

# ChibiOS/HAL

4.0.3

## Reference Manual

Sat Feb 6 2016 14:36:30



# Contents

<b>1 ChibiOS/HAL</b>	<b>1</b>
1.1 Copyright . . . . .	1
1.2 Introduction . . . . .	1
1.3 Related Documents . . . . .	1
<b>2 Deprecated List</b>	<b>3</b>
<b>3 Module Index</b>	<b>5</b>
3.1 Modules . . . . .	5
<b>4 Hierarchical Index</b>	<b>7</b>
4.1 Class Hierarchy . . . . .	7
<b>5 Data Structure Index</b>	<b>9</b>
5.1 Data Structures . . . . .	9
<b>6 File Index</b>	<b>13</b>
6.1 File List . . . . .	13
<b>7 Module Documentation</b>	<b>17</b>
7.1 ADC Driver . . . . .	17
7.1.1 Detailed Description . . . . .	17
7.1.2 Driver State Machine . . . . .	17
7.1.3 ADC Operations . . . . .	18
7.1.3.1 ADC Conversion Groups . . . . .	18
7.1.3.2 ADC Conversion Modes . . . . .	18
7.1.3.3 ADC Callbacks . . . . .	18
7.1.4 Macro Definition Documentation . . . . .	21
7.1.4.1 ADC_USE_WAIT . . . . .	21
7.1.4.2 ADC_USE_MUTUAL_EXCLUSION . . . . .	21
7.1.4.3 _adc_reset_i . . . . .	21
7.1.4.4 _adc_reset_s . . . . .	21
7.1.4.5 _adc_wakeup_isr . . . . .	22
7.1.4.6 _adc_timeout_isr . . . . .	22

7.1.4.7	_adc_isr_half_code	22
7.1.4.8	_adc_isr_full_code	23
7.1.4.9	_adc_isr_error_code	23
7.1.4.10	PLATFORM_ADC_USE_ADC1	24
7.1.5	Typedef Documentation	24
7.1.5.1	adcsample_t	24
7.1.5.2	adc_channels_num_t	24
7.1.5.3	ADCDriver	24
7.1.5.4	adccallback_t	24
7.1.5.5	adcerrorcallback_t	24
7.1.6	Enumeration Type Documentation	25
7.1.6.1	adcstate_t	25
7.1.6.2	adcerror_t	25
7.1.7	Function Documentation	25
7.1.7.1	adclInit	25
7.1.7.2	adcObjectInit	26
7.1.7.3	adcStart	26
7.1.7.4	adcStop	27
7.1.7.5	adcStartConversion	28
7.1.7.6	adcStartConversionl	29
7.1.7.7	adcStopConversion	30
7.1.7.8	adcStopConversionl	30
7.1.7.9	adcConvert	31
7.1.7.10	adcAcquireBus	31
7.1.7.11	adcReleaseBus	32
7.1.7.12	adc_lld_init	32
7.1.7.13	adc_lld_start	33
7.1.7.14	adc_lld_stop	33
7.1.7.15	adc_lld_start_conversion	33
7.1.7.16	adc_lld_stop_conversion	33
7.1.8	Variable Documentation	34
7.1.8.1	ADCD1	34
7.2	CAN Driver	35
7.2.1	Detailed Description	35
7.2.2	Driver State Machine	35
7.2.3	Macro Definition Documentation	37
7.2.3.1	CAN_LIMIT_WARNING	37
7.2.3.2	CAN_LIMIT_ERROR	38
7.2.3.3	CAN_BUS_OFF_ERROR	38
7.2.3.4	CAN_FRAMING_ERROR	38

---

7.2.3.5	CAN_OVERFLOW_ERROR . . . . .	38
7.2.3.6	CAN_ANY_MAILBOX . . . . .	38
7.2.3.7	CAN_USE_SLEEP_MODE . . . . .	38
7.2.3.8	CAN_MAILBOX_TO_MASK . . . . .	38
7.2.3.9	CAN_TX_MAILBOXES . . . . .	38
7.2.3.10	CAN_RX_MAILBOXES . . . . .	38
7.2.3.11	PLATFORM_CAN_USE_CAN1 . . . . .	38
7.2.4	Typedef Documentation . . . . .	39
7.2.4.1	canmbx_t . . . . .	39
7.2.5	Enumeration Type Documentation . . . . .	39
7.2.5.1	canstate_t . . . . .	39
7.2.6	Function Documentation . . . . .	39
7.2.6.1	canInit . . . . .	39
7.2.6.2	canObjectInit . . . . .	39
7.2.6.3	canStart . . . . .	40
7.2.6.4	canStop . . . . .	41
7.2.6.5	canTryTransmit . . . . .	41
7.2.6.6	canTryReceive . . . . .	42
7.2.6.7	canTransmit . . . . .	43
7.2.6.8	canReceive . . . . .	44
7.2.6.9	canSleep . . . . .	45
7.2.6.10	canWakeup . . . . .	46
7.2.6.11	can_lld_init . . . . .	47
7.2.6.12	can_lld_start . . . . .	47
7.2.6.13	can_lld_stop . . . . .	48
7.2.6.14	can_lld_is_tx_empty . . . . .	48
7.2.6.15	can_lld_transmit . . . . .	48
7.2.6.16	can_lld_is_rx_nonempty . . . . .	48
7.2.6.17	can_lld_receive . . . . .	49
7.2.6.18	can_lld_sleep . . . . .	49
7.2.6.19	can_lld_wakeup . . . . .	49
7.2.7	Variable Documentation . . . . .	50
7.2.7.1	CAND1 . . . . .	50
7.3	DAC Driver . . . . .	51
7.3.1	Detailed Description . . . . .	51
7.3.2	Macro Definition Documentation . . . . .	53
7.3.2.1	DAC_USE_WAIT . . . . .	53
7.3.2.2	DAC_USE_MUTUAL_EXCLUSION . . . . .	53
7.3.2.3	_dac_wait_s . . . . .	53
7.3.2.4	_dac_reset_i . . . . .	54

7.3.2.5	<a href="#">_dac_reset_s</a>	54
7.3.2.6	<a href="#">_dac_wakeup_isr</a>	54
7.3.2.7	<a href="#">_dac_timeout_isr</a>	55
7.3.2.8	<a href="#">_dac_isr_half_code</a>	55
7.3.2.9	<a href="#">_dac_isr_full_code</a>	56
7.3.2.10	<a href="#">_dac_isr_error_code</a>	56
7.3.2.11	<a href="#">DAC_MAX_CHANNELS</a>	57
7.3.2.12	<a href="#">PLATFORM_DAC_USE_DAC1</a>	57
7.3.3	<a href="#">Typedef Documentation</a>	57
7.3.3.1	<a href="#">dacchannel_t</a>	57
7.3.3.2	<a href="#">DACDriver</a>	57
7.3.3.3	<a href="#">dacsample_t</a>	57
7.3.3.4	<a href="#">daccallback_t</a>	57
7.3.3.5	<a href="#">dacerrorcallback_t</a>	57
7.3.4	<a href="#">Enumeration Type Documentation</a>	58
7.3.4.1	<a href="#">dacstate_t</a>	58
7.3.4.2	<a href="#">dacerror_t</a>	58
7.3.5	<a href="#">Function Documentation</a>	58
7.3.5.1	<a href="#">dacInit</a>	58
7.3.5.2	<a href="#">dacObjectInit</a>	59
7.3.5.3	<a href="#">dacStart</a>	59
7.3.5.4	<a href="#">dacStop</a>	60
7.3.5.5	<a href="#">dacPutChannelX</a>	60
7.3.5.6	<a href="#">dacStartConversion</a>	61
7.3.5.7	<a href="#">dacStartConversionl</a>	61
7.3.5.8	<a href="#">dacStopConversion</a>	62
7.3.5.9	<a href="#">dacStopConversionl</a>	63
7.3.5.10	<a href="#">dacConvert</a>	63
7.3.5.11	<a href="#">dacAcquireBus</a>	64
7.3.5.12	<a href="#">dacReleaseBus</a>	65
7.3.5.13	<a href="#">dac_lld_init</a>	65
7.3.5.14	<a href="#">dac_lld_start</a>	65
7.3.5.15	<a href="#">dac_lld_stop</a>	66
7.3.5.16	<a href="#">dac_lld_put_channel</a>	66
7.3.5.17	<a href="#">dac_lld_start_conversion</a>	66
7.3.5.18	<a href="#">dac_lld_stop_conversion</a>	67
7.3.6	<a href="#">Variable Documentation</a>	67
7.3.6.1	<a href="#">DACD1</a>	67
7.4	<a href="#">EXT Driver</a>	68
7.4.1	<a href="#">Detailed Description</a>	68

---

7.4.2	Driver State Machine . . . . .	68
7.4.3	EXT Operations. . . . .	68
7.4.4	Macro Definition Documentation . . . . .	70
7.4.4.1	EXT_CH_MODE_EDGES_MASK . . . . .	70
7.4.4.2	EXT_CH_MODE_DISABLED . . . . .	70
7.4.4.3	EXT_CH_MODE_RISING_EDGE . . . . .	70
7.4.4.4	EXT_CH_MODE_FALLING_EDGE . . . . .	71
7.4.4.5	EXT_CH_MODE_BOTH_EDGES . . . . .	71
7.4.4.6	EXT_CH_MODE_AUTOSTART . . . . .	71
7.4.4.7	extChannelEnable . . . . .	71
7.4.4.8	extChannelDisable . . . . .	71
7.4.4.9	extSetChannelMode . . . . .	71
7.4.4.10	EXT_MAX_CHANNELS . . . . .	72
7.4.4.11	PLATFORM_EXT_USE_EXT1 . . . . .	72
7.4.5	Typedef Documentation . . . . .	72
7.4.5.1	EXTDriver . . . . .	72
7.4.5.2	expchannel_t . . . . .	72
7.4.5.3	extcallback_t . . . . .	72
7.4.6	Enumeration Type Documentation . . . . .	72
7.4.6.1	extstate_t . . . . .	72
7.4.7	Function Documentation . . . . .	73
7.4.7.1	extInit . . . . .	73
7.4.7.2	extObjectInit . . . . .	73
7.4.7.3	extStart . . . . .	73
7.4.7.4	extStop . . . . .	74
7.4.7.5	extChannelEnable . . . . .	74
7.4.7.6	extChannelDisable . . . . .	75
7.4.7.7	extSetChannelModel . . . . .	76
7.4.7.8	ext_lld_init . . . . .	76
7.4.7.9	ext_lld_start . . . . .	76
7.4.7.10	ext_lld_stop . . . . .	77
7.4.7.11	ext_lld_channel_enable . . . . .	77
7.4.7.12	ext_lld_channel_disable . . . . .	77
7.4.8	Variable Documentation . . . . .	77
7.4.8.1	EXTD1 . . . . .	77
7.5	GPT Driver . . . . .	78
7.5.1	Detailed Description . . . . .	78
7.5.2	Driver State Machine . . . . .	78
7.5.3	GPT Operations. . . . .	78
7.5.4	Macro Definition Documentation . . . . .	81

