODE osp_go_active_and_sr (
    uint16_t deviceAddress,
    osireTempStatus\_t * p_rsp )
```

OSP_GO_ACTIVE command and read status and temperature Puts device into ACTIVE state: The LED drivers are enabled. The last PWM parameters are used to create LED PWM signals. An PWM parameter update via SETPWM() command is possible. During ACTIVE mode, diagnostic is possible and readable.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

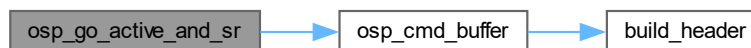
Definition at line 563 of file `osireDevice.c`.

```

00565 {
00566     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00567     ospCmdBuffer_t ospCmd;
00568     enum OSP_ERROR_CODE ospErrorCode;
00569     errorSpi_t spiError;
00570
00571     ospCmd.inCmdId = OSP_OSIRE_GO_ACTIVE_SR;
00572     ospCmd.inDeviceAddress = deviceAddress;
00573     ospCmd.p_inParameter = NULL;
00574
00575     ospErrorCode = osp_cmd_buffer (&ospCmd);
00576     if (ospErrorCode != OSP_NO_ERROR)
00577     {
00578         return ospErrorCode;
00579     }
00580
00581     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00582                                                         rspBuffer,
00583                                                         ospCmd.outCmdBufferLength,
00584                                                         ospCmd.outResponseLength);
00585
00586     if (spiError != NO_ERROR_SPI)
00587     {
00588         return OSP_ERROR_SPI;
00589     }
00590
00591     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00592     {
00593         return OSP_ERROR_CRC;
00594     }
00595
00596     for (uint8_t i = 0; i < 2; i++)
00597     {
00598         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00599     }
00600
00601     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00602     return OSP_NO_ERROR;
00603 }

```

Here is the call graph for this function:

**5.10.1.2 osp_osire_clr_error_and_sr()**

```

enum OSP_ERROR_CODE osp_osire_clr_error_and_sr (
    uint16_t deviceAddress,
    osireTempStatus_t * p_data )

```

OSP_CLEAR_ERROR command and read status and temperature This function will clear all error flags, if an error still exists for example short/open the error flag is set again and LED TEMPSTAT Register will be sent.

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

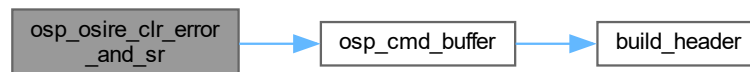
Definition at line 699 of file `osireDevice.c`.

```

00701 {
00702     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // message buffer
00703     ospCmdBuffer_t ospCmd;
00704     enum OSP_ERROR_CODE ospErrorCode;
00705     errorSpi_t spiError;
00706
00707     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00708
00709     ospCmd.inCmdId = OSP_OSIRE_CLR_ERROR_SR;
00710     ospCmd.inDeviceAddress = deviceAddress;
00711     ospCmd.p_inParameter = NULL;
00712
00713     ospErrorCode = osp_cmd_buffer (&ospCmd);
00714     if (ospErrorCode != OSP_NO_ERROR)
00715     {
00716         return ospErrorCode;
00717     }
00718
00719     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00720                                                         rspBuffer,
00721                                                         ospCmd.outCmdBufferLength,
00722                                                         ospCmd.outResponseLength);
00723
00724     if (spiError != NO_ERROR_SPI)
00725     {
00726         return OSP_ERROR_SPI;
00727     }
00728
00729     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00730     {
00731         return OSP_ERROR_CRC;
00732     }
00733
00734     for (uint8_t i = 0; i < 2; i++)
00735     {
00736         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00737     }
00738
00739     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00740     return OSP_NO_ERROR;
00741 }

```

Here is the call graph for this function:

**5.10.1.3 osp_osire_go_deep_sleep_and_sr()**

```
enum OSP_ERROR_CODE osp_osire_go_deep_sleep_and_sr (
```

```
uint16_t deviceAddress,
osireTempStatus_t * p_data )
```

OSP_OSIRE_GO_DEEP_SLEEP_SR command and read status and temperature This function sends one or all devices into DEEP_SLEEP state Additionally the STATUS and TEMP register is read.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

Definition at line 653 of file `osireDevice.c`.

```
00655 {
00656     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00657     ospCmdBuffer_t ospCmd;
00658     enum OSP_ERROR_CODE ospErrorCode;
00659     errorSpi_t spiError;
00660
00661     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00662
00663     ospCmd.inCmdId = OSP_OSIRE_GO_DEEP_SLEEP_SR;
00664     ospCmd.inDeviceAddress = deviceAddress;
00665     ospCmd.p_inParameter = NULL;
00666
00667     ospErrorCode = osp_cmd_buffer (&ospCmd);
00668     if (ospErrorCode != OSP_NO_ERROR)
00669     {
00670         return ospErrorCode;
00671     }
00672
00673     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00674                                                         rspBuffer,
00675                                                         ospCmd.outCmdBufferLength,
00676                                                         ospCmd.outResponseLength);
00677
00678     if (spiError != NO_ERROR_SPI)
00679     {
00680         return OSP_ERROR_SPI;
00681     }
00682
00683     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00684     {
00685         return OSP_ERROR_CRC;
00686     }
00687
00688     for (uint8_t i = 0; i < 2; i++)
00689     {
00690         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00691     }
00692
00693     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00694     return OSP_NO_ERROR;
00695 }
```

Here is the call graph for this function:



5.10.1.4 osp_osire_go_sleep_and_sr()

```
enum OSP_ERROR_CODE osp_osire_go_sleep_and_sr (
```

```
uint16_t deviceAddress,
osireTempStatus_t * p_data )
```

OSP_OSIRE_GO_SLEEP_SR command and read status and temperature This function sends one or all devices into SLEEP state Additionally the STATUS and TEMP register is read.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	STATUS and TEMP register response data

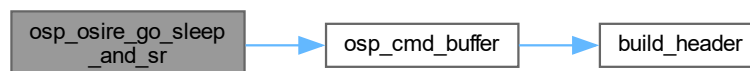
Returns

error communication or command parameter error

Definition at line 607 of file `osireDevice.c`.

```
00609 {
00610     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00611     ospCmdBuffer_t ospCmd;
00612     enum OSP_ERROR_CODE ospErrorCode;
00613     errorSpi_t spiError;
00614
00615     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00616
00617     ospCmd.inCmdId = OSP_OSIRE_GO_SLEEP_SR;
00618     ospCmd.inDeviceAddress = deviceAddress;
00619     ospCmd.p_inParameter = NULL;
00620
00621     ospErrorCode = osp_cmd_buffer (&ospCmd);
00622     if (ospErrorCode != OSP_NO_ERROR)
00623     {
00624         return ospErrorCode;
00625     }
00626
00627     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00628                                                         rspBuffer,
00629                                                         ospCmd.outCmdBufferLength,
00630                                                         ospCmd.outResponseLength);
00631
00632     if (spiError != NO_ERROR_SPI)
00633     {
00634         return OSP_ERROR_SPI;
00635     }
00636
00637     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00638     {
00639         return OSP_ERROR_CRC;
00640     }
00641
00642     for (uint8_t i = 0; i < 2; i++)
00643     {
00644         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00645     }
00646
00647     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00648     return OSP_NO_ERROR;
00649 }
```

Here is the call graph for this function:



5.10.1.5 osp_osire_p4error_bidir()

```
enum OSP_ERROR_CODE osp_osire_p4error_bidir (
    uint16_t deviceAddress,
    osireTempStatus_t * p_data )
```

OSP_PING_FOR_ERROR_BIDIR command Addressed or initialized device (typ. 1st.), checks if error flag occurs. It will check only the selected flag bits which leads to SLEEP state (setup register). If yes: answer to master (state + temperature) if not: same command will be forward to next device with a new address field (STARTADDRESS+1). If no error flag bit in chain. Last device sends status + temperature register. Note: The command shall use the address of the first device only. Using an address from a unit in the middle of the chain might lead to unpredictable behavior of the chain if another unit saw a reset condition (e.g. power loss).

Parameters

<i>deviceAddress</i>	of first RGBi device (typ. 1)
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

Parameters

<i>deviceAddress</i>	of first RGBi device (typ. 1)
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

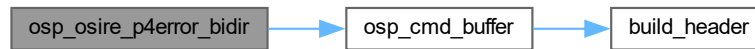
Definition at line 745 of file `osireDevice.c`.

```
00747 {
00748     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // message buffer
00749     ospCmdBuffer_t ospCmd;
00750     enum OSP_ERROR_CODE ospErrorCode;
00751     errorSpi_t spiError;
00752
00753     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00754
00755     ospCmd.inCmdId = OSP_OSIRE_P4ERROR_BIDIR;
00756     ospCmd.inDeviceAddress = deviceAddress;
00757     ospCmd.p_inParameter = NULL;
00758
00759     ospErrorCode = osp_cmd_buffer (&ospCmd);
00760     if (ospErrorCode != OSP_NO_ERROR)
00761     {
00762         return ospErrorCode;
00763     }
00764
00765     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00766                                                         rspBuffer,
00767                                                         ospCmd.outCmdBufferLength,
00768                                                         ospCmd.outResponseLength);
00769
00770     if (spiError != NO_ERROR_SPI)
00771     {
00772         return OSP_ERROR_SPI;
00773     }
00774
00775     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00776     {
00777         return OSP_ERROR_CRC;
00778     }
00779
00780     for (uint8_t i = 0; i < 2; i++)
00781     {
00782         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00783     }
00784
00785     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00786     return OSP_NO_ERROR;
```



```
00787 }
```

Here is the call graph for this function:



5.10.1.6 osp_osire_p4error_loop()

```
enum OSP_ERROR_CODE osp_osire_p4error_loop (
    uint16_t deviceAddress,
    osireTempStatus_t * p_data )
```

OSP_PING_FOR_ERROR_LOOP command Addressed or initialized device (typ. 1st.), checks if error flag occurs. It will check only the selected flag bits which leads to SLEEP state (setup register). If yes: answer to master (state + temperature) if not: same command will be forward to next device with a new address field (STARTADDRESS+1). If no error flag bit in chain. Last device sends status + temperature register. Note: The command shall use the address of the first device only. Using an address from a unit in the middle of the chain might lead to unpredictable behavior of the chain if another unit saw a reset condition (e.g. power loss).

Parameters

<i>deviceAddress</i>	of first RGBi device (typ. 1)
<i>p_data,pointer</i>	to response data from RGBi LED

Returns

error communication or command parameter error

Definition at line 790 of file `osireDevice.c`.

```

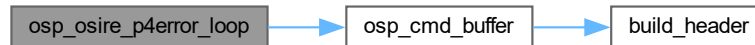
00792 {
00793     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // message buffer
00794     ospCmdBuffer_t ospCmd;
00795     enum OSP_ERROR_CODE ospErrorCode;
00796     errorSpi_t spiError;
00797
00798     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00799
00800     ospCmd.inCmdId = OSP_OSIRE_P4ERROR_LOOP;
00801     ospCmd.inDeviceAddress = deviceAddress;
00802     ospCmd.p_inParameter = NULL;
00803
00804     ospErrorCode = osp_cmd_buffer (&ospCmd);
00805     if (ospErrorCode != OSP_NO_ERROR)
00806     {
00807         return ospErrorCode;
00808     }
00809
00810     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00811                                                         rspBuffer,
00812                                                         ospCmd.outCmdBufferLength,
00813                                                         ospCmd.outResponseLength);
00814
00815     if (spiError != NO_ERROR_SPI)
00816     {
00817         return OSP_ERROR_SPI;
00818     }
00819
00820     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00821     {
00822         return OSP_ERROR_CRC;
00823     }
00824
00825     for (uint8_t i = 0; i < 2; i++)
```

```

00826     {
00827         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00828     }
00829
00830     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00831     return OSP_NO_ERROR;
00832 }

```

Here is the call graph for this function:



5.10.1.7 osp_osire_read_comstatus()

```

enum OSP_ERROR_CODE osp_osire_read_comstatus (
    uint16_t deviceAddress,
    osireComStatus_t * p_data )

```

OSP_OSIRE_READ_COM_STATUS command This function reads COM STATUS register of the device.
For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	pointer to response data from RGBi LED

Returns

error COM STATUS register response data

Definition at line 878 of file `osireDevice.c`.

```

00880 {
00881     uint8_t rspBuffer[LENGTH_READ_COMSTATUS_RSP]; // response buffer
00882     ospCmdBuffer_t ospCmd;
00883     enum OSP_ERROR_CODE ospErrorCode;
00884     errorSpi_t spiError;
00885
00886     memset (rspBuffer, 0, LENGTH_READ_COMSTATUS_RSP);
00887
00888     ospCmd.inCmdId = OSP_OSIRE_READ_COM_STATUS;
00889     ospCmd.inDeviceAddress = deviceAddress;
00890     ospCmd.p_inParameter = NULL;
00891
00892     ospErrorCode = osp_cmd_buffer (&ospCmd);
00893     if (ospErrorCode != OSP_NO_ERROR)
00894     {
00895         return ospErrorCode;
00896     }
00897
00898     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00899                                                         rspBuffer,
00900                                                         ospCmd.outCmdBufferLength,
00901                                                         ospCmd.outResponseLength);
00902
00903     if (spiError != NO_ERROR_SPI)
00904     {
00905         return OSP_ERROR_SPI;
00906     }
00907
00908     if (crc (rspBuffer, LENGTH_READ_COMSTATUS_RSP) != 0)
00909     {
00910         return OSP_ERROR_CRC;
00911     }
00912
00913     p_rsp->data.comStatus = rspBuffer[FIRST_BYTE_PAYLOAD];

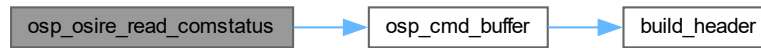
```

```

00914
00915     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00916     return OSP_NO_ERROR;
00917 }

```

Here is the call graph for this function:



5.10.1.8 osp_osire_read_ledstatus()

```

enum OSP_ERROR_CODE osp_osire_read_ledstatus (
    uint16_t deviceAddress,
    osireLedStatus_t * p_data )

```

OSP_OSIRE_READ_LED_STATUS command This function reads the LED STATUS register of the device. For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	LED STATUS register response data

Returns

error communication or command parameter error

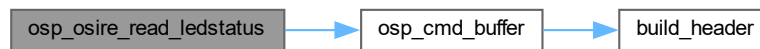
Definition at line 308 of file `osireDevice.c`.

```

00310 {
00311     uint8_t rspBuffer[LENGTH_READ_LEDSTATUS_RSP]; // response buffer
00312     ospCmdBuffer_t ospCmd;
00313     enum OSP_ERROR_CODE ospErrorCode;
00314     errorSpi_t spiError;
00315
00316     memset (rspBuffer, 0, LENGTH_READ_LEDSTATUS_RSP);
00317
00318     ospCmd.inCmdId = OSP_OSIRE_READ_LED_STATUS;
00319     ospCmd.inDeviceAddress = deviceAddress;
00320     ospCmd.p_inParameter = NULL;
00321
00322     ospErrorCode = osp_cmd_buffer (&ospCmd);
00323     if (ospErrorCode != OSP_NO_ERROR)
00324     {
00325         return ospErrorCode;
00326     }
00327
00328     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00329                                                         rspBuffer,
00330                                                         ospCmd.outCmdBufferLength,
00331                                                         ospCmd.outResponseLength);
00332
00333     if (spiError != NO_ERROR_SPI)
00334     {
00335         return OSP_ERROR_SPI;
00336     }
00337
00338     if (crc (rspBuffer, LENGTH_READ_LEDSTATUS_RSP) != 0)
00339     {
00340         return OSP_ERROR_CRC;
00341     }
00342
00343     p_rsp->data.ledStatus = rspBuffer[3];
00344
00345     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00346     return OSP_NO_ERROR;
00347 }

```

Here is the call graph for this function:



5.10.1.9 osp_osire_read_otp_complete()

```
enum OSP_ERROR_CODE osp_osire_read_otp_complete (
    uint16_t deviceAddress,
    osireOtpDataComplete_t * p_data )
```

Read complete OTP memory command Reads all bytes from OTP memory. By use of several consecutive OSP↔_OSIRE_READ_OTP commands.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	full OTP memory data

Returns

error communication or command parameter error

Definition at line 282 of file `osireDevice.c`.

```

00284 {
00285     osireOtpData_t opt; // OTP buffer
00286
00287     for (uint8_t i = 0; i < 0x1F; i = i + 8)
00288     {
00289         enum OSP_ERROR_CODE errorCode = osp_read_otp (deviceAddress, i, &opt);
00290
00291         if (errorCode != OSP_NO_ERROR)
00292         {
00293             return errorCode;
00294         }
00295
00296         for (uint8_t j = 0; j < 8; j++)
00297         {
00298             p_rsp->data.byte[i + j] = opt.data.byte[j];
00299         }
00300     }
00301
00302     p_rsp->address = opt.address;
00303     return OSP_NO_ERROR;
00304 }
```

Here is the call graph for this function:



5.10.1.10 osp_osire_read_otth()

```
enum OSP_ERROR_CODE osp_osire_read_otth (
    uint16_t deviceAddress,
    osireOtthData_t * p_data )
```

OSP_OSIRE_READ_OTTH command.

This function reads the OTTH register of the device

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	PWM register response data

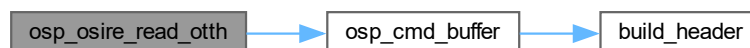
Returns

error communication or command parameter error

Definition at line 517 of file `osireDevice.c`.

```
00519 {
00520     uint8_t rspBuffer[LENGTH_READ_OTTH_RSP]; // response buffer
00521     ospCmdBuffer_t ospCmd;
00522     enum OSP_ERROR_CODE ospErrorCode;
00523     errorSpi_t spiError;
00524
00525     memset (rspBuffer, 0, LENGTH_READ_OTTH_RSP);
00526
00527     ospCmd.inCmdId = OSP_OSIRE_READ_OTTH;
00528     ospCmd.inDeviceAddress = deviceAddress;
00529     ospCmd.p_inParameter = NULL;
00530
00531     ospErrorCode = osp_cmd_buffer (&ospCmd);
00532     if (ospErrorCode != OSP_NO_ERROR)
00533     {
00534         return ospErrorCode;
00535     }
00536
00537     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00538                                                         rspBuffer,
00539                                                         ospCmd.outCmdBufferLength,
00540                                                         ospCmd.outResponseLength);
00541
00542     if (spiError != NO_ERROR_SPI)
00543     {
00544         return OSP_ERROR_SPI;
00545     }
00546
00547     if (crc (rspBuffer, LENGTH_READ_OTTH_RSP) != 0)
00548     {
00549         return OSP_ERROR_CRC;
00550     }
00551
00552     for (uint8_t i = 0; i < 3; i++)
00553     {
00554         p_rsp->data.otthData[i] = rspBuffer[5 - i];
00555     }
00556
00557     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00558     return OSP_NO_ERROR;
00559 }
```

Here is the call graph for this function:



5.10.1.11 osp_osire_read_pwm()

```
enum OSP_ERROR_CODE osp_osire_read_pwm (
    uint16_t deviceAddress,
    osirePwmData_t * p_data )
```

OSP_OSIRE_READ_PWM command.

This function reads the PWM register of the device

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	PWM register response data

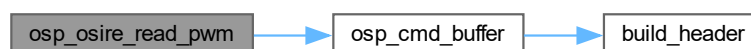
Returns

error communication or command parameter error

Definition at line 190 of file `osireDevice.c`.

```
00192 {
00193     uint8_t rspBuffer[LENGTH_READ_PWM_RSP]; // response buffer
00194     ospCmdBuffer_t ospCmd;
00195     enum OSP_ERROR_CODE ospErrorCode;
00196     errorSpi_t spiError;
00197
00198     memset (rspBuffer, 0, LENGTH_READ_PWM_RSP);
00199
00200     ospCmd.inCmdId = OSP_OSIRE_READ_PWM;
00201     ospCmd.inDeviceAddress = deviceAddress;
00202     ospCmd.p_inParameter = NULL;
00203
00204     ospErrorCode = osp_cmd_buffer (&ospCmd);
00205     if (ospErrorCode != OSP_NO_ERROR)
00206     {
00207         return ospErrorCode;
00208     }
00209
00210     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00211                                                         rspBuffer,
00212                                                         ospCmd.outCmdBufferLength,
00213                                                         ospCmd.outResponseLength);
00214
00215     if (spiError != NO_ERROR_SPI)
00216     {
00217         return OSP_ERROR_SPI;
00218     }
00219
00220     if (crc (rspBuffer, LENGTH_READ_PWM_RSP) != 0)
00221     {
00222         return OSP_ERROR_CRC;
00223     }
00224
00225     for (uint8_t i = 0; i < 6; i++)
00226     {
00227         p_rsp->data.pwmData[i] = rspBuffer[8 - i];
00228     }
00229
00230     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00231     return OSP_NO_ERROR;
00232 }
```

Here is the call graph for this function:



5.10.1.12 osp_osire_read_setup()

```
enum OSP_ERROR_CODE osp_osire_read_setup (
    uint16_t deviceAddress,
    osireSetSetupData_t * p_data )
```

OSP_OSIRE_READ_SETUP command This function reads SETUP register of the device.
For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	SETUP register response data

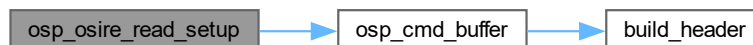
Returns

error communication or command parameter error

Definition at line 835 of file [osireDevice.c](#).

```
00837 {
00838     uint8_t rspBuffer[LENGTH_READ_SETUP_RSP]; // response buffer
00839     ospCmdBuffer_t ospCmd;
00840     enum OSP_ERROR_CODE ospErrorCode;
00841     errorSpi_t spiError;
00842
00843     memset (rspBuffer, 0, LENGTH_READ_SETUP_RSP);
00844
00845     ospCmd.inCmdId = OSP_OSIRE_READ_SETUP;
00846     ospCmd.inDeviceAddress = deviceAddress;
00847     ospCmd.p_inParameter = NULL;
00848
00849     ospErrorCode = osp_cmd_buffer (&ospCmd);
00850     if (ospErrorCode != OSP_NO_ERROR)
00851     {
00852         return ospErrorCode;
00853     }
00854
00855     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00856                                                         rspBuffer,
00857                                                         ospCmd.outCmdBufferLength,
00858                                                         ospCmd.outResponseLength);
00859
00860     if (spiError != NO_ERROR_SPI)
00861     {
00862         return OSP_ERROR_SPI;
00863     }
00864
00865     if (crc (rspBuffer, LENGTH_READ_SETUP_RSP) != 0)
00866     {
00867         return OSP_ERROR_CRC;
00868     }
00869
00870     p_rsp->data.setupData = rspBuffer[FIRST_BYTE_PAYLOAD];
00871
00872     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00873     return OSP_NO_ERROR;
00874 }
```

Here is the call graph for this function:



5.10.1.13 osp_osire_read_status()

```
enum OSP_ERROR_CODE osp_osire_read_status (
```

```
uint16_t deviceAddress,
osireStatus_t * p_data )
```

OSP_OSIRE_READ_STATUS command This function reads STATUS register of the device.
For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	STATUS register response data

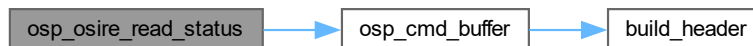
Returns

error communication or command parameter error

Definition at line 921 of file `osireDevice.c`.

```
00923 {
00924     uint8_t rspBuffer[LENGTH_READ_STATUS_RSP]; // response buffer
00925     ospCmdBuffer_t ospCmd;
00926     enum OSP_ERROR_CODE ospErrorCode;
00927     errorsSpi_t spiError;
00928
00929     memset (rspBuffer, 0, LENGTH_READ_STATUS_RSP);
00930
00931     ospCmd.inCmdId = OSP_OSIRE_READ_STATUS;
00932     ospCmd.inDeviceAddress = deviceAddress;
00933     ospCmd.p_inParameter = NULL;
00934
00935     ospErrorCode = osp_cmd_buffer (&ospCmd);
00936     if (ospErrorCode != OSP_NO_ERROR)
00937     {
00938         return ospErrorCode;
00939     }
00940
00941     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00942                                                         rspBuffer,
00943                                                         ospCmd.outCmdBufferLength,
00944                                                         ospCmd.outResponseLength);
00945
00946     if (spiError != NO_ERROR_SPI)
00947     {
00948         return OSP_ERROR_SPI;
00949     }
00950
00951     if (crc (rspBuffer, LENGTH_READ_STATUS_RSP) != 0)
00952     {
00953         return OSP_ERROR_CRC;
00954     }
00955
00956     p_rsp->data.status = rspBuffer[FIRST_BYTE_PAYLOAD];
00957
00958     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00959     return OSP_NO_ERROR;
00960 }
```

Here is the call graph for this function:



5.10.1.14 osp_osire_read_temp()

```
enum OSP_ERROR_CODE osp_osire_read_temp (
    uint16_t deviceAddress,
    osireTemp_t * p_data )
```


OSP_OSIRE_READ_TEMP command This function reads LED TEMP register of the device.
For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	LED TEMP register response data

Returns

error communication or command parameter error

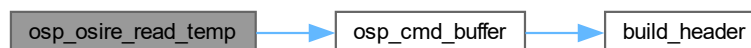
Definition at line 397 of file `osireDevice.c`.

```

00399 {
00400     uint8_t rspBuffer[LENGTH_READ_TEMP_RSP]; // response buffer
00401     ospCmdBuffer_t ospCmd;
00402     enum OSP_ERROR_CODE ospErrorCode;
00403     errorSpi_t spiError;
00404
00405     memset (rspBuffer, 0, LENGTH_READ_TEMP_RSP);
00406
00407     ospCmd.inCmdId = OSP_OSIRE_READ_TEMP;
00408     ospCmd.inDeviceAddress = deviceAddress;
00409     ospCmd.p_inParameter = NULL;
00410
00411     ospErrorCode = osp_cmd_buffer (&ospCmd);
00412     if (ospErrorCode != OSP_NO_ERROR)
00413     {
00414         return ospErrorCode;
00415     }
00416
00417     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00418                                                         rspBuffer,
00419                                                         ospCmd.outCmdBufferLength,
00420                                                         ospCmd.outResponseLength);
00421
00422     if (spiError != NO_ERROR_SPI)
00423     {
00424         return OSP_ERROR_SPI;
00425     }
00426
00427     if (crc (rspBuffer, LENGTH_READ_TEMP_RSP) != 0)
00428     {
00429         return OSP_ERROR_CRC;
00430     }
00431
00432     p_rsp->data.temp_value = rspBuffer[FIRST_BYTE_PAYLOAD];
00433
00434     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00435     return OSP_NO_ERROR;
00436 }

```

Here is the call graph for this function:



5.10.1.15 osp_osire_read_tempstatus()

```

enum OSP_ERROR_CODE osp_osire_read_tempstatus (
    uint16_t deviceAddress,
    osireTempStatus_t * p_data )

```

OSP_OSIRE_READ_TEMP_STATUS command This function reads the STATUS and TEMP register in a single 2-byte payload of the device.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

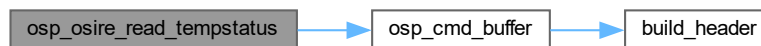
Definition at line 351 of file `osireDevice.c`.

```

00353 {
00354     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00355     ospCmdBuffer_t ospCmd;
00356     enum OSP_ERROR_CODE ospErrorCode;
00357     errorSpi_t spiError;
00358
00359     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00360
00361     ospCmd.inCmdId = OSP_OSIRE_READ_TEMP_STATUS;
00362     ospCmd.inDeviceAddress = deviceAddress;
00363     ospCmd.p_inParameter = NULL;
00364
00365     ospErrorCode = osp_cmd_buffer (&ospCmd);
00366     if (ospErrorCode != OSP_NO_ERROR)
00367     {
00368         return ospErrorCode;
00369     }
00370
00371     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00372                                                         rspBuffer,
00373                                                         ospCmd.outCmdBufferLength,
00374                                                         ospCmd.outResponseLength);
00375
00376     if (spiError != NO_ERROR_SPI)
00377     {
00378         return OSP_ERROR_SPI;
00379     }
00380
00381     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00382     {
00383         return OSP_ERROR_CRC;
00384     }
00385
00386     for (uint8_t i = 0; i < 2; i++)
00387     {
00388         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00389     }
00390
00391     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00392     return OSP_NO_ERROR;
00393 }

```

Here is the call graph for this function:



5.10.1.16 osp_osire_set_otth()

```

enum OSP_ERROR_CODE osp_osire_set_otth (
    uint16_t deviceAddress,
    osireOtthData_t data )

```

OSP_OSIRE_SET_OTTH command.

Writes the OTTH register. See READOTTH for the payload format.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
<i>data</i>	PWM register value

Returns

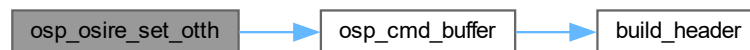
error, communication or command parameter error

Definition at line 440 of file `osireDevice.c`.

```

00442 {
00443     ospCmdBuffer_t ospCmd;
00444     enum OSP_ERROR_CODE ospErrorCode;
00445     errorSpi_t spiError;
00446
00447     ospCmd.inCmdId = OSP_OSIRE_SET_OTTH;
00448     ospCmd.inDeviceAddress = deviceAddress;
00449     ospCmd.p_inParameter = &data.data.otthData;
00450
00451     ospErrorCode = osp_cmd_buffer (&ospCmd);
00452     if (ospErrorCode != OSP_NO_ERROR)
00453     {
00454         return ospErrorCode;
00455     }
00456
00457     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00458                                           ospCmd.outCmdBufferLength);
00459
00460     if (spiError != NO_ERROR_SPI)
00461     {
00462         return OSP_ERROR_SPI;
00463     }
00464
00465     return OSP_NO_ERROR;
00466 }
```

Here is the call graph for this function:

5.10.1.17 `osp_osire_set_otth_and_sr()`

```

enum OSP_ERROR_CODE osp_osire_set_otth_and_sr (
    uint16_t deviceAddress,
    osireOtthData_t data,
    osireTempStatus_t * p_data )
```

OSP_OSIRE_SET_OTTH_SR command and reads status and temperature.

Writes the OTTH register. See READOTTH for the payload format. Additionally the STATUS and TEMP register is read.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>data</i>	PWM register value
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

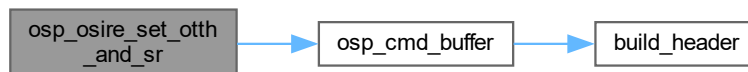
Definition at line 470 of file `osireDevice.c`.

```

00473 {
00474     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00475     ospCmdBuffer_t ospCmd;
00476     enum OSP_ERROR_CODE ospErrorCode;
00477     errorSpi_t spiError;
00478
00479     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00480
00481     ospCmd.inCmdId = OSP_OSIRE_SET_OTTH_SR;
00482     ospCmd.inDeviceAddress = deviceAddress;
00483     ospCmd.p_inParameter = &data.data.otthData;
00484
00485     ospErrorCode = osp_cmd_buffer (&ospCmd);
00486     if (ospErrorCode != OSP_NO_ERROR)
00487     {
00488         return ospErrorCode;
00489     }
00490
00491     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00492                                                         rspBuffer,
00493                                                         ospCmd.outCmdBufferLength,
00494                                                         ospCmd.outResponseLength);
00495
00496     if (spiError != NO_ERROR_SPI)
00497     {
00498         return OSP_ERROR_SPI;
00499     }
00500
00501     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00502     {
00503         return OSP_ERROR_CRC;
00504     }
00505
00506     for (uint8_t i = 0; i < 2; i++)
00507     {
00508         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00509     }
00510
00511     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00512     return OSP_NO_ERROR;
00513 }

```

Here is the call graph for this function:



5.10.1.18 osp_osire_set_pwm()

```

enum OSP_ERROR_CODE osp_osire_set_pwm (
    uint16_t deviceAddress,
    osirePwmData_t data )

```

OSP_OSIRE_SET_PWM command.

Writes the PWM register. See READPWM for the payload format.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
<i>data</i>	PWM register value

Returns

error, communication or command parameter error

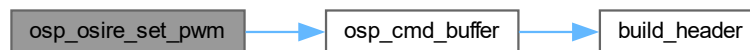
Definition at line 112 of file [osireDevice.c](#).

```

00114 {
00115     ospCmdBuffer_t ospCmd;
00116     enum OSP_ERROR_CODE ospErrorCode;
00117     errorSpi_t spiError;
00118
00119     ospCmd.inCmdId = OSP_OSIRE_SET_PWM;
00120     ospCmd.inDeviceAddress = deviceAddress;
00121     ospCmd.p_inParameter = &data.data.pwmData;
00122
00123     ospErrorCode = osp_cmd_buffer (&ospCmd);
00124     if (ospErrorCode != OSP_NO_ERROR)
00125     {
00126         return ospErrorCode;
00127     }
00128
00129     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00130                                           ospCmd.outCmdBufferLength);
00131
00132     if (spiError != NO_ERROR_SPI)
00133     {
00134         return OSP_ERROR_SPI;
00135     }
00136
00137     return OSP_NO_ERROR;
00138 }

```

Here is the call graph for this function:

**5.10.1.19 osp_osire_set_pwm_and_sr()**

```

enum OSP_ERROR_CODE osp_osire_set_pwm_and_sr (
    uint16_t deviceAddress,
    osirePwmData_t data,
    osireTempStatus_t * p_data )

```

OSP_OSIRE_SET_PWM_SR command and reads status and temperature.

Writes the PWM register. See READPWM for the payload format. Additionally the STATUS and TEMP register is read.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	0..1000 RGBi device address (0: broadcast)
<i>data</i>	PWM register value
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

Definition at line 143 of file [osireDevice.c](#).

```

00146 {
00147     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00148     ospCmdBuffer_t ospCmd;
00149     enum OSP_ERROR_CODE ospErrorCode;

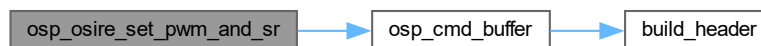
```

```

00150     errorSpi_t spiError;
00151
00152     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00153
00154     ospCmd.inCmdId = OSP_OSIRE_SET_PWM_SR;
00155     ospCmd.inDeviceAddress = deviceAddress;
00156     ospCmd.p_inParameter = &data.data.pwmData;
00157
00158     ospErrorCode = osp_cmd_buffer (&ospCmd);
00159     if (ospErrorCode != OSP_NO_ERROR)
00160     {
00161         return ospErrorCode;
00162     }
00163
00164     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00165                                                         rspBuffer,
00166                                                         ospCmd.outCmdBufferLength,
00167                                                         ospCmd.outResponseLength);
00168
00169     if (spiError != NO_ERROR_SPI)
00170     {
00171         return OSP_ERROR_SPI;
00172     }
00173
00174     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00175     {
00176         return OSP_ERROR_CRC;
00177     }
00178
00179     for (uint8_t i = 0; i < 2; i++)
00180     {
00181         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00182     }
00183
00184     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00185     return OSP_NO_ERROR;
00186 }

```

Here is the call graph for this function:



5.10.1.20 osp_osire_set_setup()

```

enum OSP_ERROR_CODE osp_osire_set_setup (
    uint16_t deviceAddress,
    osireSetSetupData_t data )

```

OSP_OSIRE_SET_SETUP command.

Include OSP Led Definitions and Data structures

Definition at line 36 of file [osireDevice.c](#).

```

00038 {
00039     ospCmdBuffer_t ospCmd;
00040     enum OSP_ERROR_CODE ospErrorCode;
00041     errorSpi_t spiError;
00042
00043     ospCmd.inCmdId = OSP_OSIRE_SET_SETUP;
00044     ospCmd.inDeviceAddress = deviceAddress;
00045     ospCmd.p_inParameter = &data.data.setupData;
00046
00047     ospErrorCode = osp_cmd_buffer (&ospCmd);
00048     if (ospErrorCode != OSP_NO_ERROR)
00049     {
00050         return ospErrorCode;
00051     }
00052
00053     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00054                                             ospCmd.outCmdBufferLength);
00055
00056     if (spiError != NO_ERROR_SPI)
00057     {

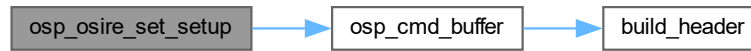
```

```

00058     return OSP_ERROR_SPI;
00059 }
00060
00061 return OSP_NO_ERROR;
00062 }

```

Here is the call graph for this function:



5.10.1.21 osp_osire_set_setup_and_sr()

```

enum OSP_ERROR_CODE osp_osire_set_setup_and_sr (
    uint16_t deviceAddress,
    osireSetSetupData_t data,
    osireTempStatus_t * p_data )

```

OSP_OSIRE_SET_SETUP_SR command and reads status and temperature.

Writes the SETUP register. See READSETUP for the payload format. Additionally the STATUS and TEMP register is read.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress, 0..1000</i>	RGBi device address (0: broadcast)
<i>data</i>	SETUP register values
<i>p_data</i>	STATUS and TEMP register response data

Returns

error communication or command parameter error

Definition at line 66 of file [osireDevice.c](#).

```

00069 {
00070     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00071     ospCmdBuffer_t ospCmd;
00072     enum OSP_ERROR_CODE ospErrorCode;
00073     errorSpi_t spiError;
00074
00075     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00076
00077     ospCmd.inCmdId = OSP_OSIRE_SET_SETUP_SR;
00078     ospCmd.inDeviceAddress = deviceAddress;
00079     ospCmd.p_inParameter = &data.data.setupData;
00080
00081     ospErrorCode = osp_cmd_buffer (&ospCmd);
00082     if (ospErrorCode != OSP_NO_ERROR)
00083     {
00084         return ospErrorCode;
00085     }
00086
00087     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00088                                                         rspBuffer,
00089                                                         ospCmd.outCmdBufferLength,
00090                                                         ospCmd.outResponseLength);
00091
00092     if (spiError != NO_ERROR_SPI)
00093     {
00094         return OSP_ERROR_SPI;
00095     }
00096
00097     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)

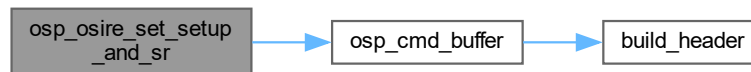
```

```

00098     {
00099         return OSP_ERROR_CRC;
00100     }
00101
00102     for (uint8_t i = 0; i < 2; i++)
00103     {
00104         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00105     }
00106     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00107     return OSP_NO_ERROR;
00108 }

```

Here is the call graph for this function:



5.10.1.22 osp_read_otp()

```

enum OSP_ERROR_CODE osp_read_otp (
    uint16_t deviceAddress,
    uint8_t otpAddress,
    osireOtpData_t * p_data )

```

OSP_OSIRE_READ_OTP command Reads 8 bytes from OTP memory, from otpAddress. If readout address is beyond OTP address range, 0x00 will be delivered for these addresses.

For further details refer to "OSIRE_E3731i_Start_Up_Guide.pdf"

Parameters

<i>deviceAddress</i>	1..1000 RGBi device address
<i>otpAddress</i>	address of the first read byte of OTP
<i>p_data</i>	OTP response data

Returns

error communication or command parameter error

Definition at line 236 of file [osireDevice.c](#).