7.5.4.1	gptChangeInterval . . . . .	81
7.5.4.2	gptGetIntervalX . . . . .	81
7.5.4.3	gptGetCounterX . . . . .	81
7.5.4.4	PLATFORM_GPT_USE_GPT1 . . . . .	83
7.5.4.5	gpt_lld_change_interval . . . . .	83
7.5.5	Typedef Documentation . . . . .	84
7.5.5.1	GPTDriver . . . . .	84
7.5.5.2	gptcallback_t . . . . .	84
7.5.5.3	gptfreq_t . . . . .	84
7.5.5.4	gptcnt_t . . . . .	84
7.5.6	Enumeration Type Documentation . . . . .	84
7.5.6.1	gptstate_t . . . . .	84
7.5.7	Function Documentation . . . . .	84
7.5.7.1	gptInit . . . . .	84
7.5.7.2	gptObjectInit . . . . .	85
7.5.7.3	gptStart . . . . .	85
7.5.7.4	gptStop . . . . .	86
7.5.7.5	gptChangeInterval . . . . .	87
7.5.7.6	gptStartContinuous . . . . .	88
7.5.7.7	gptStartContinuousl . . . . .	89
7.5.7.8	gptStartOneShot . . . . .	90
7.5.7.9	gptStartOneShotl . . . . .	90
7.5.7.10	gptStopTimer . . . . .	91
7.5.7.11	gptStopTimerl . . . . .	91
7.5.7.12	gptPolledDelay . . . . .	92
7.5.7.13	gpt_lld_init . . . . .	92
7.5.7.14	gpt_lld_start . . . . .	93
7.5.7.15	gpt_lld_stop . . . . .	93
7.5.7.16	gpt_lld_start_timer . . . . .	93
7.5.7.17	gpt_lld_stop_timer . . . . .	93
7.5.7.18	gpt_lld_polled_delay . . . . .	94
7.5.8	Variable Documentation . . . . .	94
7.5.8.1	GPTD1 . . . . .	94
7.6	HAL Driver . . . . .	95
7.6.1	Detailed Description . . . . .	95
7.6.2	Macro Definition Documentation . . . . .	96
7.6.2.1	_CHIBIOS_HAL_ . . . . .	96
7.6.2.2	CH_HAL_STABLE . . . . .	96
7.6.2.3	HAL_VERSION . . . . .	96
7.6.2.4	CH_HAL_MAJOR . . . . .	96

7.6.2.5	CH_HAL_MINOR	96
7.6.2.6	CH_HAL_PATCH	96
7.6.3	Function Documentation	96
7.6.3.1	hallInit	96
7.6.3.2	hal_lld_init	97
7.7	I/O Buffers Queues	98
7.7.1	Detailed Description	98
7.7.2	Macro Definition Documentation	99
7.7.2.1	BQ_BUFFER_SIZE	99
7.7.2.2	bqSizeX	100
7.7.2.3	bqSpaceI	100
7.7.2.4	bqGetLinkX	100
7.7.2.5	ibqlIsEmptyI	100
7.7.2.6	ibqlIsFullI	101
7.7.2.7	obqlIsEmptyI	101
7.7.2.8	obqlIsFullI	102
7.7.3	Typedef Documentation	102
7.7.3.1	io_buffers_queue_t	102
7.7.3.2	bqnotify_t	102
7.7.3.3	input_buffers_queue_t	103
7.7.3.4	output_buffers_queue_t	103
7.7.4	Function Documentation	103
7.7.4.1	ibqObjectInit	103
7.7.4.2	ibqResetI	103
7.7.4.3	ibqGetEmptyBufferI	104
7.7.4.4	ibqPostFullBufferI	104
7.7.4.5	ibqGetFullBufferTimeout	105
7.7.4.6	ibqGetFullBufferTimeoutS	106
7.7.4.7	ibqReleaseEmptyBuffer	107
7.7.4.8	ibqReleaseEmptyBufferS	107
7.7.4.9	ibqGetTimeout	108
7.7.4.10	ibqReadTimeout	109
7.7.4.11	obqObjectInit	109
7.7.4.12	obqResetI	110
7.7.4.13	obqGetFullBufferI	111
7.7.4.14	obqReleaseEmptyBufferI	111
7.7.4.15	obqGetEmptyBufferTimeout	111
7.7.4.16	obqGetEmptyBufferTimeoutS	112
7.7.4.17	obqPostFullBuffer	113
7.7.4.18	obqPostFullBufferS	114

7.7.4.19	obqPutTimeout . . . . .	114
7.7.4.20	obqWriteTimeout . . . . .	115
7.7.4.21	obqTryFlushl . . . . .	116
7.7.4.22	obqFlush . . . . .	117
7.8	Abstract I/O Channel . . . . .	118
7.8.1	Detailed Description . . . . .	118
7.8.2	Macro Definition Documentation . . . . .	119
7.8.2.1	_base_channel_methods . . . . .	119
7.8.2.2	_base_channel_data . . . . .	119
7.8.2.3	chnPutTimeout . . . . .	119
7.8.2.4	chnGetTimeout . . . . .	120
7.8.2.5	chnWrite . . . . .	120
7.8.2.6	chnWriteTimeout . . . . .	121
7.8.2.7	chnRead . . . . .	121
7.8.2.8	chnReadTimeout . . . . .	122
7.8.2.9	CHN_NO_ERROR . . . . .	122
7.8.2.10	CHN_CONNECTED . . . . .	122
7.8.2.11	CHN_DISCONNECTED . . . . .	122
7.8.2.12	CHN_INPUT_AVAILABLE . . . . .	122
7.8.2.13	CHN_OUTPUT_EMPTY . . . . .	122
7.8.2.14	CHN_TRANSMISSION_END . . . . .	122
7.8.2.15	_base_asynchronous_channel_methods . . . . .	122
7.8.2.16	_base_asynchronous_channel_data . . . . .	123
7.8.2.17	chnGetEventSource . . . . .	123
7.8.2.18	chnAddFlagsl . . . . .	123
7.9	Abstract Files . . . . .	124
7.9.1	Detailed Description . . . . .	124
7.9.2	Macro Definition Documentation . . . . .	125
7.9.2.1	FILE_OK . . . . .	125
7.9.2.2	FILE_ERROR . . . . .	125
7.9.2.3	FILE_EOF . . . . .	125
7.9.2.4	_file_stream_methods . . . . .	125
7.9.2.5	_file_stream_data . . . . .	125
7.9.2.6	fileStreamWrite . . . . .	125
7.9.2.7	fileStreamRead . . . . .	126
7.9.2.8	fileStreamPut . . . . .	126
7.9.2.9	fileStreamGet . . . . .	127
7.9.2.10	fileStreamClose . . . . .	127
7.9.2.11	fileStreamGetError . . . . .	128
7.9.2.12	fileStreamGetSize . . . . .	128

7.9.2.13	fileStreamGetPosition . . . . .	128
7.9.2.14	fileStreamSeek . . . . .	129
7.9.3	Typedef Documentation . . . . .	129
7.9.3.1	fileoffset_t . . . . .	129
7.10	Abstract I/O Block Device . . . . .	130
7.10.1	Detailed Description . . . . .	130
7.10.2	Driver State Machine . . . . .	130
7.10.3	Macro Definition Documentation . . . . .	131
7.10.3.1	_base_block_device_methods . . . . .	131
7.10.3.2	_base_block_device_data . . . . .	132
7.10.3.3	blkGetDriverState . . . . .	132
7.10.3.4	blkIsTransferring . . . . .	132
7.10.3.5	blkIsInserted . . . . .	133
7.10.3.6	blkIsWriteProtected . . . . .	133
7.10.3.7	blkConnect . . . . .	133
7.10.3.8	blkDisconnect . . . . .	134
7.10.3.9	blkRead . . . . .	134
7.10.3.10	blkWrite . . . . .	135
7.10.3.11	blkSync . . . . .	135
7.10.3.12	blkGetInfo . . . . .	135
7.10.4	Enumeration Type Documentation . . . . .	136
7.10.4.1	blkstate_t . . . . .	136
7.11	I/O Bytes Queues . . . . .	137
7.11.1	Detailed Description . . . . .	137
7.11.2	Macro Definition Documentation . . . . .	138
7.11.2.1	Q_OK . . . . .	138
7.11.2.2	Q_TIMEOUT . . . . .	139
7.11.2.3	Q_RESET . . . . .	139
7.11.2.4	Q_EMPTY . . . . .	139
7.11.2.5	Q_FULL . . . . .	139
7.11.2.6	qSizeX . . . . .	139
7.11.2.7	qSpaceI . . . . .	139
7.11.2.8	qGetLink . . . . .	140
7.11.2.9	iqGetFullI . . . . .	140
7.11.2.10	iqGetEmptyI . . . . .	140
7.11.2.11	iqIsEmptyI . . . . .	141
7.11.2.12	iqIsFullI . . . . .	141
7.11.2.13	iqGet . . . . .	142
7.11.2.14	oqGetFullI . . . . .	142
7.11.2.15	oqGetEmptyI . . . . .	142

7.11.2.16 <code>oqlIsEmptyl</code>	143
7.11.2.17 <code>oqlsFulll</code>	143
7.11.2.18 <code>oqPut</code>	144
7.11.3 <b>Typedef Documentation</b>	144
7.11.3.1 <code>io_queue_t</code>	144
7.11.3.2 <code>qnotify_t</code>	144
7.11.3.3 <code>input_queue_t</code>	144
7.11.3.4 <code>output_queue_t</code>	144
7.11.4 <b>Function Documentation</b>	144
7.11.4.1 <code>iqObjectInit</code>	145
7.11.4.2 <code>iqResetl</code>	145
7.11.4.3 <code>iqPutl</code>	146
7.11.4.4 <code>iqGetTimeout</code>	146
7.11.4.5 <code>iqReadTimeout</code>	147
7.11.4.6 <code>oqObjectInit</code>	148
7.11.4.7 <code>oqResetl</code>	149
7.11.4.8 <code>oqPutTimeout</code>	149
7.11.4.9 <code>oqGetl</code>	150
7.11.4.10 <code>oqWriteTimeout</code>	151
7.12 <b>Abstract Streams</b>	153
7.12.1 <b>Detailed Description</b>	153
7.12.2 <b>Macro Definition Documentation</b>	153
7.12.2.1 <code>_base_sequential_stream_methods</code>	153
7.12.2.2 <code>_base_sequential_stream_data</code>	154
7.12.2.3 <code>streamWrite</code>	154
7.12.2.4 <code>streamRead</code>	154
7.12.2.5 <code>streamPut</code>	154
7.12.2.6 <code>streamGet</code>	155
7.13 <b>I2C Driver</b>	156
7.13.1 <b>Detailed Description</b>	156
7.13.2 <b>Driver State Machine</b>	156
7.13.3 <b>Macro Definition Documentation</b>	158
7.13.3.1 <code>I2C_NO_ERROR</code>	158
7.13.3.2 <code>I2C_BUS_ERROR</code>	159
7.13.3.3 <code>I2C_ARBITRATION_LOST</code>	159
7.13.3.4 <code>I2C_ACK_FAILURE</code>	159
7.13.3.5 <code>I2C_OVERRUN</code>	159
7.13.3.6 <code>I2C_PEC_ERROR</code>	159
7.13.3.7 <code>I2C_TIMEOUT</code>	159
7.13.3.8 <code>I2C_SMB_ALERT</code>	159