```

00238 {
00239     uint8_t rspBuffer[LENGTH_READ_OTP_RSP]; // response buffer
00240     ospCmdBuffer_t ospCmd;
00241     enum OSP_ERROR_CODE ospErrorCode;
00242     errorSpi_t spiError;
00243
00244     memset (rspBuffer, 0, LENGTH_READ_OTP_RSP);
00245
00246     ospCmd.inCmdId = OSP_OSIRE_READ_OTP;
00247     ospCmd.inDeviceAddress = deviceAddress;
00248     ospCmd.p_inParameter = &otpAddress;
00249
00250     ospErrorCode = osp_cmd_buffer (&ospCmd);
00251     if (ospErrorCode != OSP_NO_ERROR)
00252     {
00253         return ospErrorCode;
00254     }
00255
00256     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00257                                                         rspBuffer,
00258                                                         ospCmd.outCmdBufferLength,
00259                                                         ospCmd.outResponseLength);
00260
00261     if (spiError != NO_ERROR_SPI)

```

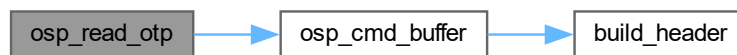


```

00262     {
00263         return OSP_ERROR_SPI;
00264     }
00265
00266     if (crc (rspBuffer, LENGTH_READ_OTP_RSP) != 0)
00267     {
00268         return OSP_ERROR_CRC;
00269     }
00270
00271     for (uint8_t i = 0; i < 8; i++)
00272     {
00273         p_rsp->data.byte[i] = rspBuffer[10 - i];
00274     }
00275
00276     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00277     return OSP_NO_ERROR;
00278 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.11 osireDevice.c

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright 2022 by ams OSRAM AG
00003  * All rights are reserved.
00004  *
00005  * IMPORTANT - PLEASE READ CAREFULLY BEFORE COPYING, INSTALLING OR USING
00006  * THE SOFTWARE.
00007  *
00008  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00009  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00010  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
00011  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00012  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00013  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00014  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00015  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00016  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00017  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00018  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00019  *****/
00020
00025 //noch keine Antwort-IDs implementiert
00026 #include <Crc/inc/crc.h>
00027 #include <Hal/Spi/inc/spiGeneral.h>
00028 #include <Osp/inc/osireDevice.h>
00029 #include <Osp/inc/ospCmdBuffer.h>
00030 #include <string.h>
00031

```

```

00032 #include <Hal/Spi/inc/spiGeneral.h>
00033
00034 /*****
00035 /*****
00036 enum OSP_ERROR_CODE osp_osire_set_setup (uint16_t deviceAddress,
00037                                         osireSetSetupData_t data)
00038 {
00039     ospCmdBuffer_t ospCmd;
00040     enum OSP_ERROR_CODE ospErrorCode;
00041     errorSpi_t spiError;
00042
00043     ospCmd.inCmdId = OSP_OSIRE_SET_SETUP;
00044     ospCmd.inDeviceAddress = deviceAddress;
00045     ospCmd.p_inParameter = &data.data.setupData;
00046
00047     ospErrorCode = osp_cmd_buffer (&ospCmd);
00048     if (ospErrorCode != OSP_NO_ERROR)
00049     {
00050         return ospErrorCode;
00051     }
00052
00053     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00054                                           ospCmd.outCmdBufferLength);
00055
00056     if (spiError != NO_ERROR_SPI)
00057     {
00058         return OSP_ERROR_SPI;
00059     }
00060
00061     return OSP_NO_ERROR;
00062 }
00063
00064 /*****
00065 /*****
00066 enum OSP_ERROR_CODE osp_osire_set_setup_and_sr (uint16_t deviceAddress,
00067                                                  osireSetSetupData_t data,
00068                                                  osireTempStatus_t *p_rsp)
00069 {
00070     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00071     ospCmdBuffer_t ospCmd;
00072     enum OSP_ERROR_CODE ospErrorCode;
00073     errorSpi_t spiError;
00074
00075     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00076
00077     ospCmd.inCmdId = OSP_OSIRE_SET_SETUP_SR;
00078     ospCmd.inDeviceAddress = deviceAddress;
00079     ospCmd.p_inParameter = &data.data.setupData;
00080
00081     ospErrorCode = osp_cmd_buffer (&ospCmd);
00082     if (ospErrorCode != OSP_NO_ERROR)
00083     {
00084         return ospErrorCode;
00085     }
00086
00087     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00088                                                         rspBuffer,
00089                                                         ospCmd.outCmdBufferLength,
00090                                                         ospCmd.outResponseLength);
00091
00092     if (spiError != NO_ERROR_SPI)
00093     {
00094         return OSP_ERROR_SPI;
00095     }
00096
00097     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00098     {
00099         return OSP_ERROR_CRC;
00100     }
00101
00102     for (uint8_t i = 0; i < 2; i++)
00103     {
00104         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00105     }
00106     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00107     return OSP_NO_ERROR;
00108 }
00109
00110 /*****
00111 /*****
00112 enum OSP_ERROR_CODE osp_osire_set_pwm (uint16_t deviceAddress,
00113                                         osirePwmData_t data)
00114 {
00115     ospCmdBuffer_t ospCmd;
00116     enum OSP_ERROR_CODE ospErrorCode;
00117     errorSpi_t spiError;
00118

```

```

00119     ospCmd.inCmdId = OSP_OSIRE_SET_PWM;
00120     ospCmd.inDeviceAddress = deviceAddress;
00121     ospCmd.p_inParameter = &data.data.pwmData;
00122
00123     ospErrorCode = osp_cmd_buffer (&ospCmd);
00124     if (ospErrorCode != OSP_NO_ERROR)
00125     {
00126         return ospErrorCode;
00127     }
00128
00129     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00130                                           ospCmd.outCmdBufferLength);
00131
00132     if (spiError != NO_ERROR_SPI)
00133     {
00134         return OSP_ERROR_SPI;
00135     }
00136
00137     return OSP_NO_ERROR;
00138 }
00139
00140 /*****
00141 /*****
00142
00143 enum OSP_ERROR_CODE osp_osire_set_pwm_and_sr (uint16_t deviceAddress,
00144                                              osirePwmData_t data,
00145                                              osireTempStatus_t *p_rsp)
00146 {
00147     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00148     ospCmdBuffer_t ospCmd;
00149     enum OSP_ERROR_CODE ospErrorCode;
00150     errorSpi_t spiError;
00151
00152     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00153
00154     ospCmd.inCmdId = OSP_OSIRE_SET_PWM_SR;
00155     ospCmd.inDeviceAddress = deviceAddress;
00156     ospCmd.p_inParameter = &data.data.pwmData;
00157
00158     ospErrorCode = osp_cmd_buffer (&ospCmd);
00159     if (ospErrorCode != OSP_NO_ERROR)
00160     {
00161         return ospErrorCode;
00162     }
00163
00164     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00165                                                         rspBuffer,
00166                                                         ospCmd.outCmdBufferLength,
00167                                                         ospCmd.outResponseLength);
00168
00169     if (spiError != NO_ERROR_SPI)
00170     {
00171         return OSP_ERROR_SPI;
00172     }
00173
00174     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00175     {
00176         return OSP_ERROR_CRC;
00177     }
00178
00179     for (uint8_t i = 0; i < 2; i++)
00180     {
00181         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00182     }
00183
00184     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00185     return OSP_NO_ERROR;
00186 }
00187
00188 /*****
00189 /*****
00190 enum OSP_ERROR_CODE osp_osire_read_pwm (uint16_t deviceAddress,
00191                                         osirePwmData_t *p_rsp)
00192 {
00193     uint8_t rspBuffer[LENGTH_READ_PWM_RSP]; // response buffer
00194     ospCmdBuffer_t ospCmd;
00195     enum OSP_ERROR_CODE ospErrorCode;
00196     errorSpi_t spiError;
00197
00198     memset (rspBuffer, 0, LENGTH_READ_PWM_RSP);
00199
00200     ospCmd.inCmdId = OSP_OSIRE_READ_PWM;
00201     ospCmd.inDeviceAddress = deviceAddress;
00202     ospCmd.p_inParameter = NULL;
00203
00204     ospErrorCode = osp_cmd_buffer (&ospCmd);
00205     if (ospErrorCode != OSP_NO_ERROR)

```

```

00206     {
00207         return ospErrorCode;
00208     }
00209
00210     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00211                                                         rspBuffer,
00212                                                         ospCmd.outCmdBufferLength,
00213                                                         ospCmd.outResponseLength);
00214
00215     if (spiError != NO_ERROR_SPI)
00216     {
00217         return OSP_ERROR_SPI;
00218     }
00219
00220     if (crc (rspBuffer, LENGTH_READ_PWM_RSP) != 0)
00221     {
00222         return OSP_ERROR_CRC;
00223     }
00224
00225     for (uint8_t i = 0; i < 6; i++)
00226     {
00227         p_rsp->data.pwmData[i] = rspBuffer[8 - i];
00228     }
00229
00230     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00231     return OSP_NO_ERROR;
00232 }
00233
00234 /*****/
00235 /*****/
00236 enum OSP_ERROR_CODE osp_read_otp (uint16_t deviceAddress, uint8_t otpAddress,
00237                                   osireOtpData_t *p_rsp)
00238 {
00239     uint8_t rspBuffer[LENGTH_READ_OTP_RSP]; // response buffer
00240     ospCmdBuffer_t ospCmd;
00241     enum OSP_ERROR_CODE ospErrorCode;
00242     errorSpi_t spiError;
00243
00244     memset (rspBuffer, 0, LENGTH_READ_OTP_RSP);
00245
00246     ospCmd.inCmdId = OSP_OSIRE_READ_OTP;
00247     ospCmd.inDeviceAddress = deviceAddress;
00248     ospCmd.p_inParameter = &otpAddress;
00249
00250     ospErrorCode = osp_cmd_buffer (&ospCmd);
00251     if (ospErrorCode != OSP_NO_ERROR)
00252     {
00253         return ospErrorCode;
00254     }
00255
00256     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00257                                                         rspBuffer,
00258                                                         ospCmd.outCmdBufferLength,
00259                                                         ospCmd.outResponseLength);
00260
00261     if (spiError != NO_ERROR_SPI)
00262     {
00263         return OSP_ERROR_SPI;
00264     }
00265
00266     if (crc (rspBuffer, LENGTH_READ_OTP_RSP) != 0)
00267     {
00268         return OSP_ERROR_CRC;
00269     }
00270
00271     for (uint8_t i = 0; i < 8; i++)
00272     {
00273         p_rsp->data.byte[i] = rspBuffer[10 - i];
00274     }
00275
00276     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00277     return OSP_NO_ERROR;
00278 }
00279
00280 /*****/
00281 /*****/
00282 enum OSP_ERROR_CODE osp_osire_read_otp_complete (uint16_t deviceAddress,
00283                                                  osireOtpDataComplete_t *p_rsp)
00284 {
00285     osireOtpData_t opt; // OTP buffer
00286
00287     for (uint8_t i = 0; i < 0x1F; i = i + 8)
00288     {
00289         enum OSP_ERROR_CODE errorCode = osp_read_otp (deviceAddress, i, &opt);
00290
00291         if (errorCode != OSP_NO_ERROR)
00292         {

```

```

00293         return errorCode;
00294     }
00295
00296     for (uint8_t j = 0; j < 8; j++)
00297     {
00298         p_rsp->data.byte[i + j] = opt.data.byte[j];
00299     }
00300 }
00301
00302 p_rsp->address = opt.address;
00303 return OSP_NO_ERROR;
00304 }
00305
00306 /*****
00307 /*****
00308 enum OSP_ERROR_CODE osp_osire_read_ledstatus (uint16_t deviceAddress,
00309                                              osireLedStatus_t *p_rsp)
00310 {
00311     uint8_t rspBuffer[LENGTH_READ_LEDSTATUS_RSP]; // response buffer
00312     ospCmdBuffer_t ospCmd;
00313     enum OSP_ERROR_CODE ospErrorCode;
00314     errorSpi_t spiError;
00315
00316     memset (rspBuffer, 0, LENGTH_READ_LEDSTATUS_RSP);
00317
00318     ospCmd.inCmdId = OSP_OSIRE_READ_LED_STATUS;
00319     ospCmd.inDeviceAddress = deviceAddress;
00320     ospCmd.p_inParameter = NULL;
00321
00322     ospErrorCode = osp_cmd_buffer (&ospCmd);
00323     if (ospErrorCode != OSP_NO_ERROR)
00324     {
00325         return ospErrorCode;
00326     }
00327
00328     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00329                                                         rspBuffer,
00330                                                         ospCmd.outCmdBufferLength,
00331                                                         ospCmd.outResponseLength);
00332
00333     if (spiError != NO_ERROR_SPI)
00334     {
00335         return OSP_ERROR_SPI;
00336     }
00337
00338     if (crc (rspBuffer, LENGTH_READ_LEDSTATUS_RSP) != 0)
00339     {
00340         return OSP_ERROR_CRC;
00341     }
00342
00343     p_rsp->data.ledStatus = rspBuffer[3];
00344
00345     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00346     return OSP_NO_ERROR;
00347 }
00348
00349 /*****
00350 /*****
00351 enum OSP_ERROR_CODE osp_osire_read_tempstatus (uint16_t deviceAddress,
00352                                              osireTempStatus_t *p_rsp)
00353 {
00354     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00355     ospCmdBuffer_t ospCmd;
00356     enum OSP_ERROR_CODE ospErrorCode;
00357     errorSpi_t spiError;
00358
00359     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00360
00361     ospCmd.inCmdId = OSP_OSIRE_READ_TEMP_STATUS;
00362     ospCmd.inDeviceAddress = deviceAddress;
00363     ospCmd.p_inParameter = NULL;
00364
00365     ospErrorCode = osp_cmd_buffer (&ospCmd);
00366     if (ospErrorCode != OSP_NO_ERROR)
00367     {
00368         return ospErrorCode;
00369     }
00370
00371     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00372                                                         rspBuffer,
00373                                                         ospCmd.outCmdBufferLength,
00374                                                         ospCmd.outResponseLength);
00375
00376     if (spiError != NO_ERROR_SPI)
00377     {
00378         return OSP_ERROR_SPI;
00379     }

```

```

00380
00381     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00382     {
00383         return OSP_ERROR_CRC;
00384     }
00385
00386     for (uint8_t i = 0; i < 2; i++)
00387     {
00388         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00389     }
00390
00391     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00392     return OSP_NO_ERROR;
00393 }
00394
00395 /*****
00396 /*****
00397 enum OSP_ERROR_CODE osp_osire_read_temp (uint16_t deviceAddress,
00398                                         osireTemp_t *p_rsp)
00399 {
00400     uint8_t rspBuffer[LENGTH_READ_TEMP_RSP]; // response buffer
00401     ospCmdBuffer_t ospCmd;
00402     enum OSP_ERROR_CODE ospErrorCode;
00403     errorSpi_t spiError;
00404
00405     memset (rspBuffer, 0, LENGTH_READ_TEMP_RSP);
00406
00407     ospCmd.inCmdId = OSP_OSIRE_READ_TEMP;
00408     ospCmd.inDeviceAddress = deviceAddress;
00409     ospCmd.p_inParameter = NULL;
00410
00411     ospErrorCode = osp_cmd_buffer (&ospCmd);
00412     if (ospErrorCode != OSP_NO_ERROR)
00413     {
00414         return ospErrorCode;
00415     }
00416
00417     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00418                                                         rspBuffer,
00419                                                         ospCmd.outCmdBufferLength,
00420                                                         ospCmd.outResponseLength);
00421
00422     if (spiError != NO_ERROR_SPI)
00423     {
00424         return OSP_ERROR_SPI;
00425     }
00426
00427     if (crc (rspBuffer, LENGTH_READ_TEMP_RSP) != 0)
00428     {
00429         return OSP_ERROR_CRC;
00430     }
00431
00432     p_rsp->data.temp_value = rspBuffer[FIRST_BYTE_PAYLOAD];
00433
00434     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00435     return OSP_NO_ERROR;
00436 }
00437
00438 /*****
00439 /*****
00440 enum OSP_ERROR_CODE osp_osire_set_otth (uint16_t deviceAddress,
00441                                         osireOtthData_t data)
00442 {
00443     ospCmdBuffer_t ospCmd;
00444     enum OSP_ERROR_CODE ospErrorCode;
00445     errorSpi_t spiError;
00446
00447     ospCmd.inCmdId = OSP_OSIRE_SET_OTTH;
00448     ospCmd.inDeviceAddress = deviceAddress;
00449     ospCmd.p_inParameter = &data.data.otthData;
00450
00451     ospErrorCode = osp_cmd_buffer (&ospCmd);
00452     if (ospErrorCode != OSP_NO_ERROR)
00453     {
00454         return ospErrorCode;
00455     }
00456
00457     spiError = send_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00458                                             ospCmd.outCmdBufferLength);
00459
00460     if (spiError != NO_ERROR_SPI)
00461     {
00462         return OSP_ERROR_SPI;
00463     }
00464
00465     return OSP_NO_ERROR;
00466 }

```