---

7.13.3.9 I2C_USE_MUTUAL_EXCLUSION . . . . .	159
7.13.3.10 _i2c_wakeup_isr . . . . .	159
7.13.3.11 _i2c_wakeup_error_isr . . . . .	160
7.13.3.12 i2cMasterTransmit . . . . .	160
7.13.3.13 i2cMasterReceive . . . . .	160
7.13.3.14 PLATFORM_I2C_USE_I2C1 . . . . .	160
7.13.3.15 i2c_lld_get_errors . . . . .	160
7.13.4 Typedef Documentation . . . . .	161
7.13.4.1 i2caddr_t . . . . .	161
7.13.4.2 i2cflags_t . . . . .	161
7.13.4.3 I2CDriver . . . . .	161
7.13.5 Enumeration Type Documentation . . . . .	161
7.13.5.1 i2cstate_t . . . . .	161
7.13.6 Function Documentation . . . . .	161
7.13.6.1 i2cInit . . . . .	161
7.13.6.2 i2cObjectInit . . . . .	162
7.13.6.3 i2cStart . . . . .	162
7.13.6.4 i2cStop . . . . .	163
7.13.6.5 i2cGetErrors . . . . .	163
7.13.6.6 i2cMasterTransmitTimeout . . . . .	164
7.13.6.7 i2cMasterReceiveTimeout . . . . .	165
7.13.6.8 i2cAcquireBus . . . . .	166
7.13.6.9 i2cReleaseBus . . . . .	166
7.13.6.10 i2c_lld_init . . . . .	167
7.13.6.11 i2c_lld_start . . . . .	167
7.13.6.12 i2c_lld_stop . . . . .	167
7.13.6.13 i2c_lld_master_receive_timeout . . . . .	168
7.13.6.14 i2c_lld_master_transmit_timeout . . . . .	168
7.13.7 Variable Documentation . . . . .	169
7.13.7.1 I2CD1 . . . . .	169
7.14 I2S Driver . . . . .	170
7.14.1 Detailed Description . . . . .	170
7.14.2 Driver State Machine . . . . .	170
7.14.3 Macro Definition Documentation . . . . .	171
7.14.3.1 i2sStartExchangel . . . . .	171
7.14.3.2 i2sStopExchangel . . . . .	172
7.14.3.3 _i2s_isr_half_code . . . . .	172
7.14.3.4 _i2s_isr_full_code . . . . .	172
7.14.3.5 PLATFORM_I2S_USE_I2S1 . . . . .	173
7.14.4 Typedef Documentation . . . . .	173

7.14.4.1 I2SDriver . . . . .	173
7.14.4.2 i2scallback_t . . . . .	173
7.14.5 Enumeration Type Documentation . . . . .	173
7.14.5.1 i2ssstate_t . . . . .	173
7.14.6 Function Documentation . . . . .	174
7.14.6.1 i2sInit . . . . .	174
7.14.6.2 i2sObjectInit . . . . .	174
7.14.6.3 i2sStart . . . . .	174
7.14.6.4 i2sStop . . . . .	175
7.14.6.5 i2sStartExchange . . . . .	175
7.14.6.6 i2sStopExchange . . . . .	176
7.14.6.7 i2s_lld_init . . . . .	176
7.14.6.8 i2s_lld_start . . . . .	177
7.14.7 Variable Documentation . . . . .	177
7.14.7.1 I2SD1 . . . . .	177
7.15 ICU Driver . . . . .	178
7.15.1 Detailed Description . . . . .	178
7.15.2 Driver State Machine . . . . .	178
7.15.3 ICU Operations. . . . .	178
7.15.4 Macro Definition Documentation . . . . .	181
7.15.4.1 icuStartCaptureI . . . . .	181
7.15.4.2 icuStopCaptureI . . . . .	181
7.15.4.3 icuEnableNotificationsI . . . . .	182
7.15.4.4 icuDisableNotificationsI . . . . .	182
7.15.4.5 icuAreNotificationsEnabledX . . . . .	182
7.15.4.6 icuGetWidthX . . . . .	183
7.15.4.7 icuGetPeriodX . . . . .	183
7.15.4.8 _icu_isr_invoke_width_cb . . . . .	184
7.15.4.9 _icu_isr_invoke_period_cb . . . . .	184
7.15.4.10 _icu_isr_invoke_overflow_cb . . . . .	184
7.15.4.11 PLATFORM_ICU_USE_ICU1 . . . . .	185
7.15.4.12 icu_lld_get_width . . . . .	185
7.15.4.13 icu_lld_get_period . . . . .	185
7.15.4.14 icu_lld_are_notifications_enabled . . . . .	185
7.15.5 Typedef Documentation . . . . .	186
7.15.5.1 ICUDriver . . . . .	186
7.15.5.2 icucallback_t . . . . .	186
7.15.5.3 icufreq_t . . . . .	186
7.15.5.4 icucnt_t . . . . .	186
7.15.6 Enumeration Type Documentation . . . . .	186

7.15.6.1	icustate_t . . . . .	186
7.15.6.2	icumode_t . . . . .	187
7.15.7	Function Documentation . . . . .	187
7.15.7.1	icuInit . . . . .	187
7.15.7.2	icuObjectInit . . . . .	187
7.15.7.3	icuStart . . . . .	187
7.15.7.4	icuStop . . . . .	188
7.15.7.5	icuStartCapture . . . . .	189
7.15.7.6	icuWaitCapture . . . . .	190
7.15.7.7	icuStopCapture . . . . .	191
7.15.7.8	icuEnableNotifications . . . . .	191
7.15.7.9	icuDisableNotifications . . . . .	192
7.15.7.10	icu_lld_init . . . . .	193
7.15.7.11	icu_lld_start . . . . .	193
7.15.7.12	icu_lld_stop . . . . .	193
7.15.7.13	icu_lld_start_capture . . . . .	194
7.15.7.14	icu_lld_wait_capture . . . . .	194
7.15.7.15	icu_lld_stop_capture . . . . .	194
7.15.7.16	icu_lld_enable_notifications . . . . .	195
7.15.7.17	icu_lld_disable_notifications . . . . .	195
7.15.8	Variable Documentation . . . . .	195
7.15.8.1	ICUD1 . . . . .	195
7.16	MAC Driver . . . . .	196
7.16.1	Detailed Description . . . . .	196
7.16.2	Macro Definition Documentation . . . . .	198
7.16.2.1	MAC_USE_ZERO_COPY . . . . .	198
7.16.2.2	MAC_USE_EVENTS . . . . .	198
7.16.2.3	macGetReceiveEventSource . . . . .	198
7.16.2.4	macWriteTransmitDescriptor . . . . .	198
7.16.2.5	macReadReceiveDescriptor . . . . .	199
7.16.2.6	macGetNextTransmitBuffer . . . . .	199
7.16.2.7	macGetNextReceiveBuffer . . . . .	200
7.16.2.8	MAC_SUPPORTS_ZERO_COPY . . . . .	200
7.16.2.9	PLATFORM_MAC_USE_MAC1 . . . . .	200
7.16.3	Typedef Documentation . . . . .	200
7.16.3.1	MACDriver . . . . .	200
7.16.4	Enumeration Type Documentation . . . . .	200
7.16.4.1	macstate_t . . . . .	200
7.16.5	Function Documentation . . . . .	201
7.16.5.1	macInit . . . . .	201

7.16.5.2	macObjectInit . . . . .	201
7.16.5.3	macStart . . . . .	201
7.16.5.4	macStop . . . . .	202
7.16.5.5	macWaitTransmitDescriptor . . . . .	203
7.16.5.6	macReleaseTransmitDescriptor . . . . .	204
7.16.5.7	macWaitReceiveDescriptor . . . . .	205
7.16.5.8	macReleaseReceiveDescriptor . . . . .	206
7.16.5.9	macPollLinkStatus . . . . .	207
7.16.5.10	mac_lld_init . . . . .	207
7.16.5.11	mac_lld_start . . . . .	208
7.16.5.12	mac_lld_stop . . . . .	208
7.16.5.13	mac_lld_get_transmit_descriptor . . . . .	208
7.16.5.14	mac_lld_release_transmit_descriptor . . . . .	209
7.16.5.15	mac_lld_get_receive_descriptor . . . . .	209
7.16.5.16	mac_lld_release_receive_descriptor . . . . .	209
7.16.5.17	mac_lld_poll_link_status . . . . .	209
7.16.5.18	mac_lld_write_transmit_descriptor . . . . .	210
7.16.5.19	mac_lld_read_receive_descriptor . . . . .	210
7.16.5.20	mac_lld_get_next_transmit_buffer . . . . .	211
7.16.5.21	mac_lld_get_next_receive_buffer . . . . .	211
7.16.6	Variable Documentation . . . . .	211
7.16.6.1	ETHD1 . . . . .	211
7.17	HAL . . . . .	212
7.17.1	Detailed Description . . . . .	212
7.17.2	HAL Device Drivers Architecture . . . . .	212
7.17.3	HAL Normal Device Drivers . . . . .	212
7.17.3.1	Diagram . . . . .	213
7.17.4	HAL Complex Device Drivers . . . . .	213
7.17.5	HAL Interfaces . . . . .	213
7.17.6	HAL Inner Code . . . . .	213
7.18	Configuration . . . . .	215
7.18.1	Detailed Description . . . . .	215
7.18.2	Macro Definition Documentation . . . . .	217
7.18.2.1	HAL_USE_PAL . . . . .	217
7.18.2.2	HAL_USE_ADC . . . . .	217
7.18.2.3	HAL_USE_CAN . . . . .	217
7.18.2.4	HAL_USE_DAC . . . . .	217
7.18.2.5	HAL_USE_EXT . . . . .	217
7.18.2.6	HAL_USE_GPT . . . . .	218
7.18.2.7	HAL_USE_I2C . . . . .	218

---

---

7.18.2.8 HAL_USE_I2S . . . . .	218
7.18.2.9 HAL_USE_ICU . . . . .	218
7.18.2.10 HAL_USE_MAC . . . . .	218
7.18.2.11 HAL_USE_MMC_SPI . . . . .	218
7.18.2.12 HAL_USE_PWM . . . . .	218
7.18.2.13 HAL_USE_RTC . . . . .	218
7.18.2.14 HAL_USE_SDC . . . . .	218
7.18.2.15 HAL_USE_SERIAL . . . . .	218
7.18.2.16 HAL_USE_SERIAL_USB . . . . .	218
7.18.2.17 HAL_USE_SPI . . . . .	218
7.18.2.18 HAL_USE_UART . . . . .	219
7.18.2.19 HAL_USE_USB . . . . .	219
7.18.2.20 HAL_USE_WDG . . . . .	219
7.18.2.21 ADC_USE_WAIT . . . . .	219
7.18.2.22 ADC_USE_MUTUAL_EXCLUSION . . . . .	219
7.18.2.23 CAN_USE_SLEEP_MODE . . . . .	219
7.18.2.24 I2C_USE_MUTUAL_EXCLUSION . . . . .	219
7.18.2.25 MAC_USE_ZERO_COPY . . . . .	219
7.18.2.26 MAC_USE_EVENTS . . . . .	219
7.18.2.27 MMC_NICE_WAITING . . . . .	219
7.18.2.28 SDC_INIT_RETRY . . . . .	220
7.18.2.29 SDC_MMC_SUPPORT . . . . .	220
7.18.2.30 SDC_NICE_WAITING . . . . .	220
7.18.2.31 SERIAL_DEFAULT_BITRATE . . . . .	220
7.18.2.32 SERIAL_BUFFERS_SIZE . . . . .	220
7.18.2.33 SERIAL_USB_BUFFERS_SIZE . . . . .	220
7.18.2.34 SERIAL_USB_BUFFERS_NUMBER . . . . .	220
7.18.2.35 SPI_USE_WAIT . . . . .	221
7.18.2.36 SPI_USE_MUTUAL_EXCLUSION . . . . .	221
7.18.2.37 UART_USE_WAIT . . . . .	221
7.18.2.38 UART_USE_MUTUAL_EXCLUSION . . . . .	221
7.18.2.39 USB_USE_WAIT . . . . .	221
7.19 Normal Drivers . . . . .	222
7.19.1 Detailed Description . . . . .	222
7.20 Complex Drivers . . . . .	223
7.20.1 Detailed Description . . . . .	223
7.21 Interfaces . . . . .	224
7.21.1 Detailed Description . . . . .	224
7.22 Inner Code . . . . .	225
7.22.1 Detailed Description . . . . .	225

---

7.23 Support Code . . . . .	226
7.23.1 Detailed Description . . . . .	226
7.24 OSAL . . . . .	227
7.24.1 Detailed Description . . . . .	227
7.24.2 Macro Definition Documentation . . . . .	232
7.24.2.1 OSAL_ST_RESOLUTION . . . . .	232
7.24.2.2 OSAL_ST_FREQUENCY . . . . .	232
7.24.2.3 OSAL_ST_MODE . . . . .	232
7.24.2.4 OSAL_IRQ_PRIORITY_LEVELS . . . . .	232
7.24.2.5 OSAL_IRQ_MAXIMUM_PRIORITY . . . . .	232
7.24.2.6 OSAL_DBG_ENABLE_ASSERTS . . . . .	232
7.24.2.7 OSAL_DBG_ENABLE_CHECKS . . . . .	232
7.24.2.8 osalDbgAssert . . . . .	232
7.24.2.9 osalDbgCheck . . . . .	233
7.24.2.10 osalDbgCheckClassI . . . . .	233
7.24.2.11 osalDbgCheckClassS . . . . .	233
7.24.2.12 OSAL_IRQ_IS_VALID_PRIORITY . . . . .	234
7.24.2.13 OSAL_IRQ_PROLOGUE . . . . .	234
7.24.2.14 OSAL_IRQ_EPILOGUE . . . . .	234
7.24.2.15 OSAL_IRQ_HANDLER . . . . .	234
7.24.2.16 OSAL_S2ST . . . . .	234
7.24.2.17 OSAL_MS2ST . . . . .	234
7.24.2.18 OSAL_US2ST . . . . .	235
7.24.2.19 OSAL_S2RTC . . . . .	235
7.24.2.20 OSAL_MS2RTC . . . . .	236
7.24.2.21 OSAL_US2RTC . . . . .	236
7.24.2.22 osalThreadSleepSeconds . . . . .	237
7.24.2.23 osalThreadSleepMilliseconds . . . . .	237
7.24.2.24 osalThreadSleepMicroseconds . . . . .	237
7.24.3 Ttypedef Documentation . . . . .	237
7.24.3.1 syssts_t . . . . .	237
7.24.3.2 msg_t . . . . .	238
7.24.3.3 systime_t . . . . .	238
7.24.3.4 rtcnt_t . . . . .	238
7.24.3.5 thread_reference_t . . . . .	238
7.24.3.6 event_source_t . . . . .	238
7.24.3.7 eventcallback_t . . . . .	238
7.24.3.8 eventflags_t . . . . .	238
7.24.3.9 mutex_t . . . . .	238
7.24.4 Function Documentation . . . . .	238

---

7.24.4.1	osallInit	238
7.24.4.2	osalSysHalt	239
7.24.4.3	osalSysPolledDelayX	239
7.24.4.4	osalOsTimerHandlerI	239
7.24.4.5	osalOsRescheduleS	240
7.24.4.6	osalOsGetSystemTimeX	240
7.24.4.7	osalThreadSleepS	240
7.24.4.8	osalThreadSleep	240
7.24.4.9	osalThreadSuspendS	241
7.24.4.10	osalThreadSuspendTimeoutS	241
7.24.4.11	osalThreadResumeI	242
7.24.4.12	osalThreadResumeS	242
7.24.4.13	osalThreadEnqueueTimeoutS	242
7.24.4.14	osalThreadDequeueNextI	243
7.24.4.15	osalThreadDequeueAllI	243
7.24.4.16	osalEventBroadcastFlagsI	243
7.24.4.17	osalEventBroadcastFlags	244
7.24.4.18	osalEventSetCallback	244
7.24.4.19	osalMutexLock	245
7.24.4.20	osalMutexUnlock	245
7.24.4.21	osalSysDisable	245
7.24.4.22	osalSysEnable	246
7.24.4.23	osalSysLock	246
7.24.4.24	osalSysUnlock	246
7.24.4.25	osalSysLockFromISR	246
7.24.4.26	osalSysUnlockFromISR	246
7.24.4.27	osalSysGetStatusAndLockX	247
7.24.4.28	osalSysRestoreStatusX	247
7.24.4.29	osalOsIsTimeWithinX	247
7.24.4.30	osalThreadQueueObjectInit	248
7.24.4.31	osalEventObjectInit	248
7.24.4.32	osalMutexObjectInit	248
7.24.5	Variable Documentation	248
7.24.5.1	osal_halt_msg	248
7.24.5.2	osal_halt_msg	249
7.25	MII/RMII Header	250
7.25.1	Detailed Description	250
7.25.2	Macro Definition Documentation	252
7.25.2.1	MII_BMCR	252
7.25.2.2	MII_BMSR	252

---

7.25.2.3 MII_PHYSID1 . . . . .	252
7.25.2.4 MII_PHYSID2 . . . . .	252
7.25.2.5 MII_ADVERTISE . . . . .	252
7.25.2.6 MII_LPA . . . . .	252
7.25.2.7 MII_EXPANSION . . . . .	253
7.25.2.8 MII_ANNPTR . . . . .	253
7.25.2.9 MII_CTRL1000 . . . . .	253
7.25.2.10 MII_STAT1000 . . . . .	253
7.25.2.11 MII_ESTATUS . . . . .	253
7.25.2.12 MII_PHYSTS . . . . .	253
7.25.2.13 MII_MICR . . . . .	253
7.25.2.14 MII_DCOUNTER . . . . .	253
7.25.2.15 MII_FCSCOUNTER . . . . .	253
7.25.2.16 MII_NWAYTEST . . . . .	253
7.25.2.17 MII_RERRCOUNTER . . . . .	253
7.25.2.18 MII_SREVISION . . . . .	253
7.25.2.19 MII_RESV1 . . . . .	254
7.25.2.20 MII_LBRERROR . . . . .	254
7.25.2.21 MII_PHYADDR . . . . .	254
7.25.2.22 MII_RESV2 . . . . .	254
7.25.2.23 MII_TPISTATUS . . . . .	254
7.25.2.24 MII_NCONFIG . . . . .	254
7.25.2.25 BMCR_RESV . . . . .	254
7.25.2.26 BMCR_CTST . . . . .	254
7.25.2.27 BMCR_FULLDPLX . . . . .	254
7.25.2.28 BMCR_ANRESTART . . . . .	254
7.25.2.29 BMCR_ISOLATE . . . . .	254
7.25.2.30 BMCR_PDOWN . . . . .	254
7.25.2.31 BMCR_ANENABLE . . . . .	255
7.25.2.32 BMCR_SPEED100 . . . . .	255
7.25.2.33 BMCR_LOOPBACK . . . . .	255
7.25.2.34 BMCR_RESET . . . . .	255
7.25.2.35 BMSR_ERCAP . . . . .	255
7.25.2.36 BMSR_JCD . . . . .	255
7.25.2.37 BMSR_LSTATUS . . . . .	255
7.25.2.38 BMSR_ANEGCAPABLE . . . . .	255
7.25.2.39 BMSR_RFAULT . . . . .	255
7.25.2.40 BMSR_ANEGCOMPLETE . . . . .	255
7.25.2.41 BMSR_MFPRESUPPCAP . . . . .	255
7.25.2.42 BMSR_RESV . . . . .	255