```

00467
00468 /*****
00469 /*****
00470 enum OSP_ERROR_CODE osp_osire_set_otth_and_sr (uint16_t deviceAddress,
00471                                             osireOtthData_t data,
00472                                             osireTempStatus_t *p_rsp)
00473 {
00474     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00475     ospCmdBuffer_t ospCmd;
00476     enum OSP_ERROR_CODE ospErrorCode;
00477     errorSpi_t spiError;
00478
00479     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00480
00481     ospCmd.inCmdId = OSP_OSIRE_SET_OTTH_SR;
00482     ospCmd.inDeviceAddress = deviceAddress;
00483     ospCmd.p_inParameter = &data.data.otthData;
00484
00485     ospErrorCode = osp_cmd_buffer (&ospCmd);
00486     if (ospErrorCode != OSP_NO_ERROR)
00487     {
00488         return ospErrorCode;
00489     }
00490
00491     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00492                                                         rspBuffer,
00493                                                         ospCmd.outCmdBufferLength,
00494                                                         ospCmd.outResponseLength);
00495
00496     if (spiError != NO_ERROR_SPI)
00497     {
00498         return OSP_ERROR_SPI;
00499     }
00500
00501     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00502     {
00503         return OSP_ERROR_CRC;
00504     }
00505
00506     for (uint8_t i = 0; i < 2; i++)
00507     {
00508         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00509     }
00510
00511     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00512     return OSP_NO_ERROR;
00513 }
00514
00515 /*****
00516 /*****
00517 enum OSP_ERROR_CODE osp_osire_read_otth (uint16_t deviceAddress,
00518                                         osireOtthData_t *p_rsp)
00519 {
00520     uint8_t rspBuffer[LENGTH_READ_OTTH_RSP]; // response buffer
00521     ospCmdBuffer_t ospCmd;
00522     enum OSP_ERROR_CODE ospErrorCode;
00523     errorSpi_t spiError;
00524
00525     memset (rspBuffer, 0, LENGTH_READ_OTTH_RSP);
00526
00527     ospCmd.inCmdId = OSP_OSIRE_READ_OTTH;
00528     ospCmd.inDeviceAddress = deviceAddress;
00529     ospCmd.p_inParameter = NULL;
00530
00531     ospErrorCode = osp_cmd_buffer (&ospCmd);
00532     if (ospErrorCode != OSP_NO_ERROR)
00533     {
00534         return ospErrorCode;
00535     }
00536
00537     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00538                                                         rspBuffer,
00539                                                         ospCmd.outCmdBufferLength,
00540                                                         ospCmd.outResponseLength);
00541
00542     if (spiError != NO_ERROR_SPI)
00543     {
00544         return OSP_ERROR_SPI;
00545     }
00546
00547     if (crc (rspBuffer, LENGTH_READ_OTTH_RSP) != 0)
00548     {
00549         return OSP_ERROR_CRC;
00550     }
00551
00552     for (uint8_t i = 0; i < 3; i++)
00553     {

```

```

00554     p_rsp->data.otthData[i] = rspBuffer[5 - i];
00555 }
00556
00557 p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00558 return OSP_NO_ERROR;
00559 }
00560
00561 /*****
00562 *****/
00563 enum OSP_ERROR_CODE osp_go_active_and_sr (uint16_t deviceAddress,
00564                                           osireTempStatus_t *p_rsp)
00565 {
00566     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00567     ospCmdBuffer_t ospCmd;
00568     enum OSP_ERROR_CODE ospErrorCode;
00569     errorSpi_t spiError;
00570
00571     ospCmd.inCmdId = OSP_OSIRE_GO_ACTIVE_SR;
00572     ospCmd.inDeviceAddress = deviceAddress;
00573     ospCmd.p_inParameter = NULL;
00574
00575     ospErrorCode = osp_cmd_buffer (&ospCmd);
00576     if (ospErrorCode != OSP_NO_ERROR)
00577     {
00578         return ospErrorCode;
00579     }
00580
00581     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00582                                                         rspBuffer,
00583                                                         ospCmd.outCmdBufferLength,
00584                                                         ospCmd.outResponseLength);
00585
00586     if (spiError != NO_ERROR_SPI)
00587     {
00588         return OSP_ERROR_SPI;
00589     }
00590
00591     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00592     {
00593         return OSP_ERROR_CRC;
00594     }
00595
00596     for (uint8_t i = 0; i < 2; i++)
00597     {
00598         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00599     }
00600
00601     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00602     return OSP_NO_ERROR;
00603 }
00604
00605 /*****
00606 *****/
00607 enum OSP_ERROR_CODE osp_osire_go_sleep_and_sr (uint16_t deviceAddress,
00608                                                 osireTempStatus_t *p_rsp)
00609 {
00610     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00611     ospCmdBuffer_t ospCmd;
00612     enum OSP_ERROR_CODE ospErrorCode;
00613     errorSpi_t spiError;
00614
00615     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00616
00617     ospCmd.inCmdId = OSP_OSIRE_GO_SLEEP_SR;
00618     ospCmd.inDeviceAddress = deviceAddress;
00619     ospCmd.p_inParameter = NULL;
00620
00621     ospErrorCode = osp_cmd_buffer (&ospCmd);
00622     if (ospErrorCode != OSP_NO_ERROR)
00623     {
00624         return ospErrorCode;
00625     }
00626
00627     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00628                                                         rspBuffer,
00629                                                         ospCmd.outCmdBufferLength,
00630                                                         ospCmd.outResponseLength);
00631
00632     if (spiError != NO_ERROR_SPI)
00633     {
00634         return OSP_ERROR_SPI;
00635     }
00636
00637     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00638     {
00639         return OSP_ERROR_CRC;
00640     }

```



```

00641
00642     for (uint8_t i = 0; i < 2; i++)
00643     {
00644         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00645     }
00646
00647     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00648     return OSP_NO_ERROR;
00649 }
00650
00651 /*****
00652 /*****
00653 enum OSP_ERROR_CODE osp_osire_go_deep_sleep_and_sr (uint16_t deviceAddress,
00654                                                     osireTempStatus_t *p_rsp)
00655 {
00656     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // response buffer
00657     ospCmdBuffer_t ospCmd;
00658     enum OSP_ERROR_CODE ospErrorCode;
00659     errorSpi_t spiError;
00660
00661     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00662
00663     ospCmd.inCmdId = OSP_OSIRE_GO_DEEP_SLEEP_SR;
00664     ospCmd.inDeviceAddress = deviceAddress;
00665     ospCmd.p_inParameter = NULL;
00666
00667     ospErrorCode = osp_cmd_buffer (&ospCmd);
00668     if (ospErrorCode != OSP_NO_ERROR)
00669     {
00670         return ospErrorCode;
00671     }
00672
00673     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00674                                                         rspBuffer,
00675                                                         ospCmd.outCmdBufferLength,
00676                                                         ospCmd.outResponseLength);
00677
00678     if (spiError != NO_ERROR_SPI)
00679     {
00680         return OSP_ERROR_SPI;
00681     }
00682
00683     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00684     {
00685         return OSP_ERROR_CRC;
00686     }
00687
00688     for (uint8_t i = 0; i < 2; i++)
00689     {
00690         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00691     }
00692
00693     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00694     return OSP_NO_ERROR;
00695 }
00696
00697 /*****
00698 /*****
00699 enum OSP_ERROR_CODE osp_osire_clr_error_and_sr (uint16_t deviceAddress,
00700                                                  osireTempStatus_t *p_rsp)
00701 {
00702     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // message buffer
00703     ospCmdBuffer_t ospCmd;
00704     enum OSP_ERROR_CODE ospErrorCode;
00705     errorSpi_t spiError;
00706
00707     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00708
00709     ospCmd.inCmdId = OSP_OSIRE_CLR_ERROR_SR;
00710     ospCmd.inDeviceAddress = deviceAddress;
00711     ospCmd.p_inParameter = NULL;
00712
00713     ospErrorCode = osp_cmd_buffer (&ospCmd);
00714     if (ospErrorCode != OSP_NO_ERROR)
00715     {
00716         return ospErrorCode;
00717     }
00718
00719     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00720                                                         rspBuffer,
00721                                                         ospCmd.outCmdBufferLength,
00722                                                         ospCmd.outResponseLength);
00723
00724     if (spiError != NO_ERROR_SPI)
00725     {
00726         return OSP_ERROR_SPI;
00727     }

```

```

00728
00729     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00730     {
00731         return OSP_ERROR_CRC;
00732     }
00733
00734     for (uint8_t i = 0; i < 2; i++)
00735     {
00736         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00737     }
00738
00739     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00740     return OSP_NO_ERROR;
00741 }
00742
00743 /*****
00744 /*****
00745 enum OSP_ERROR_CODE osp_osire_p4error_bidir (uint16_t deviceAddress,
00746                                             osireTempStatus_t *p_rsp)
00747 {
00748     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // message buffer
00749     ospCmdBuffer_t ospCmd;
00750     enum OSP_ERROR_CODE ospErrorCode;
00751     errorSpi_t spiError;
00752
00753     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00754
00755     ospCmd.inCmdId = OSP_OSIRE_P4ERROR_BIDIR;
00756     ospCmd.inDeviceAddress = deviceAddress;
00757     ospCmd.p_inParameter = NULL;
00758
00759     ospErrorCode = osp_cmd_buffer (&ospCmd);
00760     if (ospErrorCode != OSP_NO_ERROR)
00761     {
00762         return ospErrorCode;
00763     }
00764
00765     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00766                                                         rspBuffer,
00767                                                         ospCmd.outCmdBufferLength,
00768                                                         ospCmd.outResponseLength);
00769
00770     if (spiError != NO_ERROR_SPI)
00771     {
00772         return OSP_ERROR_SPI;
00773     }
00774
00775     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00776     {
00777         return OSP_ERROR_CRC;
00778     }
00779
00780     for (uint8_t i = 0; i < 2; i++)
00781     {
00782         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00783     }
00784
00785     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00786     return OSP_NO_ERROR;
00787 }
00788 /*****
00789 /*****
00790 enum OSP_ERROR_CODE osp_osire_p4error_loop (uint16_t deviceAddress,
00791                                             osireTempStatus_t *p_rsp)
00792 {
00793     uint8_t rspBuffer[LENGTH_READ_TEMPSTATUS_RSP]; // message buffer
00794     ospCmdBuffer_t ospCmd;
00795     enum OSP_ERROR_CODE ospErrorCode;
00796     errorSpi_t spiError;
00797
00798     memset (rspBuffer, 0, LENGTH_READ_TEMPSTATUS_RSP);
00799
00800     ospCmd.inCmdId = OSP_OSIRE_P4ERROR_LOOP;
00801     ospCmd.inDeviceAddress = deviceAddress;
00802     ospCmd.p_inParameter = NULL;
00803
00804     ospErrorCode = osp_cmd_buffer (&ospCmd);
00805     if (ospErrorCode != OSP_NO_ERROR)
00806     {
00807         return ospErrorCode;
00808     }
00809
00810     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00811                                                         rspBuffer,
00812                                                         ospCmd.outCmdBufferLength,
00813                                                         ospCmd.outResponseLength);
00814

```

```

00815     if (spiError != NO_ERROR_SPI)
00816     {
00817         return OSP_ERROR_SPI;
00818     }
00819
00820     if (crc (rspBuffer, LENGTH_READ_TEMPSTATUS_RSP) != 0)
00821     {
00822         return OSP_ERROR_CRC;
00823     }
00824
00825     for (uint8_t i = 0; i < 2; i++)
00826     {
00827         p_rsp->data.tempStatus[i] = rspBuffer[4 - i];
00828     }
00829
00830     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00831     return OSP_NO_ERROR;
00832 }
00833 /*****
00834 /*****
00835 enum OSP_ERROR_CODE osp_osire_read_setup (uint16_t deviceAddress,
00836                                         osireSetSetupData_t *p_rsp)
00837 {
00838     uint8_t rspBuffer[LENGTH_READ_SETUP_RSP]; // response buffer
00839     ospCmdBuffer_t ospCmd;
00840     enum OSP_ERROR_CODE ospErrorCode;
00841     errorSpi_t spiError;
00842
00843     memset (rspBuffer, 0, LENGTH_READ_SETUP_RSP);
00844
00845     ospCmd.inCmdId = OSP_OSIRE_READ_SETUP;
00846     ospCmd.inDeviceAddress = deviceAddress;
00847     ospCmd.p_inParameter = NULL;
00848
00849     ospErrorCode = osp_cmd_buffer (&ospCmd);
00850     if (ospErrorCode != OSP_NO_ERROR)
00851     {
00852         return ospErrorCode;
00853     }
00854
00855     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00856                                                         rspBuffer,
00857                                                         ospCmd.outCmdBufferLength,
00858                                                         ospCmd.outResponseLength);
00859
00860     if (spiError != NO_ERROR_SPI)
00861     {
00862         return OSP_ERROR_SPI;
00863     }
00864
00865     if (crc (rspBuffer, LENGTH_READ_SETUP_RSP) != 0)
00866     {
00867         return OSP_ERROR_CRC;
00868     }
00869
00870     p_rsp->data.setupData = rspBuffer[FIRST_BYTE_PAYLOAD];
00871
00872     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00873     return OSP_NO_ERROR;
00874 }
00875 /*****
00876 /*****
00877 enum OSP_ERROR_CODE osp_osire_read_comstatus (uint16_t deviceAddress,
00878                                               osireComStatus_t *p_rsp)
00879 {
00880     uint8_t rspBuffer[LENGTH_READ_COMSTATUS_RSP]; // response buffer
00881     ospCmdBuffer_t ospCmd;
00882     enum OSP_ERROR_CODE ospErrorCode;
00883     errorSpi_t spiError;
00884
00885     memset (rspBuffer, 0, LENGTH_READ_COMSTATUS_RSP);
00886
00887     ospCmd.inCmdId = OSP_OSIRE_READ_COM_STATUS;
00888     ospCmd.inDeviceAddress = deviceAddress;
00889     ospCmd.p_inParameter = NULL;
00890
00891     ospErrorCode = osp_cmd_buffer (&ospCmd);
00892     if (ospErrorCode != OSP_NO_ERROR)
00893     {
00894         return ospErrorCode;
00895     }
00896
00897     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00898                                                         rspBuffer,
00899                                                         ospCmd.outCmdBufferLength,
00900                                                         ospCmd.outResponseLength);
00901

```

```

00902
00903     if (spiError != NO_ERROR_SPI)
00904     {
00905         return OSP_ERROR_SPI;
00906     }
00907
00908     if (crc (rspBuffer, LENGTH_READ_COMSTATUS_RSP) != 0)
00909     {
00910         return OSP_ERROR_CRC;
00911     }
00912
00913     p_rsp->data.comStatus = rspBuffer[FIRST_BYTE_PAYLOAD];
00914
00915     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00916     return OSP_NO_ERROR;
00917 }
00918
00919 /*****
00920 /*****
00921 enum OSP_ERROR_CODE osp_osire_read_status (uint16_t deviceAddress,
00922                                           osireStatus_t *p_rsp)
00923 {
00924     uint8_t rspBuffer[LENGTH_READ_STATUS_RSP]; // response buffer
00925     ospCmdBuffer_t ospCmd;
00926     enum OSP_ERROR_CODE ospErrorCode;
00927     errorSpi_t spiError;
00928
00929     memset (rspBuffer, 0, LENGTH_READ_STATUS_RSP);
00930
00931     ospCmd.inCmdId = OSP_OSIRE_READ_STATUS;
00932     ospCmd.inDeviceAddress = deviceAddress;
00933     ospCmd.p_inParameter = NULL;
00934
00935     ospErrorCode = osp_cmd_buffer (&ospCmd);
00936     if (ospErrorCode != OSP_NO_ERROR)
00937     {
00938         return ospErrorCode;
00939     }
00940
00941     spiError = send_and_receive_data_over_spi_blocking (ospCmd.p_outCmdBuffer,
00942                                                         rspBuffer,
00943                                                         ospCmd.outCmdBufferLength,
00944                                                         ospCmd.outResponseLength);
00945
00946     if (spiError != NO_ERROR_SPI)
00947     {
00948         return OSP_ERROR_SPI;
00949     }
00950
00951     if (crc (rspBuffer, LENGTH_READ_STATUS_RSP) != 0)
00952     {
00953         return OSP_ERROR_CRC;
00954     }
00955
00956     p_rsp->data.status = rspBuffer[FIRST_BYTE_PAYLOAD];
00957
00958     p_rsp->address = ((rspBuffer[0] & 0x0F) << 6) | ((rspBuffer[1] >> 2) & 0x3F);
00959     return OSP_NO_ERROR;
00960 }
00961

```

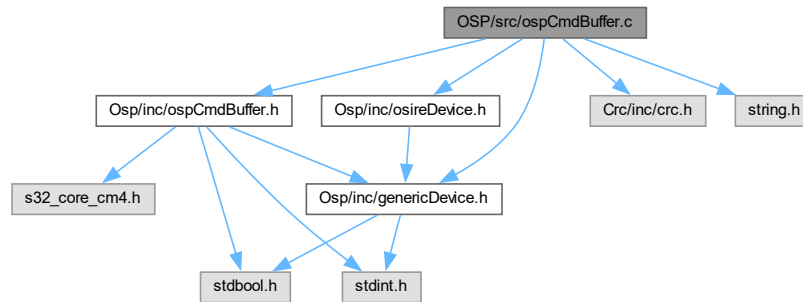
5.12 OSP/src/ospCmdBuffer.c File Reference

```

#include <Osp/inc/genericDevice.h>
#include <Osp/inc/osireDevice.h>
#include <Crc/inc/crc.h>
#include <Osp/inc/ospCmdBuffer.h>
#include <string.h>

```

Include dependency graph for ospCmdBuffer.c:



Macros

- #define [MAX_CMD_BUFFER_SIZE](#) 16

Functions

- [START_FUNCTION_DEFINITION_RAMSECTION](#) enum [OSP_ERROR_CODE](#) [osp_cmd_buffer](#) ([ospCmdBuffer_t](#) *p_cmdInfo)

5.12.1 Macro Definition Documentation

5.12.1.1 MAX_CMD_BUFFER_SIZE

```
#define MAX_CMD_BUFFER_SIZE 16
```

Definition at line 27 of file [ospCmdBuffer.c](#).

5.12.2 Function Documentation

5.12.2.1 osp_cmd_buffer()

```
START_FUNCTION_DEFINITION_RAMSECTION enum OSP\_ERROR\_CODE osp_cmd_buffer (
    ospCmdBuffer\_t * p_cmdInfo )
```

Definition at line 33 of file [ospCmdBuffer.c](#).