7.25.2.43 BMSR_10HALF . . . . .	256
7.25.2.44 BMSR_10FULL . . . . .	256
7.25.2.45 BMSR_100HALF . . . . .	256
7.25.2.46 BMSR_100FULL . . . . .	256
7.25.2.47 BMSR_100BASE4 . . . . .	256
7.25.2.48 ADVERTISE_SLCT . . . . .	256
7.25.2.49 ADVERTISE_CSMA . . . . .	256
7.25.2.50 ADVERTISE_10HALF . . . . .	256
7.25.2.51 ADVERTISE_10FULL . . . . .	256
7.25.2.52 ADVERTISE_100HALF . . . . .	256
7.25.2.53 ADVERTISE_100FULL . . . . .	256
7.25.2.54 ADVERTISE_100BASE4 . . . . .	256
7.25.2.55 ADVERTISE_PAUSE_CAP . . . . .	257
7.25.2.56 ADVERTISE_PAUSE_ASYM . . . . .	257
7.25.2.57 ADVERTISE_RESV . . . . .	257
7.25.2.58 ADVERTISE_RFAULT . . . . .	257
7.25.2.59 ADVERTISE_LPACK . . . . .	257
7.25.2.60 ADVERTISE_NPAGE . . . . .	257
7.25.2.61 LPA_SLCT . . . . .	257
7.25.2.62 LPA_10HALF . . . . .	257
7.25.2.63 LPA_10FULL . . . . .	257
7.25.2.64 LPA_100HALF . . . . .	257
7.25.2.65 LPA_100FULL . . . . .	257
7.25.2.66 LPA_100BASE4 . . . . .	257
7.25.2.67 LPA_PAUSE_CAP . . . . .	258
7.25.2.68 LPA_PAUSE_ASYM . . . . .	258
7.25.2.69 LPA_RESV . . . . .	258
7.25.2.70 LPA_RFAULT . . . . .	258
7.25.2.71 LPA_LPACK . . . . .	258
7.25.2.72 LPA_NPAGE . . . . .	258
7.25.2.73 EXPANSION_NWAY . . . . .	258
7.25.2.74 EXPANSION_LCWP . . . . .	258
7.25.2.75 EXPANSION_ENABLENPAGE . . . . .	258
7.25.2.76 EXPANSION_NPCAPABLE . . . . .	258
7.25.2.77 EXPANSION_MFAULTS . . . . .	258
7.25.2.78 EXPANSION_RESV . . . . .	258
7.25.2.79 NWAYTEST_RESV1 . . . . .	259
7.25.2.80 NWAYTEST_LOOPBACK . . . . .	259
7.25.2.81 NWAYTEST_RESV2 . . . . .	259
7.26 MMC over SPI Driver . . . . .	260

7.26.1	Detailed Description	260
7.26.2	Driver State Machine	260
7.26.3	Driver Operations	260
7.26.4	Macro Definition Documentation	262
7.26.4.1	MMC_NICE_WAITING	262
7.26.4.2	_mmc_driver_methods	262
7.26.4.3	mmclsCardInserted	262
7.26.4.4	mmclsWriteProtected	262
7.26.5	Function Documentation	263
7.26.5.1	crc7	263
7.26.5.2	wait	263
7.26.5.3	send_hdr	264
7.26.5.4	recv1	264
7.26.5.5	recv3	265
7.26.5.6	send_command_R1	266
7.26.5.7	send_command_R3	266
7.26.5.8	read_CxD	267
7.26.5.9	sync	268
7.26.5.10	mmcInit	269
7.26.5.11	mmcObjectInit	269
7.26.5.12	mmcStart	269
7.26.5.13	mmcStop	269
7.26.5.14	mmcConnect	270
7.26.5.15	mmcDisconnect	271
7.26.5.16	mmcStartSequentialRead	272
7.26.5.17	mmcSequentialRead	273
7.26.5.18	mmcStopSequentialRead	274
7.26.5.19	mmcStartSequentialWrite	275
7.26.5.20	mmcSequentialWrite	276
7.26.5.21	mmcStopSequentialWrite	277
7.26.5.22	mmcSync	278
7.26.5.23	mmcGetInfo	279
7.26.5.24	mmcErase	279
7.26.6	Variable Documentation	280
7.26.6.1	mmc_vmt	280
7.26.6.2	crc7_lookup_table	280
7.27	MMC/SD Block Device	282
7.27.1	Detailed Description	282
7.27.2	Macro Definition Documentation	286
7.27.2.1	MMCSD_BLOCK_SIZE	286

---

7.27.2.2 MMCSD_R1_ERROR_MASK . . . . .	286
7.27.2.3 MMCSD_CMD8_PATTERN . . . . .	286
7.27.2.4 MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE . . . . .	286
7.27.2.5 MMCSD_CID_SDC_CRC_SLICE . . . . .	286
7.27.2.6 _mmcsd_block_device_methods . . . . .	287
7.27.2.7 _mmcsd_block_device_data . . . . .	287
7.27.2.8 MMCSD_R1_ERROR . . . . .	287
7.27.2.9 MMCSD_R1_STS . . . . .	287
7.27.2.10 MMCSD_R1_IS_CARD_LOCKED . . . . .	287
7.27.2.11 mmcsdGetCardCapacity . . . . .	287
7.27.3 Function Documentation . . . . .	288
7.27.3.1 _mmcsd_get_slice . . . . .	288
7.27.3.2 _mmcsd_get_capacity . . . . .	288
7.27.3.3 _mmcsd_get_capacity_ext . . . . .	289
7.27.3.4 _mmcsd_unpack_sdc_cid . . . . .	289
7.27.3.5 _mmcsd_unpack_mmc_cid . . . . .	289
7.27.3.6 _mmcsd_unpack_csd_mmc . . . . .	290
7.27.3.7 _mmcsd_unpack_csd_v10 . . . . .	290
7.27.3.8 _mmcsd_unpack_csd_v20 . . . . .	291
7.28 PAL Driver . . . . .	292
7.28.1 Detailed Description . . . . .	292
7.28.2 Implementation Rules . . . . .	292
7.28.2.1 Writing on input pads . . . . .	292
7.28.2.2 Reading from output pads . . . . .	292
7.28.2.3 Writing unused or unimplemented port bits . . . . .	292
7.28.2.4 Reading from unused or unimplemented port bits . . . . .	293
7.28.2.5 Reading or writing on pins associated to other functionalities . . . . .	293
7.28.3 Macro Definition Documentation . . . . .	296
7.28.3.1 PAL_MODE_RESET . . . . .	296
7.28.3.2 PAL_MODE_UNCONNECTED . . . . .	296
7.28.3.3 PAL_MODE_INPUT . . . . .	296
7.28.3.4 PAL_MODE_INPUT_PULLUP . . . . .	296
7.28.3.5 PAL_MODE_INPUT_PULLDOWN . . . . .	296
7.28.3.6 PAL_MODE_INPUT_ANALOG . . . . .	296
7.28.3.7 PAL_MODE_OUTPUT_PUSHULL . . . . .	297
7.28.3.8 PAL_MODE_OUTPUT_OPENDRAIN . . . . .	297
7.28.3.9 PAL_LOW . . . . .	297
7.28.3.10 PAL_HIGH . . . . .	297
7.28.3.11 PAL_PORT_BIT . . . . .	297
7.28.3.12 PAL_GROUP_MASK . . . . .	297

7.28.3.13 _IOBUS_DATA . . . . .	297
7.28.3.14 IOBUS_DECL . . . . .	298
7.28.3.15 pallnit . . . . .	298
7.28.3.16 palReadPort . . . . .	298
7.28.3.17 palReadLatch . . . . .	299
7.28.3.18 palWritePort . . . . .	299
7.28.3.19 palSetPort . . . . .	299
7.28.3.20 palClearPort . . . . .	300
7.28.3.21 palTogglePort . . . . .	300
7.28.3.22 palReadGroup . . . . .	300
7.28.3.23 palWriteGroup . . . . .	301
7.28.3.24 palSetGroupMode . . . . .	301
7.28.3.25 palReadPad . . . . .	302
7.28.3.26 palWritePad . . . . .	302
7.28.3.27 palSetPad . . . . .	303
7.28.3.28 palClearPad . . . . .	303
7.28.3.29 palTogglePad . . . . .	304
7.28.3.30 palSetPadMode . . . . .	304
7.28.3.31 palReadLine . . . . .	304
7.28.3.32 palWriteLine . . . . .	305
7.28.3.33 palSetLine . . . . .	305
7.28.3.34 palClearLine . . . . .	305
7.28.3.35 palToggleLine . . . . .	306
7.28.3.36 palSetLineMode . . . . .	306
7.28.3.37 PAL_IOPORTS_WIDTH . . . . .	306
7.28.3.38 PAL_WHOLE_PORT . . . . .	306
7.28.3.39 PAL_LINE . . . . .	306
7.28.3.40 PAL_PORT . . . . .	307
7.28.3.41 PAL_PAD . . . . .	307
7.28.3.42 PAL_NOLINE . . . . .	307
7.28.3.43 IOPORT1 . . . . .	307
7.28.3.44 pal_lld_init . . . . .	307
7.28.3.45 pal_lld_readport . . . . .	307
7.28.3.46 pal_lld_readlatch . . . . .	307
7.28.3.47 pal_lld_writeport . . . . .	308
7.28.3.48 pal_lld_setport . . . . .	308
7.28.3.49 pal_lld_clearport . . . . .	309
7.28.3.50 pal_lld_toggleport . . . . .	309
7.28.3.51 pal_lld_readgroup . . . . .	309
7.28.3.52 pal_lld_writegroup . . . . .	310

---

7.28.3.53 pal_lld_setgroupmode . . . . .	310
7.28.3.54 pal_lld_readpad . . . . .	311
7.28.3.55 pal_lld_writepad . . . . .	311
7.28.3.56 pal_lld_setpad . . . . .	312
7.28.3.57 pal_lld_clearpad . . . . .	312
7.28.3.58 pal_lld_togglepad . . . . .	313
7.28.3.59 pal_lld_setpadmode . . . . .	313
7.28.4 Typedef Documentation . . . . .	314
7.28.4.1 ioportmask_t . . . . .	314
7.28.4.2 iomode_t . . . . .	314
7.28.4.3 ioline_t . . . . .	314
7.28.4.4 ioportid_t . . . . .	314
7.28.5 Function Documentation . . . . .	314
7.28.5.1 palReadBus . . . . .	314
7.28.5.2 palWriteBus . . . . .	315
7.28.5.3 palSetBusMode . . . . .	315
7.28.5.4 __pal_lld_init . . . . .	315
7.28.5.5 __pal_lld_setgroupmode . . . . .	316
7.29 PWM Driver . . . . .	318
7.29.1 Detailed Description . . . . .	318
7.29.2 Driver State Machine . . . . .	318
7.29.3 PWM Operations . . . . .	318
7.29.4 Macro Definition Documentation . . . . .	321
7.29.4.1 PWM_OUTPUT_MASK . . . . .	321
7.29.4.2 PWM_OUTPUT_DISABLED . . . . .	321
7.29.4.3 PWM_OUTPUT_ACTIVE_HIGH . . . . .	321
7.29.4.4 PWM_OUTPUT_ACTIVE_LOW . . . . .	321
7.29.4.5 PWM_FRACTION_TO_WIDTH . . . . .	322
7.29.4.6 PWM_DEGREES_TO_WIDTH . . . . .	322
7.29.4.7 PWM_PERCENTAGE_TO_WIDTH . . . . .	322
7.29.4.8 pwmChangePeriodl . . . . .	323
7.29.4.9 pwmEnableChannell . . . . .	323
7.29.4.10 pwmDisableChannell . . . . .	324
7.29.4.11 pwmIsChannelEnabledl . . . . .	325
7.29.4.12 pwmEnablePeriodicNotificationl . . . . .	325
7.29.4.13 pwmDisablePeriodicNotificationl . . . . .	325
7.29.4.14 pwmEnableChannelNotificationl . . . . .	326
7.29.4.15 pwmDisableChannelNotificationl . . . . .	326
7.29.4.16 PWM_CHANNELS . . . . .	327
7.29.4.17 PLATFORM_PWM_USE_PWM1 . . . . .	327

7.29.4.18 <code>pwm_lld_change_period</code> . . . . .	327
7.29.5 <b>Typedef Documentation</b> . . . . .	327
7.29.5.1 <code>PWMDriver</code> . . . . .	327
7.29.5.2 <code>pwmcallback_t</code> . . . . .	327
7.29.5.3 <code>pwmemode_t</code> . . . . .	328
7.29.5.4 <code>pwmchannel_t</code> . . . . .	328
7.29.5.5 <code>pwmchnmsk_t</code> . . . . .	328
7.29.5.6 <code>pwmcnt_t</code> . . . . .	328
7.29.6 <b>Enumeration Type Documentation</b> . . . . .	328
7.29.6.1 <code>pwmstate_t</code> . . . . .	328
7.29.7 <b>Function Documentation</b> . . . . .	328
7.29.7.1 <code>pwmlInit</code> . . . . .	328
7.29.7.2 <code>pwmObjectInit</code> . . . . .	329
7.29.7.3 <code>pwmStart</code> . . . . .	329
7.29.7.4 <code>pwmStop</code> . . . . .	330
7.29.7.5 <code>pwmChangePeriod</code> . . . . .	330
7.29.7.6 <code>pwmEnableChannel</code> . . . . .	331
7.29.7.7 <code>pwmDisableChannel</code> . . . . .	332
7.29.7.8 <code>pwmEnablePeriodicNotification</code> . . . . .	333
7.29.7.9 <code>pwmDisablePeriodicNotification</code> . . . . .	333
7.29.7.10 <code>pwmEnableChannelNotification</code> . . . . .	334
7.29.7.11 <code>pwmDisableChannelNotification</code> . . . . .	335
7.29.7.12 <code>pwm_lld_init</code> . . . . .	335
7.29.7.13 <code>pwm_lld_start</code> . . . . .	336
7.29.7.14 <code>pwm_lld_stop</code> . . . . .	336
7.29.7.15 <code>pwm_lld_enable_channel</code> . . . . .	336
7.29.7.16 <code>pwm_lld_disable_channel</code> . . . . .	337
7.29.7.17 <code>pwm_lld_enable_periodic_notification</code> . . . . .	337
7.29.7.18 <code>pwm_lld_disable_periodic_notification</code> . . . . .	338
7.29.7.19 <code>pwm_lld_enable_channel_notification</code> . . . . .	338
7.29.7.20 <code>pwm_lld_disable_channel_notification</code> . . . . .	338
7.29.8 <b>Variable Documentation</b> . . . . .	339
7.29.8.1 <code>PWMD1</code> . . . . .	339
7.30 <b>RTC Driver</b> . . . . .	340
7.30.1 <b>Detailed Description</b> . . . . .	340
7.30.2 <b>Macro Definition Documentation</b> . . . . .	342
7.30.2.1 <code>RTC_BASE_YEAR</code> . . . . .	342
7.30.2.2 <code>RTC_SUPPORTS_CALLBACKS</code> . . . . .	342
7.30.2.3 <code>RTC_ALARMS</code> . . . . .	342
7.30.2.4 <code>RTC_HAS_STORAGE</code> . . . . .	342

---

7.30.2.5	PLATFORM_RTC_USE_RTC1	342
7.30.2.6	_rtc_driver_methods	342
7.30.3	Typedef Documentation	343
7.30.3.1	RTCDriver	343
7.30.3.2	rtcalarm_t	343
7.30.3.3	rtccb_t	343
7.30.4	Enumeration Type Documentation	343
7.30.4.1	rtcevent_t	343
7.30.5	Function Documentation	343
7.30.5.1	rtcInit	343
7.30.5.2	rtcObjectInit	343
7.30.5.3	rtcSetTime	344
7.30.5.4	rtcGetTime	344
7.30.5.5	rtcSetAlarm	345
7.30.5.6	rtcGetAlarm	345
7.30.5.7	rtcSetCallback	346
7.30.5.8	rtcConvertDateTimeToStructTm	346
7.30.5.9	rtcConvertStructTmToDateTm	347
7.30.5.10	rtcConvertDateTimeToFAT	347
7.30.5.11	rtc_lld_init	347
7.30.5.12	rtc_lld_set_time	348
7.30.5.13	rtc_lld_get_time	348
7.30.5.14	rtc_lld_set_alarm	348
7.30.5.15	rtc_lld_get_alarm	349
7.30.6	Variable Documentation	349
7.30.6.1	RTCD1	349
7.31	SDC Driver	350
7.31.1	Detailed Description	350
7.31.2	Driver State Machine	350
7.31.3	Driver Operations	350
7.31.4	Macro Definition Documentation	353
7.31.4.1	SDC_INIT_RETRY	353
7.31.4.2	SDC_MMC_SUPPORT	354
7.31.4.3	SDC_NICE_WAITING	354
7.31.4.4	sdclsCardInserted	354
7.31.4.5	sdclsWriteProtected	354
7.31.4.6	PLATFORM_SDC_USE_SDC1	355
7.31.4.7	_sdc_driver_methods	355
7.31.5	Typedef Documentation	355
7.31.5.1	sdcmode_t	355

7.31.5.2 <code>sdcflags_t</code>	355
7.31.5.3 <code>SDCDriver</code>	355
7.31.6 Enumeration Type Documentation	355
7.31.6.1 <code>mmc_switch_t</code>	355
7.31.6.2 <code>sd_switch_t</code>	355
7.31.6.3 <code>sd_switch_function_t</code>	356
7.31.6.4 <code>sdcbusmode_t</code>	356
7.31.6.5 <code>sdcbusclk_t</code>	356
7.31.7 Function Documentation	356
7.31.7.1 <code>mode_detect</code>	356
7.31.7.2 <code>mmc_init</code>	356
7.31.7.3 <code>sdc_init</code>	357
7.31.7.4 <code>mmc_cmd6_construct</code>	358
7.31.7.5 <code>sdc_cmd6_construct</code>	358
7.31.7.6 <code>sdc_cmd6_extract_info</code>	359
7.31.7.7 <code>sdc_cmd6_check_status</code>	359
7.31.7.8 <code>sdc_detect_bus_clk</code>	359
7.31.7.9 <code>mmc_detect_bus_clk</code>	360
7.31.7.10 <code>detect_bus_clk</code>	361
7.31.7.11 <code>sdc_set_bus_width</code>	362
7.31.7.12 <code>mmc_set_bus_width</code>	363
7.31.7.13 <code>_sdc_wait_for_transfer_state</code>	364
7.31.7.14 <code>sdcInit</code>	365
7.31.7.15 <code>sdcObjectInit</code>	365
7.31.7.16 <code>sdcStart</code>	365
7.31.7.17 <code>sdcStop</code>	366
7.31.7.18 <code>sdcConnect</code>	367
7.31.7.19 <code>sdcDisconnect</code>	368
7.31.7.20 <code>sdcRead</code>	369
7.31.7.21 <code>sdcWrite</code>	370
7.31.7.22 <code>sdcGetAndClearErrors</code>	371
7.31.7.23 <code>sdcSync</code>	371
7.31.7.24 <code>sdcGetInfo</code>	372
7.31.7.25 <code>sdcErase</code>	372
7.31.7.26 <code>sdc_lld_init</code>	373
7.31.7.27 <code>sdc_lld_start</code>	373
7.31.7.28 <code>sdc_lld_stop</code>	373
7.31.7.29 <code>sdc_lld_start_clk</code>	374
7.31.7.30 <code>sdc_lld_set_data_clk</code>	374
7.31.7.31 <code>sdc_lld_stop_clk</code>	374

7.31.7.32 sdc_lld_set_bus_mode . . . . .	374
7.31.7.33 sdc_lld_send_cmd_none . . . . .	375
7.31.7.34 sdc_lld_send_cmd_short . . . . .	375
7.31.7.35 sdc_lld_send_cmd_short_crc . . . . .	375
7.31.7.36 sdc_lld_send_cmd_long_crc . . . . .	376
7.31.7.37 sdc_lld_read . . . . .	376
7.31.7.38 sdc_lld_write . . . . .	377
7.31.7.39 sdc_lld_sync . . . . .	377
7.31.8 Variable Documentation . . . . .	378
7.31.8.1 sdc_vmt . . . . .	378
7.31.8.2 SDCD1 . . . . .	378
7.32 Serial Driver . . . . .	379
7.32.1 Detailed Description . . . . .	379
7.32.2 Driver State Machine . . . . .	379
7.32.3 Macro Definition Documentation . . . . .	381
7.32.3.1 SD_PARITY_ERROR . . . . .	381
7.32.3.2 SD_FRAMING_ERROR . . . . .	382
7.32.3.3 SD_OVERRUN_ERROR . . . . .	382
7.32.3.4 SD_NOISE_ERROR . . . . .	382
7.32.3.5 SD_BREAK_DETECTED . . . . .	382
7.32.3.6 SERIAL_DEFAULT_BITRATE . . . . .	382
7.32.3.7 SERIAL_BUFFERS_SIZE . . . . .	382
7.32.3.8 _serial_driver_methods . . . . .	382
7.32.3.9 sdPut . . . . .	382
7.32.3.10 sdPutTimeout . . . . .	383
7.32.3.11 sdGet . . . . .	383
7.32.3.12 sdGetTimeout . . . . .	383
7.32.3.13 sdWrite . . . . .	384
7.32.3.14 sdWriteTimeout . . . . .	384
7.32.3.15 sdAsynchronousWrite . . . . .	384
7.32.3.16 sdRead . . . . .	385
7.32.3.17 sdReadTimeout . . . . .	385
7.32.3.18 sdAsynchronousRead . . . . .	385
7.32.3.19 PLATFORM_SERIAL_USE_USART1 . . . . .	385
7.32.3.20 _serial_driver_data . . . . .	386
7.32.4 Typedef Documentation . . . . .	386
7.32.4.1 SerialDriver . . . . .	386
7.32.5 Enumeration Type Documentation . . . . .	386
7.32.5.1 sdstate_t . . . . .	386
7.32.6 Function Documentation . . . . .	386