```

00034 {
00035     memset (cmdBuffer, 0, MAX\_CMD\_BUFFER\_SIZE);
00036
00037     switch (p_cmdInfo->inCmdId)
00038     {
00039         /*****
00040         // for genericDevice.c
00041         *****/
00042         case (OSP\_INIT\_BIDIR) :
00043         {
00044             if (p_cmdInfo->p_inParameter != NULL)
00045             {
00046                 return OSP\_ERROR\_PARAMETER;
00047             }
00048             if (p_cmdInfo->inDeviceAddress != 1)
00049             {
00050                 return OSP\_ADDRESS\_ERROR;
00051             }
00052
00053             build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00054                 LENGTH\_INIT\_MSG);
00055         }
00056     }
00057 }
```

```

00056         cmdBuffer[LENGTH_INIT_MSG - 1] = crc (cmdBuffer,
00057         LENGTH_INIT_MSG - 1);
00058
00059         p_cmdInfo->outCmdBufferLength = LENGTH_INIT_MSG;
00060         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00061         p_cmdInfo->outResponseLength = LENGTH_INIT_RSP; // response expected
00062         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00063         break;
00064     }
00065
00066     case (OSP_INIT_LOOP):
00067     {
00068
00069         if (p_cmdInfo->p_inParameter != NULL)
00070         {
00071             return OSP_ERROR_PARAMETER;
00072         }
00073         if (p_cmdInfo->inDeviceAddress != 1)
00074         {
00075             return OSP_ADDRESS_ERROR;
00076         }
00077
00078         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00079         LENGTH_INIT_MSG);
00080
00081         cmdBuffer[LENGTH_INIT_MSG - 1] = crc (cmdBuffer,
00082         LENGTH_INIT_MSG - 1);
00083
00084         p_cmdInfo->outCmdBufferLength = LENGTH_INIT_MSG;
00085         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00086         p_cmdInfo->outResponseLength = LENGTH_INIT_RSP; // response expected
00087         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00088
00089         break;
00090     }
00091
00092     case (OSP_RESET):
00093     {
00094
00095         if (p_cmdInfo->p_inParameter != NULL)
00096         {
00097             return OSP_ERROR_PARAMETER;
00098         }
00099         if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00100         {
00101             return OSP_ADDRESS_ERROR;
00102         }
00103
00104         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00105         LENGTH_RESET_MSG);
00106
00107         cmdBuffer[LENGTH_RESET_MSG - 1] = crc (cmdBuffer,
00108         LENGTH_RESET_MSG - 1);
00109
00110         p_cmdInfo->outCmdBufferLength = LENGTH_RESET_MSG;
00111         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00112         p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00113         p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00114
00115         break;
00116     }
00117
00118     case (OSP_GO_ACTIVE):
00119     {
00120
00121         if (p_cmdInfo->p_inParameter != NULL)
00122         {
00123             return OSP_ERROR_PARAMETER;
00124         }
00125         if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00126         {
00127             return OSP_ADDRESS_ERROR;
00128         }
00129
00130         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00131         LENGTH_GO_ACTIVE_MSG);
00132
00133         cmdBuffer[LENGTH_GO_ACTIVE_MSG - 1] = crc (cmdBuffer,
00134         LENGTH_GO_ACTIVE_MSG - 1);
00135
00136         p_cmdInfo->outCmdBufferLength = LENGTH_GO_ACTIVE_MSG;
00137         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00138         p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00139         p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00140
00141         break;
00142     }

```

```

00143
00144     case (OSP_GO_SLEEP):
00145     {
00146
00147         if (p_cmdInfo->p_inParameter != NULL)
00148         {
00149             return OSP_ERROR_PARAMETER;
00150         }
00151         if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00152         {
00153             return OSP_ADDRESS_ERROR;
00154         }
00155
00156         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00157             LENGTH_GO_SLEEP_MSG);
00158
00159         cmdBuffer[LENGTH_GO_SLEEP_MSG - 1] = crc (cmdBuffer,
00160             LENGTH_GO_SLEEP_MSG - 1);
00161
00162         p_cmdInfo->outCmdBufferLength = LENGTH_GO_SLEEP_MSG;
00163         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00164         p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00165         p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00166
00167         break;
00168     }
00169
00170     case (OSP_GO_DEEP_SLEEP):
00171     {
00172
00173         if (p_cmdInfo->p_inParameter != NULL)
00174         {
00175             return OSP_ERROR_PARAMETER;
00176         }
00177         if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00178         {
00179             return OSP_ADDRESS_ERROR;
00180         }
00181
00182         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00183             LENGTH_GO_DEEP_SLEEP_MSG);
00184
00185         cmdBuffer[LENGTH_GO_DEEP_SLEEP_MSG - 1] = crc (cmdBuffer,
00186             LENGTH_GO_DEEP_SLEEP_MSG - 1);
00187
00188         p_cmdInfo->outCmdBufferLength = LENGTH_GO_DEEP_SLEEP_MSG;
00189         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00190         p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00191         p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00192
00193         break;
00194     }
00195
00196     case (OSP_CLR_ERROR):
00197     {
00198
00199         if (p_cmdInfo->p_inParameter != NULL)
00200         {
00201             return OSP_ERROR_PARAMETER;
00202         }
00203         if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00204         {
00205             return OSP_ADDRESS_ERROR;
00206         }
00207
00208         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00209             LENGTH_CLR_ERROR_MSG);
00210
00211         cmdBuffer[LENGTH_CLR_ERROR_MSG - 1] = crc (cmdBuffer,
00212             LENGTH_CLR_ERROR_MSG - 1);
00213
00214         p_cmdInfo->outCmdBufferLength = LENGTH_CLR_ERROR_MSG;
00215         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00216         p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00217         p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00218
00219         break;
00220     }
00221     END_FUNCTION_DEFINITION_RAMSECTION
00222     /*****
00223     // for osireDevice.c
00224     *****/
00225
00226     case (OSP_OSIRE_SET_SETUP):
00227     {
00228
00229         if (p_cmdInfo->p_inParameter == NULL)

```

```

00230         {
00231             return OSP_ERROR_PARAMETER;
00232         }
00233     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00234     {
00235         return OSP_ADDRESS_ERROR;
00236     }
00237
00238     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00239                 LENGTH_SET_SETUP_MSG);
00240
00241     osireSetSetupData_t *p_data;
00242     p_data = (osireSetSetupData_t*) p_cmdInfo->p_inParameter;
00243     cmdBuffer[3] = p_data->data.setupData;
00244
00245     cmdBuffer[LENGTH_SET_SETUP_MSG - 1] = crc (cmdBuffer,
00246         LENGTH_SET_SETUP_MSG - 1);
00247
00248     p_cmdInfo->outCmdBufferLength = LENGTH_SET_SETUP_MSG;
00249     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00250     p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00251     p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00252
00253     break;
00254 }
00255
00256 case (OSP_OSIRE_SET_SETUP_SR):
00257 {
00258     if (p_cmdInfo->p_inParameter == NULL)
00259     {
00260         return OSP_ERROR_PARAMETER;
00261     }
00262     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00263         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00264     {
00265         return OSP_ADDRESS_ERROR;
00266     }
00267
00268     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00269                 LENGTH_SET_SETUP_MSG);
00270
00271     osireSetSetupData_t *p_data;
00272     p_data = (osireSetSetupData_t*) p_cmdInfo->p_inParameter;
00273     cmdBuffer[3] = p_data->data.setupData;
00274
00275     cmdBuffer[LENGTH_SET_SETUP_MSG - 1] = crc (cmdBuffer,
00276         LENGTH_SET_SETUP_MSG - 1);
00277
00278     p_cmdInfo->outCmdBufferLength = LENGTH_SET_SETUP_MSG;
00279     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00280     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00281     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00282
00283     break;
00284 }
00285
00286 case (OSP_OSIRE_SET_PWM):
00287 {
00288     if (p_cmdInfo->p_inParameter == NULL)
00289     {
00290         return OSP_ERROR_PARAMETER;
00291     }
00292     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00293     {
00294         return OSP_ADDRESS_ERROR;
00295     }
00296
00297     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00298                 LENGTH_SET_PWM_MSG);
00299
00300     osirePwmData_t *p_data;
00301     p_data = (osirePwmData_t*) p_cmdInfo->p_inParameter;
00302     cmdBuffer[8] = p_data->data.pwmData[0];
00303     cmdBuffer[7] = p_data->data.pwmData[1];
00304     cmdBuffer[6] = p_data->data.pwmData[2];
00305     cmdBuffer[5] = p_data->data.pwmData[3];
00306     cmdBuffer[4] = p_data->data.pwmData[4];
00307     cmdBuffer[3] = p_data->data.pwmData[5];
00308
00309     cmdBuffer[LENGTH_SET_PWM_MSG - 1] = crc (cmdBuffer,
00310         LENGTH_SET_PWM_MSG - 1);
00311
00312     p_cmdInfo->outCmdBufferLength = LENGTH_SET_PWM_MSG;
00313     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00314     p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00315
00316

```



```

00317     p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00318
00319     break;
00320 }
00321
00322 case (OSP_OSIRE_SET_PWM_SR):
00323 {
00324
00325     if (p_cmdInfo->p_inParameter == NULL)
00326     {
00327         return OSP_ERROR_PARAMETER;
00328     }
00329     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00330         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00331     {
00332         return OSP_ADDRESS_ERROR;
00333     }
00334
00335     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00336                 LENGTH_SET_PWM_MSG);
00337
00338     osirePwmData_t *p_data;
00339     p_data = (osirePwmData_t*) p_cmdInfo->p_inParameter;
00340     cmdBuffer[8] = p_data->data.pwmData[0];
00341     cmdBuffer[7] = p_data->data.pwmData[1];
00342     cmdBuffer[6] = p_data->data.pwmData[2];
00343     cmdBuffer[5] = p_data->data.pwmData[3];
00344     cmdBuffer[4] = p_data->data.pwmData[4];
00345     cmdBuffer[3] = p_data->data.pwmData[5];
00346
00347     cmdBuffer[LENGTH_SET_PWM_MSG - 1] = crc (cmdBuffer,
00348         LENGTH_SET_PWM_MSG - 1);
00349
00350     p_cmdInfo->outCmdBufferLength = LENGTH_SET_PWM_MSG;
00351     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00352     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00353     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00354
00355     break;
00356 }
00357
00358 case (OSP_OSIRE_READ_PWM):
00359 {
00360
00361     if (p_cmdInfo->p_inParameter != NULL)
00362     {
00363         return OSP_ERROR_PARAMETER;
00364     }
00365     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00366         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00367     {
00368         return OSP_ADDRESS_ERROR;
00369     }
00370
00371     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00372                 LENGTH_READ_PWM_MSG);
00373
00374     cmdBuffer[LENGTH_READ_PWM_MSG - 1] = crc (cmdBuffer,
00375         LENGTH_READ_PWM_MSG - 1);
00376
00377     p_cmdInfo->outCmdBufferLength = LENGTH_READ_PWM_MSG;
00378     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00379     p_cmdInfo->outResponseLength = LENGTH_READ_PWM_RSP; // response expected
00380     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00381
00382     break;
00383 }
00384
00385 case (OSP_OSIRE_READ_OTP):
00386 {
00387
00388     if (p_cmdInfo->p_inParameter == NULL)
00389     {
00390         return OSP_ERROR_PARAMETER;
00391     }
00392     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00393         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00394     {
00395         return OSP_ADDRESS_ERROR;
00396     }
00397
00398     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00399                 LENGTH_READ_OTP_MSG);
00400
00401     //Offset in OTP memory
00402     cmdBuffer[3] = (*(uint8_t*) p_cmdInfo->p_inParameter);
00403

```

```

00404     cmdBuffer[LENGTH_READ_OTP_MSG - 1] = crc (cmdBuffer,
00405     LENGTH_READ_OTP_MSG - 1);
00406
00407     p_cmdInfo->outCmdBufferLength = LENGTH_READ_OTP_MSG;
00408     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00409     p_cmdInfo->outResponseLength = LENGTH_READ_OTP_RSP; // response expected
00410     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00411
00412     break;
00413 }
00414
00415 case (OSP_OSIRE_READ_LED_STATUS):
00416 {
00417     if (p_cmdInfo->p_inParameter != NULL)
00418     {
00419         return OSP_ERROR_PARAMETER;
00420     }
00421     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00422         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00423     {
00424         return OSP_ADDRESS_ERROR;
00425     }
00426
00427     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00428     LENGTH_READ_LEDSTATUS_MSG);
00429
00430     cmdBuffer[LENGTH_READ_LEDSTATUS_MSG - 1] = crc (cmdBuffer,
00431     LENGTH_READ_LEDSTATUS_MSG - 1);
00432
00433     p_cmdInfo->outCmdBufferLength = LENGTH_READ_LEDSTATUS_MSG;
00434     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00435     p_cmdInfo->outResponseLength = LENGTH_READ_LEDSTATUS_RSP; // response expected
00436     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00437
00438     break;
00439 }
00440
00441 case (OSP_OSIRE_READ_TEMP_STATUS):
00442 {
00443     if (p_cmdInfo->p_inParameter != NULL)
00444     {
00445         return OSP_ERROR_PARAMETER;
00446     }
00447     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00448         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00449     {
00450         return OSP_ADDRESS_ERROR;
00451     }
00452
00453     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00454     LENGTH_READ_TEMPSTATUS_MSG);
00455
00456     cmdBuffer[LENGTH_READ_TEMPSTATUS_MSG - 1] = crc (cmdBuffer,
00457     LENGTH_READ_TEMPSTATUS_MSG - 1);
00458
00459     p_cmdInfo->outCmdBufferLength = LENGTH_READ_TEMPSTATUS_MSG;
00460     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00461     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00462     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00463
00464     break;
00465 }
00466
00467 case (OSP_OSIRE_READ_TEMP):
00468 {
00469     if (p_cmdInfo->p_inParameter != NULL)
00470     {
00471         return OSP_ERROR_PARAMETER;
00472     }
00473     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00474         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00475     {
00476         return OSP_ADDRESS_ERROR;
00477     }
00478
00479     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00480     LENGTH_READ_TEMP_MSG);
00481
00482     cmdBuffer[LENGTH_READ_TEMP_MSG - 1] = crc (cmdBuffer,
00483     LENGTH_READ_TEMP_MSG - 1);
00484
00485     p_cmdInfo->outCmdBufferLength = LENGTH_READ_TEMP_MSG;
00486     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00487     p_cmdInfo->outResponseLength = LENGTH_READ_TEMP_RSP; // response expected
00488
00489     break;
00490 }

```

```

00491     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00492
00493     break;
00494 }
00495
00496 case (OSP_OSIRE_SET_OTTH):
00497 {
00498
00499     if (p_cmdInfo->p_inParameter == NULL)
00500     {
00501         return OSP_ERROR_PARAMETER;
00502     }
00503     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00504     {
00505         return OSP_ADDRESS_ERROR;
00506     }
00507
00508     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00509                 LENGTH_SET_OTTH_MSG);
00510
00511     osireOtthData_t *p_data;
00512     p_data = (osireOtthData_t*) p_cmdInfo->p_inParameter;
00513     cmdBuffer[5] = p_data->data.otthData[0];
00514     cmdBuffer[4] = p_data->data.otthData[1];
00515     cmdBuffer[3] = p_data->data.otthData[2];
00516
00517     cmdBuffer[LENGTH_SET_OTTH_MSG - 1] = crc (cmdBuffer,
00518         LENGTH_SET_OTTH_MSG - 1);
00519
00520     p_cmdInfo->outCmdBufferLength = LENGTH_SET_OTTH_MSG;
00521     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00522     p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00523     p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00524
00525     break;
00526 }
00527
00528 case (OSP_OSIRE_SET_OTTH_SR):
00529 {
00530
00531     if (p_cmdInfo->p_inParameter == NULL)
00532     {
00533         return OSP_ERROR_PARAMETER;
00534     }
00535     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00536         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00537     {
00538         return OSP_ADDRESS_ERROR;
00539     }
00540
00541     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00542                 LENGTH_SET_OTTH_MSG);
00543
00544     osireOtthData_t *p_data;
00545     p_data = (osireOtthData_t*) p_cmdInfo->p_inParameter;
00546     cmdBuffer[5] = p_data->data.otthData[0];
00547     cmdBuffer[4] = p_data->data.otthData[1];
00548     cmdBuffer[3] = p_data->data.otthData[2];
00549
00550     cmdBuffer[LENGTH_SET_OTTH_MSG - 1] = crc (cmdBuffer,
00551         LENGTH_SET_OTTH_MSG - 1);
00552
00553     p_cmdInfo->outCmdBufferLength = LENGTH_SET_OTTH_MSG;
00554     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00555     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00556     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00557
00558     break;
00559 }
00560
00561 case (OSP_OSIRE_READ_OTTH):
00562 {
00563
00564     if (p_cmdInfo->p_inParameter != NULL)
00565     {
00566         return OSP_ERROR_PARAMETER;
00567     }
00568     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00569         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00570     {
00571         return OSP_ADDRESS_ERROR;
00572     }
00573
00574     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00575                 LENGTH_READ_OTTH_MSG);
00576
00577     cmdBuffer[LENGTH_READ_OTTH_MSG - 1] = crc (cmdBuffer,

```

```

00578     LENGTH_READ_OTTH_MSG - 1);
00579
00580     p_cmdInfo->outCmdBufferLength = LENGTH_READ_OTTH_MSG;
00581     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00582     p_cmdInfo->outResponseLength = LENGTH_READ_OTTH_RSP; // response expected
00583     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00584
00585     break;
00586 }
00587
00588 case (OSP_OSIRE_GO_ACTIVE_SR):
00589 {
00590
00591     if (p_cmdInfo->p_inParameter != NULL)
00592     {
00593         return OSP_ERROR_PARAMETER;
00594     }
00595     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00596     {
00597         return OSP_ADDRESS_ERROR;
00598     }
00599
00600     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00601                 LENGTH_GO_ACTIVE_MSG);
00602
00603     cmdBuffer[LENGTH_GO_ACTIVE_MSG - 1] = crc (cmdBuffer,
00604         LENGTH_GO_ACTIVE_MSG - 1);
00605
00606     p_cmdInfo->outCmdBufferLength = LENGTH_GO_ACTIVE_MSG;
00607     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00608     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00609     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00610
00611     break;
00612 }
00613
00614 case (OSP_OSIRE_GO_SLEEP_SR):
00615 {
00616
00617     if (p_cmdInfo->p_inParameter != NULL)
00618     {
00619         return OSP_ERROR_PARAMETER;
00620     }
00621     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00622         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00623     {
00624         return OSP_ADDRESS_ERROR;
00625     }
00626
00627     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00628                 LENGTH_GO_SLEEP_MSG);
00629
00630     cmdBuffer[LENGTH_GO_SLEEP_MSG - 1] = crc (cmdBuffer,
00631         LENGTH_GO_SLEEP_MSG - 1);
00632
00633     p_cmdInfo->outCmdBufferLength = LENGTH_GO_SLEEP_MSG;
00634     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00635     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00636     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00637
00638     break;
00639 }
00640
00641 case (OSP_OSIRE_GO_DEEP_SLEEP_SR):
00642 {
00643
00644     if (p_cmdInfo->p_inParameter != NULL)
00645     {
00646         return OSP_ERROR_PARAMETER;
00647     }
00648     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00649         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00650     {
00651         return OSP_ADDRESS_ERROR;
00652     }
00653
00654     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00655                 LENGTH_GO_DEEP_SLEEP_MSG);
00656
00657     cmdBuffer[LENGTH_GO_DEEP_SLEEP_MSG - 1] = crc (cmdBuffer,
00658         LENGTH_GO_DEEP_SLEEP_MSG - 1);
00659
00660     p_cmdInfo->outCmdBufferLength = LENGTH_GO_DEEP_SLEEP_MSG;
00661     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00662     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00663     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00664

```

```

00665         break;
00666     }
00667
00668     case (OSP_OSIRE_CLR_ERROR_SR):
00669     {
00670
00671         if (p_cmdInfo->p_inParameter != NULL)
00672         {
00673             return OSP_ERROR_PARAMETER;
00674         }
00675         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00676             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00677         {
00678             return OSP_ADDRESS_ERROR;
00679         }
00680
00681         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00682                     LENGTH_CLR_ERROR_MSG);
00683
00684         cmdBuffer[LENGTH_CLR_ERROR_MSG - 1] = crc (cmdBuffer,
00685             LENGTH_CLR_ERROR_MSG - 1);
00686
00687         p_cmdInfo->outCmdBufferLength = LENGTH_CLR_ERROR_MSG;
00688         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00689         p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00690         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00691
00692         break;
00693     }
00694
00695     case (OSP_OSIRE_P4ERROR_BIDIR):
00696     {
00697
00698         if (p_cmdInfo->p_inParameter != NULL)
00699         {
00700             return OSP_ERROR_PARAMETER;
00701         }
00702         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00703             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00704         {
00705             return OSP_ADDRESS_ERROR;
00706         }
00707
00708         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00709                     LENGTH_P4ERROR_MSG);
00710
00711         cmdBuffer[LENGTH_P4ERROR_MSG - 1] = crc (cmdBuffer,
00712             LENGTH_P4ERROR_MSG - 1);
00713
00714         p_cmdInfo->outCmdBufferLength = LENGTH_P4ERROR_MSG;
00715         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00716         p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00717         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00718
00719         break;
00720     }
00721
00722     case (OSP_OSIRE_P4ERROR_LOOP):
00723     {
00724
00725         if (p_cmdInfo->p_inParameter != NULL)
00726         {
00727             return OSP_ERROR_PARAMETER;
00728         }
00729         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00730             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00731         {
00732             return OSP_ADDRESS_ERROR;
00733         }
00734
00735         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00736                     LENGTH_P4ERROR_MSG);
00737
00738         cmdBuffer[LENGTH_P4ERROR_MSG - 1] = crc (cmdBuffer,
00739             LENGTH_P4ERROR_MSG - 1);
00740
00741         p_cmdInfo->outCmdBufferLength = LENGTH_P4ERROR_MSG;
00742         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00743         p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00744         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00745
00746         break;
00747     }
00748
00749     case (OSP_OSIRE_READ_SETUP):
00750     {
00751

```

```

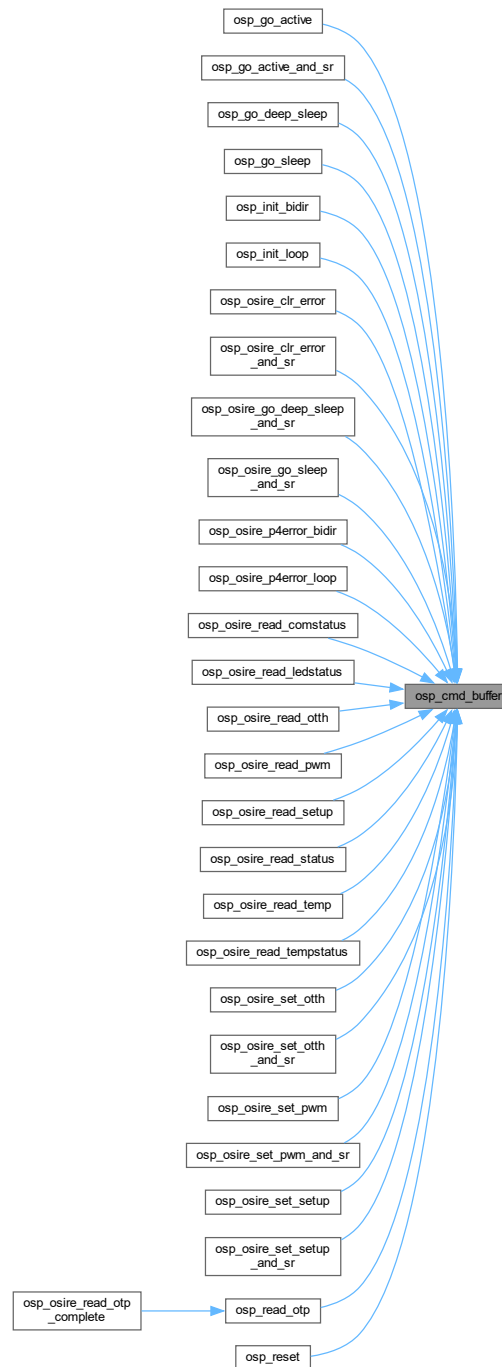
00752         if (p_cmdInfo->p_inParameter != NULL)
00753         {
00754             return OSP_ERROR_PARAMETER;
00755         }
00756         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00757             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00758         {
00759             return OSP_ADDRESS_ERROR;
00760         }
00761
00762         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00763                     LENGTH_READ_SETUP_MSG);
00764
00765         cmdBuffer[LENGTH_READ_SETUP_MSG - 1] = crc (cmdBuffer,
00766             LENGTH_READ_SETUP_MSG - 1);
00767
00768         p_cmdInfo->outCmdBufferLength = LENGTH_READ_SETUP_MSG;
00769         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00770         p_cmdInfo->outResponseLength = LENGTH_READ_SETUP_RSP; // response expected
00771         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00772
00773         break;
00774     }
00775
00776     case (OSP_OSIRE_READ_COM_STATUS):
00777     {
00778
00779         if (p_cmdInfo->p_inParameter != NULL)
00780         {
00781             return OSP_ERROR_PARAMETER;
00782         }
00783         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00784             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00785         {
00786             return OSP_ADDRESS_ERROR;
00787         }
00788
00789         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00790                     LENGTH_READ_COMSTATUS_MSG);
00791
00792         cmdBuffer[LENGTH_READ_COMSTATUS_MSG - 1] = crc (cmdBuffer,
00793             LENGTH_READ_COMSTATUS_MSG - 1);
00794
00795         p_cmdInfo->outCmdBufferLength = LENGTH_READ_COMSTATUS_MSG;
00796         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00797         p_cmdInfo->outResponseLength = LENGTH_READ_COMSTATUS_RSP; // response expected
00798         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00799
00800         break;
00801     }
00802
00803     case (OSP_OSIRE_READ_STATUS):
00804     {
00805
00806         if (p_cmdInfo->p_inParameter != NULL)
00807         {
00808             return OSP_ERROR_PARAMETER;
00809         }
00810         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00811             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00812         {
00813             return OSP_ADDRESS_ERROR;
00814         }
00815
00816         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00817                     LENGTH_READ_STATUS_MSG);
00818
00819         cmdBuffer[LENGTH_READ_STATUS_MSG - 1] = crc (cmdBuffer,
00820             LENGTH_READ_STATUS_MSG - 1);
00821
00822         p_cmdInfo->outCmdBufferLength = LENGTH_READ_STATUS_MSG;
00823         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00824         p_cmdInfo->outResponseLength = LENGTH_READ_STATUS_RSP; // response expected
00825         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00826
00827         break;
00828     }
00829
00830     default:
00831     {
00832         return OSP_ERROR_NOT_IMPLEMENTED;
00833     }
00834 }
00835
00836 return OSP_NO_ERROR;
00837 }
00838

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.13 ospCmdBuffer.c

[Go to the documentation of this file.](#)

```

00001 /*****
00002 * Copyright 2022 by ams OSRAM AG
00003 * All rights are reserved.
00004 *
00005 * IMPORTANT - PLEASE READ CAREFULLY BEFORE COPYING, INSTALLING OR USING
00006 * THE SOFTWARE.

```



```

00007 *
00008 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS *
00009 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT *
00010 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS *
00011 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT *
00012 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
00013 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT *
00014 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, *
00015 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY *
00016 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT *
00017 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE *
00018 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. *
00019 *****/
00020
00021 #include <Osp/inc/genericDevice.h>
00022 #include <Osp/inc/osireDevice.h>
00023 #include <Osp/inc/crc.h>
00024 #include <Osp/inc/ospCmdBuffer.h>
00025 #include <string.h>
00026
00027 #define MAX_CMD_BUFFER_SIZE 16
00028 static uint8_t cmdBuffer[MAX_CMD_BUFFER_SIZE];
00029
00030 /*****
00031 /*****
00032 START_FUNCTION_DEFINITION_RAMSECTION
00033 enum OSP_ERROR_CODE osp_cmd_buffer (ospCmdBuffer_t *p_cmdInfo)
00034 {
00035     memset (cmdBuffer, 0, MAX_CMD_BUFFER_SIZE);
00036
00037     switch (p_cmdInfo->inCmdId)
00038     {
00039 /*****
00040 // for genericDevice.c
00041 /*****
00042     case (OSP_INIT_BIDIR):
00043     {
00044         if (p_cmdInfo->p_inParameter != NULL)
00045         {
00046             return OSP_ERROR_PARAMETER;
00047         }
00048         if (p_cmdInfo->inDeviceAddress != 1)
00049         {
00050             return OSP_ADDRESS_ERROR;
00051         }
00052
00053         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00054                     LENGTH_INIT_MSG);
00055
00056         cmdBuffer[LENGTH_INIT_MSG - 1] = crc (cmdBuffer,
00057         LENGTH_INIT_MSG - 1);
00058
00059         p_cmdInfo->outCmdBufferLength = LENGTH_INIT_MSG;
00060         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00061         p_cmdInfo->outResponseLength = LENGTH_INIT_RSP; // response expected
00062         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00063         break;
00064     }
00065
00066     case (OSP_INIT_LOOP):
00067     {
00068         if (p_cmdInfo->p_inParameter != NULL)
00069         {
00070             return OSP_ERROR_PARAMETER;
00071         }
00072         if (p_cmdInfo->inDeviceAddress != 1)
00073         {
00074             return OSP_ADDRESS_ERROR;
00075         }
00076
00077         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00078                     LENGTH_INIT_MSG);
00079
00080         cmdBuffer[LENGTH_INIT_MSG - 1] = crc (cmdBuffer,
00081         LENGTH_INIT_MSG - 1);
00082
00083         p_cmdInfo->outCmdBufferLength = LENGTH_INIT_MSG;
00084         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00085         p_cmdInfo->outResponseLength = LENGTH_INIT_RSP; // response expected
00086         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00087
00088         break;
00089     }
00090
00091     case (OSP_RESET):
00092     {

```

```

00094
00095     if (p_cmdInfo->p_inParameter != NULL)
00096     {
00097         return OSP_ERROR_PARAMETER;
00098     }
00099     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00100     {
00101         return OSP_ADDRESS_ERROR;
00102     }
00103
00104     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00105                 LENGTH_RESET_MSG);
00106
00107     cmdBuffer[LENGTH_RESET_MSG - 1] = crc (cmdBuffer,
00108                 LENGTH_RESET_MSG - 1);
00109
00110     p_cmdInfo->outCmdBufferLength = LENGTH_RESET_MSG;
00111     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00112     p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00113     p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00114
00115     break;
00116 }
00117
00118 case (OSP_GO_ACTIVE):
00119 {
00120
00121     if (p_cmdInfo->p_inParameter != NULL)
00122     {
00123         return OSP_ERROR_PARAMETER;
00124     }
00125     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00126     {
00127         return OSP_ADDRESS_ERROR;
00128     }
00129
00130     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00131                 LENGTH_GO_ACTIVE_MSG);
00132
00133     cmdBuffer[LENGTH_GO_ACTIVE_MSG - 1] = crc (cmdBuffer,
00134                 LENGTH_GO_ACTIVE_MSG - 1);
00135
00136     p_cmdInfo->outCmdBufferLength = LENGTH_GO_ACTIVE_MSG;
00137     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00138     p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00139     p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00140
00141     break;
00142 }
00143
00144 case (OSP_GO_SLEEP):
00145 {
00146
00147     if (p_cmdInfo->p_inParameter != NULL)
00148     {
00149         return OSP_ERROR_PARAMETER;
00150     }
00151     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00152     {
00153         return OSP_ADDRESS_ERROR;
00154     }
00155
00156     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00157                 LENGTH_GO_SLEEP_MSG);
00158
00159     cmdBuffer[LENGTH_GO_SLEEP_MSG - 1] = crc (cmdBuffer,
00160                 LENGTH_GO_SLEEP_MSG - 1);
00161
00162     p_cmdInfo->outCmdBufferLength = LENGTH_GO_SLEEP_MSG;
00163     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00164     p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00165     p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00166
00167     break;
00168 }
00169
00170 case (OSP_GO_DEEP_SLEEP):
00171 {
00172
00173     if (p_cmdInfo->p_inParameter != NULL)
00174     {
00175         return OSP_ERROR_PARAMETER;
00176     }
00177     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00178     {
00179         return OSP_ADDRESS_ERROR;
00180     }

```

```

00181
00182     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00183     LENGTH_GO_DEEP_SLEEP_MSG);
00184
00185     cmdBuffer[LENGTH_GO_DEEP_SLEEP_MSG - 1] = crc (cmdBuffer,
00186     LENGTH_GO_DEEP_SLEEP_MSG - 1);
00187
00188     p_cmdInfo->outCmdBufferLength = LENGTH_GO_DEEP_SLEEP_MSG;
00189     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00190     p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00191     p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00192
00193     break;
00194 }
00195
00196 case (OSP_CLR_ERROR):
00197 {
00198     if (p_cmdInfo->p_inParameter != NULL)
00199     {
00200         return OSP_ERROR_PARAMETER;
00201     }
00202     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00203     {
00204         return OSP_ADDRESS_ERROR;
00205     }
00206
00207     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00208     LENGTH_CLR_ERROR_MSG);
00209
00210     cmdBuffer[LENGTH_CLR_ERROR_MSG - 1] = crc (cmdBuffer,
00211     LENGTH_CLR_ERROR_MSG - 1);
00212
00213     p_cmdInfo->outCmdBufferLength = LENGTH_CLR_ERROR_MSG;
00214     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00215     p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00216     p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00217
00218     break;
00219 }
00220
00221 END_FUNCTION_DEFINITION_RAMSECTION
00222 /*****
00223 // for osireDevice.c
00224 *****/
00225
00226 case (OSP_OSIRE_SET_SETUP):
00227 {
00228     if (p_cmdInfo->p_inParameter == NULL)
00229     {
00230         return OSP_ERROR_PARAMETER;
00231     }
00232     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00233     {
00234         return OSP_ADDRESS_ERROR;
00235     }
00236
00237     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00238     LENGTH_SET_SETUP_MSG);
00239
00240     osireSetSetupData_t *p_data;
00241     p_data = (osireSetSetupData_t*) p_cmdInfo->p_inParameter;
00242     cmdBuffer[3] = p_data->data.setupData;
00243
00244     cmdBuffer[LENGTH_SET_SETUP_MSG - 1] = crc (cmdBuffer,
00245     LENGTH_SET_SETUP_MSG - 1);
00246
00247     p_cmdInfo->outCmdBufferLength = LENGTH_SET_SETUP_MSG;
00248     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00249     p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00250     p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00251
00252     break;
00253 }
00254
00255 case (OSP_OSIRE_SET_SETUP_SR):
00256 {
00257     if (p_cmdInfo->p_inParameter == NULL)
00258     {
00259         return OSP_ERROR_PARAMETER;
00260     }
00261     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00262         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00263     {
00264         return OSP_ADDRESS_ERROR;
00265     }
00266
00267     break;
00268 }

```

```

00268
00269     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00270     LENGTH_SET_SETUP_MSG);
00271
00272     osireSetSetupData_t *p_data;
00273     p_data = (osireSetSetupData_t*) p_cmdInfo->p_inParameter;
00274     cmdBuffer[3] = p_data->data.setupData;
00275
00276     cmdBuffer[LENGTH_SET_SETUP_MSG - 1] = crc (cmdBuffer,
00277     LENGTH_SET_SETUP_MSG - 1);
00278
00279     p_cmdInfo->outCmdBufferLength = LENGTH_SET_SETUP_MSG;
00280     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00281     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00282     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00283
00284     break;
00285 }
00286
00287 case (OSP_OSIRE_SET_PWM) :
00288 {
00289     if (p_cmdInfo->p_inParameter == NULL)
00290     {
00291         return OSP_ERROR_PARAMETER;
00292     }
00293     if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00294     {
00295         return OSP_ADDRESS_ERROR;
00296     }
00297
00298     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00299     LENGTH_SET_PWM_MSG);
00300
00301     osirePwmData_t *p_data;
00302     p_data = (osirePwmData_t*) p_cmdInfo->p_inParameter;
00303     cmdBuffer[8] = p_data->data.pwmData[0];
00304     cmdBuffer[7] = p_data->data.pwmData[1];
00305     cmdBuffer[6] = p_data->data.pwmData[2];
00306     cmdBuffer[5] = p_data->data.pwmData[3];
00307     cmdBuffer[4] = p_data->data.pwmData[4];
00308     cmdBuffer[3] = p_data->data.pwmData[5];
00309
00310     cmdBuffer[LENGTH_SET_PWM_MSG - 1] = crc (cmdBuffer,
00311     LENGTH_SET_PWM_MSG - 1);
00312
00313     p_cmdInfo->outCmdBufferLength = LENGTH_SET_PWM_MSG;
00314     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00315     p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00316     p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00317
00318     break;
00319 }
00320
00321 case (OSP_OSIRE_SET_PWM_SR) :
00322 {
00323     if (p_cmdInfo->p_inParameter == NULL)
00324     {
00325         return OSP_ERROR_PARAMETER;
00326     }
00327     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00328     || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00329     {
00330         return OSP_ADDRESS_ERROR;
00331     }
00332
00333     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00334     LENGTH_SET_PWM_MSG);
00335
00336     osirePwmData_t *p_data;
00337     p_data = (osirePwmData_t*) p_cmdInfo->p_inParameter;
00338     cmdBuffer[8] = p_data->data.pwmData[0];
00339     cmdBuffer[7] = p_data->data.pwmData[1];
00340     cmdBuffer[6] = p_data->data.pwmData[2];
00341     cmdBuffer[5] = p_data->data.pwmData[3];
00342     cmdBuffer[4] = p_data->data.pwmData[4];
00343     cmdBuffer[3] = p_data->data.pwmData[5];
00344
00345     cmdBuffer[LENGTH_SET_PWM_MSG - 1] = crc (cmdBuffer,
00346     LENGTH_SET_PWM_MSG - 1);
00347
00348     p_cmdInfo->outCmdBufferLength = LENGTH_SET_PWM_MSG;
00349     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00350     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00351     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00352
00353
00354

```

```

00355         break;
00356     }
00357
00358     case (OSP_OSIRE_READ_PWM):
00359     {
00360
00361         if (p_cmdInfo->p_inParameter != NULL)
00362         {
00363             return OSP_ERROR_PARAMETER;
00364         }
00365         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00366             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00367         {
00368             return OSP_ADDRESS_ERROR;
00369         }
00370
00371         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00372                     LENGTH_READ_PWM_MSG);
00373
00374         cmdBuffer[LENGTH_READ_PWM_MSG - 1] = crc (cmdBuffer,
00375             LENGTH_READ_PWM_MSG - 1);
00376
00377         p_cmdInfo->outCmdBufferLength = LENGTH_READ_PWM_MSG;
00378         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00379         p_cmdInfo->outResponseLength = LENGTH_READ_PWM_RSP; // response expected
00380         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00381
00382         break;
00383     }
00384
00385     case (OSP_OSIRE_READ_OTP):
00386     {
00387
00388         if (p_cmdInfo->p_inParameter == NULL)
00389         {
00390             return OSP_ERROR_PARAMETER;
00391         }
00392         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00393             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00394         {
00395             return OSP_ADDRESS_ERROR;
00396         }
00397
00398         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00399                     LENGTH_READ_OTP_MSG);
00400
00401         //Offset in OTP memory
00402         cmdBuffer[3] = (*(uint8_t*) p_cmdInfo->p_inParameter));
00403
00404         cmdBuffer[LENGTH_READ_OTP_MSG - 1] = crc (cmdBuffer,
00405             LENGTH_READ_OTP_MSG - 1);
00406
00407         p_cmdInfo->outCmdBufferLength = LENGTH_READ_OTP_MSG;
00408         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00409         p_cmdInfo->outResponseLength = LENGTH_READ_OTP_RSP; // response expected
00410         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00411
00412         break;
00413     }
00414
00415     case (OSP_OSIRE_READ_LED_STATUS):
00416     {
00417
00418         if (p_cmdInfo->p_inParameter != NULL)
00419         {
00420             return OSP_ERROR_PARAMETER;
00421         }
00422         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00423             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00424         {
00425             return OSP_ADDRESS_ERROR;
00426         }
00427
00428         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00429                     LENGTH_READ_LEDSTATUS_MSG);
00430
00431         cmdBuffer[LENGTH_READ_LEDSTATUS_MSG - 1] = crc (cmdBuffer,
00432             LENGTH_READ_LEDSTATUS_MSG - 1);
00433
00434         p_cmdInfo->outCmdBufferLength = LENGTH_READ_LEDSTATUS_MSG;
00435         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00436         p_cmdInfo->outResponseLength = LENGTH_READ_LEDSTATUS_RSP; // response expected
00437         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00438
00439         break;
00440     }
00441