7.32.6.1	<a href="#">sdInit</a>	386
7.32.6.2	<a href="#">sdObjectInit</a>	387
7.32.6.3	<a href="#">sdStart</a>	387
7.32.6.4	<a href="#">sdStop</a>	388
7.32.6.5	<a href="#">sdIncomingData</a>	389
7.32.6.6	<a href="#">sdRequestData</a>	389
7.32.6.7	<a href="#">sdPutWouldBlock</a>	390
7.32.6.8	<a href="#">sdGetWouldBlock</a>	391
7.32.6.9	<a href="#">sd_lld_init</a>	392
7.32.6.10	<a href="#">sd_lld_start</a>	392
7.32.6.11	<a href="#">sd_lld_stop</a>	392
7.32.7	<a href="#">Variable Documentation</a>	392
7.32.7.1	<a href="#">SD1</a>	392
7.32.7.2	<a href="#">default_config</a>	393
7.33	<a href="#">Serial over USB Driver</a>	394
7.33.1	<a href="#">Detailed Description</a>	394
7.33.2	<a href="#">Driver State Machine</a>	394
7.33.3	<a href="#">Macro Definition Documentation</a>	396
7.33.3.1	<a href="#">SERIAL_USB_BUFFERS_SIZE</a>	396
7.33.3.2	<a href="#">SERIAL_USB_BUFFERS_NUMBER</a>	396
7.33.3.3	<a href="#">_serial_usb_driver_data</a>	396
7.33.3.4	<a href="#">_serial_usb_driver_methods</a>	396
7.33.4	<a href="#">Typedef Documentation</a>	396
7.33.4.1	<a href="#">SerialUSBDriver</a>	396
7.33.5	<a href="#">Enumeration Type Documentation</a>	396
7.33.5.1	<a href="#">sdustate_t</a>	396
7.33.6	<a href="#">Function Documentation</a>	397
7.33.6.1	<a href="#">ibnotify</a>	397
7.33.6.2	<a href="#">obnotify</a>	397
7.33.6.3	<a href="#">sdulinit</a>	397
7.33.6.4	<a href="#">sduObjectInit</a>	398
7.33.6.5	<a href="#">sduStart</a>	398
7.33.6.6	<a href="#">sduStop</a>	399
7.33.6.7	<a href="#">sduDisconnectI</a>	400
7.33.6.8	<a href="#">sduConfigureHookI</a>	400
7.33.6.9	<a href="#">sduRequestsHook</a>	401
7.33.6.10	<a href="#">sduSOFHookI</a>	401
7.33.6.11	<a href="#">sduDataTransmitted</a>	402
7.33.6.12	<a href="#">sduDataReceived</a>	402
7.33.6.13	<a href="#">sdulInterruptTransmitted</a>	404

7.34 SPI Driver . . . . .	405
7.34.1 Detailed Description . . . . .	405
7.34.2 Driver State Machine . . . . .	405
7.34.3 Macro Definition Documentation . . . . .	408
7.34.3.1 SPI_USE_WAIT . . . . .	408
7.34.3.2 SPI_USE_MUTUAL_EXCLUSION . . . . .	408
7.34.3.3 spiSelectl . . . . .	408
7.34.3.4 spiUnselectl . . . . .	408
7.34.3.5 spiStartIgnorel . . . . .	409
7.34.3.6 spiStartExchangel . . . . .	409
7.34.3.7 spiStartSendl . . . . .	410
7.34.3.8 spiStartReceive l . . . . .	411
7.34.3.9 spiPolledExchange . . . . .	411
7.34.3.10 _spi_wakeup_isr . . . . .	412
7.34.3.11 _spi_isr_code . . . . .	412
7.34.3.12 PLATFORM_SPI_USE_SPI1 . . . . .	413
7.34.4 Typedef Documentation . . . . .	413
7.34.4.1 SPIDriver . . . . .	413
7.34.4.2 spicallback_t . . . . .	413
7.34.5 Enumeration Type Documentation . . . . .	413
7.34.5.1 spistate_t . . . . .	413
7.34.6 Function Documentation . . . . .	413
7.34.6.1 spilinit . . . . .	413
7.34.6.2 spiObjectInit . . . . .	414
7.34.6.3 spiStart . . . . .	414
7.34.6.4 spiStop . . . . .	415
7.34.6.5 spiSelect . . . . .	416
7.34.6.6 spiUnselect . . . . .	417
7.34.6.7 spiStartIgnore . . . . .	417
7.34.6.8 spiStartExchange . . . . .	418
7.34.6.9 spiStartSend . . . . .	419
7.34.6.10 spiStartReceive . . . . .	420
7.34.6.11 spilgnore . . . . .	421
7.34.6.12 spiExchange . . . . .	421
7.34.6.13 spiSend . . . . .	422
7.34.6.14 spiReceive . . . . .	423
7.34.6.15 spiAcquireBus . . . . .	424
7.34.6.16 spiReleaseBus . . . . .	424
7.34.6.17 spi_lld_init . . . . .	425
7.34.6.18 spi_lld_start . . . . .	425

7.34.6.19 spi_lld_stop . . . . .	425
7.34.6.20 spi_lld_select . . . . .	426
7.34.6.21 spi_lld_unselect . . . . .	426
7.34.6.22 spi_lld_ignore . . . . .	426
7.34.6.23 spi_lld_exchange . . . . .	426
7.34.6.24 spi_lld_send . . . . .	427
7.34.6.25 spi_lld_receive . . . . .	427
7.34.6.26 spi_lld_polled_exchange . . . . .	428
7.34.7 Variable Documentation . . . . .	428
7.34.7.1 SPID1 . . . . .	428
7.35 ST Driver . . . . .	429
7.35.1 Detailed Description . . . . .	429
7.35.2 Macro Definition Documentation . . . . .	429
7.35.2.1 stGetCounter . . . . .	429
7.35.2.2 stIsAlarmActive . . . . .	430
7.35.3 Function Documentation . . . . .	430
7.35.3.1 stInit . . . . .	430
7.35.3.2 stStartAlarm . . . . .	431
7.35.3.3 stStopAlarm . . . . .	431
7.35.3.4 stSetAlarm . . . . .	431
7.35.3.5 stGetAlarm . . . . .	432
7.35.3.6 st_lld_init . . . . .	432
7.35.3.7 st_lld_get_counter . . . . .	433
7.35.3.8 st_lld_start_alarm . . . . .	433
7.35.3.9 st_lld_stop_alarm . . . . .	433
7.35.3.10 st_lld_set_alarm . . . . .	433
7.35.3.11 st_lld_get_alarm . . . . .	433
7.35.3.12 st_lld_is_alarm_active . . . . .	434
7.36 UART Driver . . . . .	435
7.36.1 Detailed Description . . . . .	435
7.36.2 Driver State Machine . . . . .	435
7.36.2.1 Transmitter sub State Machine . . . . .	436
7.36.2.2 Receiver sub State Machine . . . . .	436
7.36.3 Macro Definition Documentation . . . . .	439
7.36.3.1 UART_NO_ERROR . . . . .	439
7.36.3.2 UART_PARITY_ERROR . . . . .	439
7.36.3.3 UART_FRAMING_ERROR . . . . .	439
7.36.3.4 UART_OVERRUN_ERROR . . . . .	440
7.36.3.5 UART_NOISE_ERROR . . . . .	440
7.36.3.6 UART_BREAK_DETECTED . . . . .	440

7.36.3.7	UART_USE_WAIT . . . . .	440
7.36.3.8	UART_USE_MUTUAL_EXCLUSION . . . . .	440
7.36.3.9	_uart_wakeup_tx1_isr . . . . .	440
7.36.3.10	_uart_wakeup_tx2_isr . . . . .	441
7.36.3.11	_uart_wakeup_rx_complete_isr . . . . .	441
7.36.3.12	_uart_wakeup_rx_error_isr . . . . .	441
7.36.3.13	_uart_tx1_isr_code . . . . .	442
7.36.3.14	_uart_tx2_isr_code . . . . .	442
7.36.3.15	_uart_rx_complete_isr_code . . . . .	443
7.36.3.16	_uart_rx_error_isr_code . . . . .	443
7.36.3.17	_uart_rx_idle_code . . . . .	444
7.36.3.18	PLATFORM_UART_USE_UART1 . . . . .	444
7.36.4	Typedef Documentation . . . . .	445
7.36.4.1	uartflags_t . . . . .	445
7.36.4.2	UARTDriver . . . . .	445
7.36.4.3	uartcb_t . . . . .	445
7.36.4.4	uartccb_t . . . . .	445
7.36.4.5	uartecb_t . . . . .	445
7.36.5	Enumeration Type Documentation . . . . .	445
7.36.5.1	uartstate_t . . . . .	445
7.36.5.2	uarttxstate_t . . . . .	445
7.36.5.3	uartrxstate_t . . . . .	446
7.36.6	Function Documentation . . . . .	446
7.36.6.1	uartInit . . . . .	446
7.36.6.2	uartObjectInit . . . . .	446
7.36.6.3	uartStart . . . . .	447
7.36.6.4	uartStop . . . . .	447
7.36.6.5	uartStartSend . . . . .	448
7.36.6.6	uartStartSendl . . . . .	449
7.36.6.7	uartStopSend . . . . .	449
7.36.6.8	uartStopSendl . . . . .	450
7.36.6.9	uartStartReceive . . . . .	451
7.36.6.10	uartStartReceive1 . . . . .	451
7.36.6.11	uartStopReceive . . . . .	452
7.36.6.12	uartStopReceive1 . . . . .	453
7.36.6.13	uartSendTimeout . . . . .	453
7.36.6.14	uartSendFullTimeout . . . . .	454
7.36.6.15	uartReceiveTimeout . . . . .	455
7.36.6.16	uartAcquireBus . . . . .	456
7.36.6.17	uartReleaseBus . . . . .	457