```

```

00442     case (OSP_OSIRE_READ_TEMP_STATUS):
00443     {
00444
00445         if (p_cmdInfo->p_inParameter != NULL)
00446         {
00447             return OSP_ERROR_PARAMETER;
00448         }
00449         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00450             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00451         {
00452             return OSP_ADDRESS_ERROR;
00453         }
00454
00455         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00456                     LENGTH_READ_TEMPSTATUS_MSG);
00457
00458         cmdBuffer[LENGTH_READ_TEMPSTATUS_MSG - 1] = crc (cmdBuffer,
00459                     LENGTH_READ_TEMPSTATUS_MSG - 1);
00460
00461         p_cmdInfo->outCmdBufferLength = LENGTH_READ_TEMPSTATUS_MSG;
00462         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00463         p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00464         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00465
00466         break;
00467     }
00468
00469     case (OSP_OSIRE_READ_TEMP):
00470     {
00471
00472         if (p_cmdInfo->p_inParameter != NULL)
00473         {
00474             return OSP_ERROR_PARAMETER;
00475         }
00476         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00477             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00478         {
00479             return OSP_ADDRESS_ERROR;
00480         }
00481
00482         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00483                     LENGTH_READ_TEMP_MSG);
00484
00485         cmdBuffer[LENGTH_READ_TEMP_MSG - 1] = crc (cmdBuffer,
00486                     LENGTH_READ_TEMP_MSG - 1);
00487
00488         p_cmdInfo->outCmdBufferLength = LENGTH_READ_TEMP_MSG;
00489         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00490         p_cmdInfo->outResponseLength = LENGTH_READ_TEMP_RSP; // response expected
00491         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00492
00493         break;
00494     }
00495
00496     case (OSP_OSIRE_SET_OTTH):
00497     {
00498
00499         if (p_cmdInfo->p_inParameter == NULL)
00500         {
00501             return OSP_ERROR_PARAMETER;
00502         }
00503         if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00504         {
00505             return OSP_ADDRESS_ERROR;
00506         }
00507
00508         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00509                     LENGTH_SET_OTTH_MSG);
00510
00511         osireOtthData_t *p_data;
00512         p_data = (osireOtthData_t*) p_cmdInfo->p_inParameter;
00513         cmdBuffer[5] = p_data->data.otthData[0];
00514         cmdBuffer[4] = p_data->data.otthData[1];
00515         cmdBuffer[3] = p_data->data.otthData[2];
00516
00517         cmdBuffer[LENGTH_SET_OTTH_MSG - 1] = crc (cmdBuffer,
00518                     LENGTH_SET_OTTH_MSG - 1);
00519
00520         p_cmdInfo->outCmdBufferLength = LENGTH_SET_OTTH_MSG;
00521         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00522         p_cmdInfo->outResponseLength = LENGTH_NO_OSP_RSP; // no response expected
00523         p_cmdInfo->outResponseMsg = NO_OSP_RSP; // no response expected
00524
00525         break;
00526     }
00527
00528     case (OSP_OSIRE_SET_OTTH_SR):

```

```

00529     {
00530
00531         if (p_cmdInfo->p_inParameter == NULL)
00532         {
00533             return OSP_ERROR_PARAMETER;
00534         }
00535         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00536             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00537         {
00538             return OSP_ADDRESS_ERROR;
00539         }
00540
00541         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00542                     LENGTH_SET_OTTH_MSG);
00543
00544         osireOtthData_t *p_data;
00545         p_data = (osireOtthData_t*) p_cmdInfo->p_inParameter;
00546         cmdBuffer[5] = p_data->data.otthData[0];
00547         cmdBuffer[4] = p_data->data.otthData[1];
00548         cmdBuffer[3] = p_data->data.otthData[2];
00549
00550         cmdBuffer[LENGTH_SET_OTTH_MSG - 1] = crc (cmdBuffer,
00551                     LENGTH_SET_OTTH_MSG - 1);
00552
00553         p_cmdInfo->outCmdBufferLength = LENGTH_SET_OTTH_MSG;
00554         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00555         p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00556         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00557
00558         break;
00559     }
00560
00561     case (OSP_OSIRE_READ_OTTH):
00562     {
00563
00564         if (p_cmdInfo->p_inParameter != NULL)
00565         {
00566             return OSP_ERROR_PARAMETER;
00567         }
00568         if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00569             || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00570         {
00571             return OSP_ADDRESS_ERROR;
00572         }
00573
00574         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00575                     LENGTH_READ_OTTH_MSG);
00576
00577         cmdBuffer[LENGTH_READ_OTTH_MSG - 1] = crc (cmdBuffer,
00578                     LENGTH_READ_OTTH_MSG - 1);
00579
00580         p_cmdInfo->outCmdBufferLength = LENGTH_READ_OTTH_MSG;
00581         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00582         p_cmdInfo->outResponseLength = LENGTH_READ_OTTH_RSP; // response expected
00583         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00584
00585         break;
00586     }
00587
00588     case (OSP_OSIRE_GO_ACTIVE_SR):
00589     {
00590
00591         if (p_cmdInfo->p_inParameter != NULL)
00592         {
00593             return OSP_ERROR_PARAMETER;
00594         }
00595         if (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS)
00596         {
00597             return OSP_ADDRESS_ERROR;
00598         }
00599
00600         build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00601                     LENGTH_GO_ACTIVE_MSG);
00602
00603         cmdBuffer[LENGTH_GO_ACTIVE_MSG - 1] = crc (cmdBuffer,
00604                     LENGTH_GO_ACTIVE_MSG - 1);
00605
00606         p_cmdInfo->outCmdBufferLength = LENGTH_GO_ACTIVE_MSG;
00607         p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00608         p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00609         p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00610
00611         break;
00612     }
00613
00614     case (OSP_OSIRE_GO_SLEEP_SR):
00615     {

```

```

00616
00617     if (p_cmdInfo->p_inParameter != NULL)
00618     {
00619         return OSP_ERROR_PARAMETER;
00620     }
00621     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00622         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00623     {
00624         return OSP_ADDRESS_ERROR;
00625     }
00626
00627     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00628                 LENGTH_GO_SLEEP_MSG);
00629
00630     cmdBuffer[LENGTH_GO_SLEEP_MSG - 1] = crc (cmdBuffer,
00631         LENGTH_GO_SLEEP_MSG - 1);
00632
00633     p_cmdInfo->outCmdBufferLength = LENGTH_GO_SLEEP_MSG;
00634     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00635     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00636     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00637
00638     break;
00639 }
00640
00641 case (OSP_OSIRE_GO_DEEP_SLEEP_SR):
00642 {
00643
00644     if (p_cmdInfo->p_inParameter != NULL)
00645     {
00646         return OSP_ERROR_PARAMETER;
00647     }
00648     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00649         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00650     {
00651         return OSP_ADDRESS_ERROR;
00652     }
00653
00654     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00655                 LENGTH_GO_DEEP_SLEEP_MSG);
00656
00657     cmdBuffer[LENGTH_GO_DEEP_SLEEP_MSG - 1] = crc (cmdBuffer,
00658         LENGTH_GO_DEEP_SLEEP_MSG - 1);
00659
00660     p_cmdInfo->outCmdBufferLength = LENGTH_GO_DEEP_SLEEP_MSG;
00661     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00662     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00663     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00664
00665     break;
00666 }
00667
00668 case (OSP_OSIRE_CLR_ERROR_SR):
00669 {
00670
00671     if (p_cmdInfo->p_inParameter != NULL)
00672     {
00673         return OSP_ERROR_PARAMETER;
00674     }
00675     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00676         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00677     {
00678         return OSP_ADDRESS_ERROR;
00679     }
00680
00681     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00682                 LENGTH_CLR_ERROR_MSG);
00683
00684     cmdBuffer[LENGTH_CLR_ERROR_MSG - 1] = crc (cmdBuffer,
00685         LENGTH_CLR_ERROR_MSG - 1);
00686
00687     p_cmdInfo->outCmdBufferLength = LENGTH_CLR_ERROR_MSG;
00688     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00689     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00690     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00691
00692     break;
00693 }
00694
00695 case (OSP_OSIRE_P4ERROR_BIDIR):
00696 {
00697
00698     if (p_cmdInfo->p_inParameter != NULL)
00699     {
00700         return OSP_ERROR_PARAMETER;
00701     }
00702     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)

```



```

00703         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00704     {
00705         return OSP_ADDRESS_ERROR;
00706     }
00707
00708     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00709                 LENGTH_P4ERROR_MSG);
00710
00711     cmdBuffer[LENGTH_P4ERROR_MSG - 1] = crc (cmdBuffer,
00712         LENGTH_P4ERROR_MSG - 1);
00713
00714     p_cmdInfo->outCmdBufferLength = LENGTH_P4ERROR_MSG;
00715     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00716     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00717     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00718
00719     break;
00720 }
00721
00722 case (OSP_OSIRE_P4ERROR_LOOP):
00723 {
00724
00725     if (p_cmdInfo->p_inParameter != NULL)
00726     {
00727         return OSP_ERROR_PARAMETER;
00728     }
00729     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00730         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00731     {
00732         return OSP_ADDRESS_ERROR;
00733     }
00734
00735     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00736                 LENGTH_P4ERROR_MSG);
00737
00738     cmdBuffer[LENGTH_P4ERROR_MSG - 1] = crc (cmdBuffer,
00739         LENGTH_P4ERROR_MSG - 1);
00740
00741     p_cmdInfo->outCmdBufferLength = LENGTH_P4ERROR_MSG;
00742     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00743     p_cmdInfo->outResponseLength = LENGTH_READ_TEMPSTATUS_RSP; // response expected
00744     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00745
00746     break;
00747 }
00748
00749 case (OSP_OSIRE_READ_SETUP):
00750 {
00751
00752     if (p_cmdInfo->p_inParameter != NULL)
00753     {
00754         return OSP_ERROR_PARAMETER;
00755     }
00756     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00757         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00758     {
00759         return OSP_ADDRESS_ERROR;
00760     }
00761
00762     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00763                 LENGTH_READ_SETUP_MSG);
00764
00765     cmdBuffer[LENGTH_READ_SETUP_MSG - 1] = crc (cmdBuffer,
00766         LENGTH_READ_SETUP_MSG - 1);
00767
00768     p_cmdInfo->outCmdBufferLength = LENGTH_READ_SETUP_MSG;
00769     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00770     p_cmdInfo->outResponseLength = LENGTH_READ_SETUP_RSP; // response expected
00771     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00772
00773     break;
00774 }
00775
00776 case (OSP_OSIRE_READ_COM_STATUS):
00777 {
00778
00779     if (p_cmdInfo->p_inParameter != NULL)
00780     {
00781         return OSP_ERROR_PARAMETER;
00782     }
00783     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00784         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00785     {
00786         return OSP_ADDRESS_ERROR;
00787     }
00788
00789     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,

```

```
00790     LENGTH_READ_COMSTATUS_MSG);
00791
00792     cmdBuffer[LENGTH_READ_COMSTATUS_MSG - 1] = crc (cmdBuffer,
00793     LENGTH_READ_COMSTATUS_MSG - 1);
00794
00795     p_cmdInfo->outCmdBufferLength = LENGTH_READ_COMSTATUS_MSG;
00796     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00797     p_cmdInfo->outResponseLength = LENGTH_READ_COMSTATUS_RSP; // response expected
00798     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00799
00800     break;
00801 }
00802
00803 case (OSP_OSIRE_READ_STATUS):
00804 {
00805
00806     if (p_cmdInfo->p_inParameter != NULL)
00807     {
00808         return OSP_ERROR_PARAMETER;
00809     }
00810     if ((p_cmdInfo->inDeviceAddress == BROADCAST_ADDRESS)
00811         || (p_cmdInfo->inDeviceAddress > MAXIMUM_ADDRESS))
00812     {
00813         return OSP_ADDRESS_ERROR;
00814     }
00815
00816     build_header (cmdBuffer, p_cmdInfo->inDeviceAddress, p_cmdInfo->inCmdId,
00817     LENGTH_READ_STATUS_MSG);
00818
00819     cmdBuffer[LENGTH_READ_STATUS_MSG - 1] = crc (cmdBuffer,
00820     LENGTH_READ_STATUS_MSG - 1);
00821
00822     p_cmdInfo->outCmdBufferLength = LENGTH_READ_STATUS_MSG;
00823     p_cmdInfo->p_outCmdBuffer = (uint8_t*) cmdBuffer;
00824     p_cmdInfo->outResponseLength = LENGTH_READ_STATUS_RSP; // response expected
00825     p_cmdInfo->outResponseMsg = OSP_RSP; // response expected
00826
00827     break;
00828 }
00829
00830 default:
00831 {
00832     return OSP_ERROR_NOT_IMPLEMENTED;
00833 }
00834
00835 }
00836
00837 return OSP_NO_ERROR;
00838 }
```

Disclaimer:

ams-OSRAM AG assumes neither warranty, nor guarantee nor any other liability of any kind for the contents and correctness of the provided data. The data has been generated with highest diligence but may in reality not represent the complete possible variation range of all component parameters. Therefore, in certain cases a deviation between the real optical, thermal, electrical behaviour and the characteristics which are encoded in the provided data could occur.

ams-OSRAM AG reserves the right to undertake technical changes of the component without further notification which could lead to changes in the provided data. ams-OSRAM AG assumes no liability of any kind for the loss of data or any other damage resulting from the usage of the provided data. The user agrees to this disclaimer and user agreement with the download or usage of the provided files.

Index

address
 ComStatus_t, 7
 InitRsp_t, 8
 LedStatus_t, 10
 osireTemp_t, 11
 ospHeader_t, 13
 OtpData_t, 15
 OtpDataComplete_t, 15
 OtthData_t, 16
 PwmData_t, 17
 SetSetupData_t, 19
 Status_t, 21
 TempStatus_t, 23

bit
 ComStatus_t, 7
 InitRsp_t, 8
 LedStatus_t, 10
 ospHeader_t, 13
 OtthData_t, 16
 PwmData_t, 18
 SetSetupData_t, 19
 Status_t, 21

blue_curr
 PwmData_t, 18

blue_open
 LedStatus_t, 10

blue_pwm
 PwmData_t, 18

blue_short
 LedStatus_t, 10

BROADCAST_ADDRESS
 genericDevice.h, 27

buf
 ospHeader_t, 14

build_header
 genericDevice.c, 76
 genericDevice.h, 29

byte
 OtpData_t, 15
 OtpDataComplete_t, 15
 TempStatus_t, 23

C:/_uc-fpu/Documentation/mainpage_osp.md, 25

ce_flag
 Status_t, 21

ce_fsave
 SetSetupData_t, 19

com_inv
 SetSetupData_t, 19

com_mode
 Status_t, 21

command
 ospHeader_t, 14

comStatus
 ComStatus_t, 7

ComStatus_t, 7
 address, 7
 bit, 7
 comStatus, 7
 data, 7
 reserved, 8
 sio1_state, 8
 sio2_state, 8

crc_en
 SetSetupData_t, 20

data
 ComStatus_t, 7
 InitRsp_t, 9
 LedStatus_t, 10
 osireTemp_t, 11
 OtpData_t, 15
 OtpDataComplete_t, 15
 OtthData_t, 16
 PwmData_t, 18
 SetSetupData_t, 20
 Status_t, 21
 TempStatus_t, 23

fast_pwm
 SetSetupData_t, 20

FIRST_BYTE_PAYLOAD
 genericDevice.h, 27

genericDevice.c
 build_header, 76
 osp_go_active, 77
 osp_go_deep_sleep, 78
 osp_go_sleep, 79
 osp_init_bidir, 80
 osp_init_loop, 81
 osp_osire_clr_error, 82
 osp_reset, 83

genericDevice.h
 BROADCAST_ADDRESS, 27
 build_header, 29
 FIRST_BYTE_PAYLOAD, 27
 LENGTH_CLR_ERROR_MSG, 27
 LENGTH_GO_ACTIVE_MSG, 27

- LENGTH_GO_DEEP_SLEEP_MSG, 27
- LENGTH_GO_SLEEP_MSG, 27
- LENGTH_INIT_MSG, 27
- LENGTH_INIT_RSP, 27
- LENGTH_NO_OSP_RSP, 27
- LENGTH_RESET_MSG, 27
- MAXIMUM_ADDRESS, 28
- NO_OSP_RSP, 28
- OSP_ADDRESS_ERROR, 28
- OSP_CLR_ERROR, 29
- OSP_ERROR_CODE, 28
- OSP_ERROR_CRC, 28
- OSP_ERROR_INITIALIZATION, 28
- OSP_ERROR_NOT_IMPLEMENTED, 28
- OSP_ERROR_PARAMETER, 28
- OSP_ERROR_SPI, 28
- OSP_GENERIC_DEVICE_CMDS, 29
- OSP_GO_ACTIVE, 29
- osp_go_active, 31
- OSP_GO_DEEP_SLEEP, 29
- osp_go_deep_sleep, 32
- OSP_GO_SLEEP, 29
- osp_go_sleep, 33
- OSP_INIT_BIDIR, 29
- osp_init_bidir, 34
- OSP_INIT_LOOP, 29
- osp_init_loop, 35
- OSP_NO_ERROR, 28
- osp_osire_clr_error, 36
- OSP_PROTOCOL_PREAMPLE, 28
- OSP_RESET, 29
- osp_reset, 37
- OSP_RSP, 28
- osplnitRsp_t, 28
- green_curr
 - PwmData_t, 18
- green_open
 - LedStatus_t, 10
- green_pwm
 - PwmData_t, 18
- green_short
 - LedStatus_t, 10
- inCmdId
 - ospCmd_t, 12
- inDeviceAddress
 - ospCmd_t, 12
- InitRsp_t, 8
 - address, 8
 - bit, 8
 - data, 9
 - rsp, 9
 - status, 9
 - temp, 9
- ledStatus
 - LedStatus_t, 10
- ledStatus_reserved_1
 - LedStatus_t, 10
- ledStatus_reserved_2
 - LedStatus_t, 11
- LedStatus_t, 9
 - address, 10
 - bit, 10
 - blue_open, 10
 - blue_short, 10
 - data, 10
 - green_open, 10
 - green_short, 10
 - ledStatus, 10
 - ledStatus_reserved_1, 10
 - ledStatus_reserved_2, 11
 - red_open, 11
 - red_short, 11
- LENGTH_CLR_ERROR_MSG
 - genericDevice.h, 27
- LENGTH_GO_ACTIVE_MSG
 - genericDevice.h, 27
- LENGTH_GO_DEEP_SLEEP_MSG
 - genericDevice.h, 27
- LENGTH_GO_SLEEP_MSG
 - genericDevice.h, 27
- LENGTH_INIT_MSG
 - genericDevice.h, 27
- LENGTH_INIT_RSP
 - genericDevice.h, 27
- LENGTH_NO_OSP_RSP
 - genericDevice.h, 27
- LENGTH_P4ERROR_MSG
 - osireDevice.h, 43
- LENGTH_READ_COMSTATUS_MSG
 - osireDevice.h, 43
- LENGTH_READ_COMSTATUS_RSP
 - osireDevice.h, 43
- LENGTH_READ_LEDSTATUS_MSG
 - osireDevice.h, 43
- LENGTH_READ_LEDSTATUS_RSP
 - osireDevice.h, 43
- LENGTH_READ_OTP_MSG
 - osireDevice.h, 43
- LENGTH_READ_OTP_RSP
 - osireDevice.h, 43
- LENGTH_READ_OTTH_MSG
 - osireDevice.h, 44
- LENGTH_READ_OTTH_RSP
 - osireDevice.h, 44
- LENGTH_READ_PWM_MSG
 - osireDevice.h, 44
- LENGTH_READ_PWM_RSP
 - osireDevice.h, 44
- LENGTH_READ_SETUP_MSG
 - osireDevice.h, 44
- LENGTH_READ_SETUP_RSP
 - osireDevice.h, 44
- LENGTH_READ_STATUS_MSG
 - osireDevice.h, 44
- LENGTH_READ_STATUS_RSP

- osireDevice.h, 44
- LENGTH_READ_TEMP_MSG
 - osireDevice.h, 44
- LENGTH_READ_TEMP_RSP
 - osireDevice.h, 44
- LENGTH_READ_TEMPSTATUS_MSG
 - osireDevice.h, 45
- LENGTH_READ_TEMPSTATUS_RSP
 - osireDevice.h, 45
- LENGTH_RESET_MSG
 - genericDevice.h, 27
- LENGTH_SET_OTTH_MSG
 - osireDevice.h, 45
- LENGTH_SET_PWM_MSG
 - osireDevice.h, 45
- LENGTH_SET_SETUP_MSG
 - osireDevice.h, 45
- los_flag
 - Status_t, 22
- los_fsave
 - SetSetupData_t, 20
- MAX_CMD_BUFFER_SIZE
 - ospCmdBuffer.c, 123
- MAXIMUM_ADDRESS
 - genericDevice.h, 28
- NO_OSP_RSP
 - genericDevice.h, 28
- or_cycle
 - OtthData_t, 16
- osireComStatus_t
 - osireDevice.h, 45
- osireDevice.c
 - osp_go_active_and_sr, 89
 - osp_osire_clr_error_and_sr, 90
 - osp_osire_go_deep_sleep_and_sr, 91
 - osp_osire_go_sleep_and_sr, 92
 - osp_osire_p4error_bidir, 93
 - osp_osire_p4error_loop, 95
 - osp_osire_read_comstatus, 96
 - osp_osire_read_ledstatus, 97
 - osp_osire_read_otp_complete, 98
 - osp_osire_read_otth, 98
 - osp_osire_read_pwm, 99
 - osp_osire_read_setup, 100
 - osp_osire_read_status, 101
 - osp_osire_read_temp, 102
 - osp_osire_read_tempstatus, 103
 - osp_osire_set_otth, 104
 - osp_osire_set_otth_and_sr, 105
 - osp_osire_set_pwm, 106
 - osp_osire_set_pwm_and_sr, 107
 - osp_osire_set_setup, 108
 - osp_osire_set_setup_and_sr, 109
 - osp_read_otp, 110
- osireDevice.h
 - LENGTH_P4ERROR_MSG, 43
 - LENGTH_READ_COMSTATUS_MSG, 43
 - LENGTH_READ_COMSTATUS_RSP, 43
 - LENGTH_READ_LEDSTATUS_MSG, 43
 - LENGTH_READ_LEDSTATUS_RSP, 43
 - LENGTH_READ_OTP_MSG, 43
 - LENGTH_READ_OTP_RSP, 43
 - LENGTH_READ_OTTH_MSG, 44
 - LENGTH_READ_OTTH_RSP, 44
 - LENGTH_READ_PWM_MSG, 44
 - LENGTH_READ_PWM_RSP, 44
 - LENGTH_READ_SETUP_MSG, 44
 - LENGTH_READ_SETUP_RSP, 44
 - LENGTH_READ_STATUS_MSG, 44
 - LENGTH_READ_STATUS_RSP, 44
 - LENGTH_READ_TEMP_MSG, 44
 - LENGTH_READ_TEMP_RSP, 44
 - LENGTH_READ_TEMPSTATUS_MSG, 45
 - LENGTH_READ_TEMPSTATUS_RSP, 45
 - LENGTH_SET_OTTH_MSG, 45
 - LENGTH_SET_PWM_MSG, 45
 - LENGTH_SET_SETUP_MSG, 45
 - osireComStatus_t, 45
 - osireLedStatus_t, 45
 - osireOtpData_t, 45
 - osireOtpDataComplete_t, 45
 - osireOtthData_t, 46
 - osirePwmData_t, 46
 - osireSetSetupData_t, 46
 - osireStatus_t, 46
 - osireTemp_t, 46
 - osireTempStatus_t, 46
 - osp_go_active_and_sr, 47
 - osp_osire_clr_error_and_sr, 48
 - OSP_OSIRE_CLR_ERROR_SR, 46
 - OSP_OSIRE_DEVICE_CMDS, 46
 - OSP_OSIRE_GO_ACTIVE_SR, 46
 - osp_osire_go_deep_sleep_and_sr, 49
 - OSP_OSIRE_GO_DEEP_SLEEP_SR, 46
 - osp_osire_go_sleep_and_sr, 50
 - OSP_OSIRE_GO_SLEEP_SR, 46
 - OSP_OSIRE_P4ERROR_BIDIR, 46
 - osp_osire_p4error_bidir, 51
 - OSP_OSIRE_P4ERROR_LOOP, 47
 - osp_osire_p4error_loop, 52
 - OSP_OSIRE_READ_COM_STATUS, 46
 - osp_osire_read_comstatus, 54
 - OSP_OSIRE_READ_LED_STATUS, 46
 - osp_osire_read_ledstatus, 54
 - OSP_OSIRE_READ_OTP, 47
 - osp_osire_read_otp_complete, 55
 - OSP_OSIRE_READ_OTTH, 46
 - osp_osire_read_otth, 56
 - OSP_OSIRE_READ_PWM, 47
 - osp_osire_read_pwm, 57
 - OSP_OSIRE_READ_SETUP, 47
 - osp_osire_read_setup, 58
 - OSP_OSIRE_READ_STATUS, 46
 - osp_osire_read_status, 59

- OSP_OSIRE_READ_TEMP, 46
- osp_osire_read_temp, 60
- OSP_OSIRE_READ_TEMP_STATUS, 46
- osp_osire_read_tempstatus, 61
- OSP_OSIRE_SET_OTTH, 46
- osp_osire_set_otth, 62
- osp_osire_set_otth_and_sr, 63
- OSP_OSIRE_SET_OTTH_SR, 47
- OSP_OSIRE_SET_PWM, 47
- osp_osire_set_pwm, 64
- osp_osire_set_pwm_and_sr, 65
- OSP_OSIRE_SET_PWM_SR, 47
- OSP_OSIRE_SET_SETUP, 47
- osp_osire_set_setup, 66
- osp_osire_set_setup_and_sr, 67
- OSP_OSIRE_SET_SETUP_SR, 47
- osp_read_otp, 68
- osireLedStatus_t
 - osireDevice.h, 45
- osireOtpData_t
 - osireDevice.h, 45
- osireOtpDataComplete_t
 - osireDevice.h, 45
- osireOtthData_t
 - osireDevice.h, 46
- osirePwmData_t
 - osireDevice.h, 46
- osireSetSetupData_t
 - osireDevice.h, 46
- osireStatus_t
 - osireDevice.h, 46
- osireTemp_t, 11
 - address, 11
 - data, 11
 - osireDevice.h, 46
 - temp_value, 12
- osireTempStatus_t
 - osireDevice.h, 46
- OSP/inc/genericDevice.h, 25, 38
- OSP/inc/osireDevice.h, 40, 69
- OSP/inc/ospCmdBuffer.h, 73, 74
- OSP/src/genericDevice.c, 75, 84
- OSP/src/osireDevice.c, 87, 111
- OSP/src/ospCmdBuffer.c, 122, 134
- OSP_ADDRESS_ERROR
 - genericDevice.h, 28
- OSP_CLR_ERROR
 - genericDevice.h, 29
- osp_cmd_buffer
 - ospCmdBuffer.c, 123
- OSP_ERROR_CODE
 - genericDevice.h, 28
- OSP_ERROR_CRC
 - genericDevice.h, 28
- OSP_ERROR_INITIALIZATION
 - genericDevice.h, 28
- OSP_ERROR_NOT_IMPLEMENTED
 - genericDevice.h, 28
- OSP_ERROR_PARAMETER
 - genericDevice.h, 28
- OSP_ERROR_SPI
 - genericDevice.h, 28
- OSP_GENERIC_DEVICE_CMDS
 - genericDevice.h, 29
- OSP_GO_ACTIVE
 - genericDevice.h, 29
- osp_go_active
 - genericDevice.c, 77
 - genericDevice.h, 31
- osp_go_active_and_sr
 - osireDevice.c, 89
 - osireDevice.h, 47
- OSP_GO_DEEP_SLEEP
 - genericDevice.h, 29
- osp_go_deep_sleep
 - genericDevice.c, 78
 - genericDevice.h, 32
- OSP_GO_SLEEP
 - genericDevice.h, 29
- osp_go_sleep
 - genericDevice.c, 79
 - genericDevice.h, 33
- OSP_INIT_BIDIR
 - genericDevice.h, 29
- osp_init_bidir
 - genericDevice.c, 80
 - genericDevice.h, 34
- OSP_INIT_LOOP
 - genericDevice.h, 29
- osp_init_loop
 - genericDevice.c, 81
 - genericDevice.h, 35
- OSP_NO_ERROR
 - genericDevice.h, 28
- osp_osire_clr_error
 - genericDevice.c, 82
 - genericDevice.h, 36
- osp_osire_clr_error_and_sr
 - osireDevice.c, 90
 - osireDevice.h, 48
- OSP_OSIRE_CLR_ERROR_SR
 - osireDevice.h, 46
- OSP_OSIRE_DEVICE_CMDS
 - osireDevice.h, 46
- OSP_OSIRE_GO_ACTIVE_SR
 - osireDevice.h, 46
- osp_osire_go_deep_sleep_and_sr
 - osireDevice.c, 91
 - osireDevice.h, 49
- OSP_OSIRE_GO_DEEP_SLEEP_SR
 - osireDevice.h, 46
- osp_osire_go_sleep_and_sr
 - osireDevice.c, 92
 - osireDevice.h, 50
- OSP_OSIRE_GO_SLEEP_SR
 - osireDevice.h, 46

OSP_OSIRE_P4ERROR_BIDIR
 osireDevice.h, 46
osp_osire_p4error_bidir
 osireDevice.c, 93
 osireDevice.h, 51
OSP_OSIRE_P4ERROR_LOOP
 osireDevice.h, 47
osp_osire_p4error_loop
 osireDevice.c, 95
 osireDevice.h, 52
OSP_OSIRE_READ_COM_STATUS
 osireDevice.h, 46
osp_osire_read_comstatus
 osireDevice.c, 96
 osireDevice.h, 54
OSP_OSIRE_READ_LED_STATUS
 osireDevice.h, 46
osp_osire_read_ledstatus
 osireDevice.c, 97
 osireDevice.h, 54
OSP_OSIRE_READ_OTP
 osireDevice.h, 47
osp_osire_read_otp_complete
 osireDevice.c, 98
 osireDevice.h, 55
OSP_OSIRE_READ_OTTH
 osireDevice.h, 46
osp_osire_read_otth
 osireDevice.c, 98
 osireDevice.h, 56
OSP_OSIRE_READ_PWM
 osireDevice.h, 47
osp_osire_read_pwm
 osireDevice.c, 99
 osireDevice.h, 57
OSP_OSIRE_READ_SETUP
 osireDevice.h, 47
osp_osire_read_setup
 osireDevice.c, 100
 osireDevice.h, 58
OSP_OSIRE_READ_STATUS
 osireDevice.h, 46
osp_osire_read_status
 osireDevice.c, 101
 osireDevice.h, 59
OSP_OSIRE_READ_TEMP
 osireDevice.h, 46
osp_osire_read_temp
 osireDevice.c, 102
 osireDevice.h, 60
OSP_OSIRE_READ_TEMP_STATUS
 osireDevice.h, 46
osp_osire_read_tempstatus
 osireDevice.c, 103
 osireDevice.h, 61
OSP_OSIRE_SET_OTTH
 osireDevice.h, 46
osp_osire_set_otth
 osireDevice.c, 104
 osireDevice.h, 62
osp_osire_set_otth_and_sr
 osireDevice.c, 105
 osireDevice.h, 63
OSP_OSIRE_SET_OTTH_SR
 osireDevice.h, 47
OSP_OSIRE_SET_PWM
 osireDevice.h, 47
osp_osire_set_pwm
 osireDevice.c, 106
 osireDevice.h, 64
osp_osire_set_pwm_and_sr
 osireDevice.c, 107
 osireDevice.h, 65
OSP_OSIRE_SET_PWM_SR
 osireDevice.h, 47
OSP_OSIRE_SET_SETUP
 osireDevice.h, 47
osp_osire_set_setup
 osireDevice.c, 108
 osireDevice.h, 66
osp_osire_set_setup_and_sr
 osireDevice.c, 109
 osireDevice.h, 67
OSP_OSIRE_SET_SETUP_SR
 osireDevice.h, 47
OSP_PROTOCOL_PREAMPLE
 genericDevice.h, 28
osp_read_otp
 osireDevice.c, 110
 osireDevice.h, 68
OSP_RESET
 genericDevice.h, 29
osp_reset
 genericDevice.c, 83
 genericDevice.h, 37
OSP_RSP
 genericDevice.h, 28
ospCmd_t, 12
 inCmdId, 12
 inDeviceAddress, 12
 outCmdBufferLength, 12
 outResponseLength, 12
 outResponseMsg, 13
 p_inParameter, 13
 p_outCmdBuffer, 13
ospCmdBuffer.c
 MAX_CMD_BUFFER_SIZE, 123
 osp_cmd_buffer, 123
ospCmdBuffer.h
 ospCmdBuffer_t, 74
ospCmdBuffer_t
 ospCmdBuffer.h, 74
ospHeader_t, 13
 address, 13
 bit, 13
 buf, 14

- command, [14](#)
 - preample, [14](#)
 - psi, [14](#)
 - reserved, [14](#)
- osplnitRsp_t
 - genericDevice.h, [28](#)
- ot_flag
 - Status_t, [22](#)
- ot_fsave
 - SetSetupData_t, [20](#)
- ot_high_value
 - OtthData_t, [16](#)
- ot_low_value
 - OtthData_t, [17](#)
- otpcrc_flag
 - Status_t, [22](#)
- OtpData_t, [14](#)
 - address, [15](#)
 - byte, [15](#)
 - data, [15](#)
- OtpDataComplete_t, [15](#)
 - address, [15](#)
 - byte, [15](#)
 - data, [15](#)
- otth_reserved
 - OtthData_t, [17](#)
- otthData
 - OtthData_t, [17](#)
- OtthData_t, [16](#)
 - address, [16](#)
 - bit, [16](#)
 - data, [16](#)
 - or_cycle, [16](#)
 - ot_high_value, [16](#)
 - ot_low_value, [17](#)
 - otth_reserved, [17](#)
 - otthData, [17](#)
- outCmdBufferLength
 - ospCmd_t, [12](#)
- outResponseLength
 - ospCmd_t, [12](#)
- outResponseMsg
 - ospCmd_t, [13](#)
- p_inParameter
 - ospCmd_t, [13](#)
- p_outCmdBuffer
 - ospCmd_t, [13](#)
- preample
 - ospHeader_t, [14](#)
- psi
 - ospHeader_t, [14](#)
- pwmData
 - PwmData_t, [18](#)
- PwmData_t, [17](#)
 - address, [17](#)
 - bit, [18](#)
 - blue_curr, [18](#)
 - blue_pwm, [18](#)
 - data, [18](#)
 - green_curr, [18](#)
 - green_pwm, [18](#)
 - pwmData, [18](#)
 - red_curr, [18](#)
 - red_pwm, [18](#)
- red_curr
 - PwmData_t, [18](#)
- red_open
 - LedStatus_t, [11](#)
- red_pwm
 - PwmData_t, [18](#)
- red_short
 - LedStatus_t, [11](#)
- reserved
 - ComStatus_t, [8](#)
 - ospHeader_t, [14](#)
- rsp
 - InitRsp_t, [9](#)
- SetSetupData_t, [19](#)
 - address, [19](#)
 - bit, [19](#)
 - ce_fsave, [19](#)
 - com_inv, [19](#)
 - crc_en, [20](#)
 - data, [20](#)
 - fast_pwm, [20](#)
 - los_fsave, [20](#)
 - ot_fsave, [20](#)
 - setupData, [20](#)
 - tempck_sel, [20](#)
 - uv_fsave, [20](#)
- setupData
 - SetSetupData_t, [20](#)
- sio1_state
 - ComStatus_t, [8](#)
- sio2_state
 - ComStatus_t, [8](#)
- state
 - Status_t, [22](#)
- Status
 - TempStatus_t, [23](#)
- status
 - InitRsp_t, [9](#)
 - Status_t, [22](#)
- Status_t, [21](#)
 - address, [21](#)
 - bit, [21](#)
 - ce_flag, [21](#)
 - com_mode, [21](#)
 - data, [21](#)
 - los_flag, [22](#)
 - ot_flag, [22](#)
 - otpcrc_flag, [22](#)
 - state, [22](#)
 - status, [22](#)
 - uv_flag, [22](#)

- Temp
 - TempStatus_t, [23](#)
- temp
 - InitRsp_t, [9](#)
- temp_value
 - osireTemp_t, [12](#)
- tempck_sel
 - SetSetupData_t, [20](#)
- tempStatus
 - TempStatus_t, [23](#)
- TempStatus_t, [22](#)
 - address, [23](#)
 - byte, [23](#)
 - data, [23](#)
 - Status, [23](#)
 - Temp, [23](#)
 - tempStatus, [23](#)
- uv_flag
 - Status_t, [22](#)
- uv_fsave
 - SetSetupData_t, [20](#)