7.36.6.18 uart_lld_init . . . . .	457
7.36.6.19 uart_lld_start . . . . .	458
7.36.6.20 uart_lld_stop . . . . .	458
7.36.6.21 uart_lld_start_send . . . . .	458
7.36.6.22 uart_lld_stop_send . . . . .	459
7.36.6.23 uart_lld_start_receive . . . . .	459
7.36.6.24 uart_lld_stop_receive . . . . .	459
7.36.7 Variable Documentation . . . . .	460
7.36.7.1 UARTD1 . . . . .	460
7.37 USB Driver . . . . .	461
7.37.1 Detailed Description . . . . .	461
7.37.2 Driver State Machine . . . . .	461
7.37.3 USB Operations . . . . .	461
7.37.3.1 USB Implementation . . . . .	461
7.37.3.2 USB Endpoints . . . . .	462
7.37.3.3 USB Callbacks . . . . .	463
7.37.4 Macro Definition Documentation . . . . .	468
7.37.4.1 USB_DESC_INDEX . . . . .	468
7.37.4.2 USB_DESC_BYTE . . . . .	468
7.37.4.3 USB_DESC_WORD . . . . .	468
7.37.4.4 USB_DESC_BCD . . . . .	468
7.37.4.5 USB_DESC_DEVICE . . . . .	468
7.37.4.6 USB_DESC_CONFIGURATION_SIZE . . . . .	469
7.37.4.7 USB_DESC_CONFIGURATION . . . . .	469
7.37.4.8 USB_DESC_INTERFACE_SIZE . . . . .	469
7.37.4.9 USB_DESC_INTERFACE . . . . .	469
7.37.4.10 USB_DESC_INTERFACE_ASSOCIATION_SIZE . . . . .	470
7.37.4.11 USB_DESC_INTERFACE_ASSOCIATION . . . . .	470
7.37.4.12 USB_DESC_ENDPOINT_SIZE . . . . .	470
7.37.4.13 USB_DESC_ENDPOINT . . . . .	470
7.37.4.14 USB_EP_MODE_TYPE . . . . .	470
7.37.4.15 USB_EP_MODE_TYPE_CTRL . . . . .	471
7.37.4.16 USB_EP_MODE_TYPE_ISOC . . . . .	471
7.37.4.17 USB_EP_MODE_TYPE_BULK . . . . .	471
7.37.4.18 USB_EP_MODE_TYPE_INTR . . . . .	471
7.37.4.19 USB_USE_WAIT . . . . .	471
7.37.4.20 usbGetDriverStatel . . . . .	471
7.37.4.21 usbConnectBus . . . . .	471
7.37.4.22 usbDisconnectBus . . . . .	471
7.37.4.23 usbGetFrameNumberX . . . . .	472

7.37.4.24 <code>usbGetTransmitStatus()</code>	472
7.37.4.25 <code>usbGetReceiveStatus()</code>	472
7.37.4.26 <code>usbGetReceiveTransactionSizeX()</code>	473
7.37.4.27 <code>usbSetupTransfer()</code>	473
7.37.4.28 <code>usbReadSetup()</code>	474
7.37.4.29 <code>_usb_isr_invoke_event_cb()</code>	474
7.37.4.30 <code>_usb_isr_invoke_sof_cb()</code>	474
7.37.4.31 <code>_usb_isr_invoke_setup_cb()</code>	476
7.37.4.32 <code>_usb_isr_invoke_in_cb()</code>	476
7.37.4.33 <code>_usb_isr_invoke_out_cb()</code>	476
7.37.4.34 <code>USB_MAX_ENDPOINTS</code>	477
7.37.4.35 <code>USB_EP0_STATUS_STAGE</code>	477
7.37.4.36 <code>USB_SET_ADDRESS_MODE</code>	477
7.37.4.37 <code>USB_SET_ADDRESS_ACK_HANDLING</code>	477
7.37.4.38 <code>PLATFORM_USB_USE_USB1</code>	477
7.37.4.39 <code>usb_lld_get_frame_number()</code>	477
7.37.4.40 <code>usb_lld_get_transaction_size()</code>	478
7.37.4.41 <code>usb_lld_connect_bus()</code>	478
7.37.4.42 <code>usb_lld_disconnect_bus()</code>	478
7.37.5 <b>Typedef Documentation</b>	478
7.37.5.1 <code>USBDriver</code>	478
7.37.5.2 <code>usbep_t</code>	479
7.37.5.3 <code>usbcallback_t</code>	479
7.37.5.4 <code>usbepcallback_t</code>	479
7.37.5.5 <code>usbeventcb_t</code>	479
7.37.5.6 <code>usbreqhandler_t</code>	479
7.37.5.7 <code>usbgetdescriptor_t</code>	479
7.37.6 <b>Enumeration Type Documentation</b>	480
7.37.6.1 <code>usbstate_t</code>	480
7.37.6.2 <code>usbepstatus_t</code>	480
7.37.6.3 <code>usbep0state_t</code>	480
7.37.6.4 <code>usbevent_t</code>	480
7.37.7 <b>Function Documentation</b>	481
7.37.7.1 <code>set_address()</code>	481
7.37.7.2 <code>default_handler()</code>	482
7.37.7.3 <code>usbInit()</code>	483
7.37.7.4 <code>usbObjectInit()</code>	483
7.37.7.5 <code>usbStart()</code>	484
7.37.7.6 <code>usbStop()</code>	484
7.37.7.7 <code>usbInitEndpoint()</code>	485

7.37.7.8	usbDisableEndpoints	485
7.37.7.9	usbStartReceive	486
7.37.7.10	usbStartTransmit	487
7.37.7.11	usbReceive	487
7.37.7.12	usbTransmit	488
7.37.7.13	usbStallReceive	489
7.37.7.14	usbStallTransmit	490
7.37.7.15	_usb_reset	490
7.37.7.16	_usb_suspend	491
7.37.7.17	_usb_wakeup	492
7.37.7.18	_usb_ep0setup	492
7.37.7.19	_usb_ep0in	493
7.37.7.20	_usb_ep0out	494
7.37.7.21	usb_lld_init	495
7.37.7.22	usb_lld_start	495
7.37.7.23	usb_lld_stop	496
7.37.7.24	usb_lld_reset	497
7.37.7.25	usb_lld_set_address	497
7.37.7.26	usb_lld_init_endpoint	497
7.37.7.27	usb_lld_disable_endpoints	498
7.37.7.28	usb_lld_get_status_out	499
7.37.7.29	usb_lld_get_status_in	499
7.37.7.30	usb_lld_read_setup	499
7.37.7.31	usb_lld_prepare_receive	500
7.37.7.32	usb_lld_prepare_transmit	500
7.37.7.33	usb_lld_start_out	500
7.37.7.34	usb_lld_start_in	500
7.37.7.35	usb_lld_stall_out	501
7.37.7.36	usb_lld_stall_in	501
7.37.7.37	usb_lld_clear_out	501
7.37.7.38	usb_lld_clear_in	501
7.37.8	Variable Documentation	502
7.37.8.1	USBD1	502
7.37.8.2	ep0_state	502
7.37.8.3	in	502
7.37.8.4	out	502
7.37.8.5	ep0config	502
7.38	USB CDC Header	503
7.38.1	Detailed Description	503
7.39	WDG Driver	505

---

7.39.1	Detailed Description	505
7.39.2	Macro Definition Documentation	506
7.39.2.1	wdgReset!	506
7.39.2.2	PLATFORM_WDG_USE_WDG1	506
7.39.3	Typedef Documentation	506
7.39.3.1	WDGDriver	506
7.39.4	Enumeration Type Documentation	506
7.39.4.1	wdgstate_t	506
7.39.5	Function Documentation	507
7.39.5.1	wdgInit	507
7.39.5.2	wdgStart	507
7.39.5.3	wdgStop	508
7.39.5.4	wdgReset	509
7.39.5.5	wdg_lld_init	509
7.39.5.6	wdg_lld_start	510
7.39.5.7	wdg_lld_stop	510
7.39.5.8	wdg_lld_reset	510
<b>8</b>	<b>Data Structure Documentation</b>	<b>511</b>
8.1	ADCCConfig Struct Reference	511
8.1.1	Detailed Description	511
8.2	ADCConversionGroup Struct Reference	512
8.2.1	Detailed Description	512
8.2.2	Field Documentation	513
8.2.2.1	circular	513
8.2.2.2	num_channels	513
8.2.2.3	end_cb	513
8.2.2.4	error_cb	513
8.3	ADCDriver Struct Reference	513
8.3.1	Detailed Description	514
8.3.2	Field Documentation	515
8.3.2.1	state	515
8.3.2.2	config	515
8.3.2.3	samples	515
8.3.2.4	depth	515
8.3.2.5	grpp	515
8.3.2.6	thread	515
8.3.2.7	mutex	515
8.4	BaseAsynchronousChannel Struct Reference	515
8.4.1	Detailed Description	517

8.4.2	Field Documentation	517
8.4.2.1	vmt	517
8.5	BaseAsynchronousChannelVMT Struct Reference	517
8.5.1	Detailed Description	519
8.6	BaseBlockDevice Struct Reference	519
8.6.1	Detailed Description	521
8.6.2	Field Documentation	521
8.6.2.1	vmt	521
8.7	BaseBlockDeviceVMT Struct Reference	521
8.7.1	Detailed Description	522
8.8	BaseChannel Struct Reference	522
8.8.1	Detailed Description	524
8.8.2	Field Documentation	524
8.8.2.1	vmt	524
8.9	BaseChannelVMT Struct Reference	524
8.9.1	Detailed Description	526
8.10	BaseSequentialStream Struct Reference	526
8.10.1	Detailed Description	527
8.10.2	Field Documentation	527
8.10.2.1	vmt	527
8.11	BaseSequentialStreamVMT Struct Reference	527
8.11.1	Detailed Description	528
8.12	BlockDeviceInfo Struct Reference	528
8.12.1	Detailed Description	529
8.12.2	Field Documentation	529
8.12.2.1	blk_size	529
8.12.2.2	blk_num	529
8.13	CANConfig Struct Reference	529
8.13.1	Detailed Description	530
8.14	CANDriver Struct Reference	530
8.14.1	Detailed Description	531
8.14.2	Field Documentation	531
8.14.2.1	state	531
8.14.2.2	config	531
8.14.2.3	txqueue	531
8.14.2.4	rxqueue	531
8.14.2.5	rxfull_event	531
8.14.2.6	txempty_event	532
8.14.2.7	error_event	532
8.14.2.8	sleep_event	532

8.14.2.9	wakeup_event	532
8.15	CANRxFrame Struct Reference	532
8.15.1	Detailed Description	533
8.15.2	Field Documentation	533
8.15.2.1	FMI	533
8.15.2.2	TIME	533
8.15.2.3	DLC	533
8.15.2.4	RTR	533
8.15.2.5	IDE	534
8.15.2.6	SID	534
8.15.2.7	EID	534
8.15.2.8	data8	534
8.15.2.9	data16	534
8.15.2.10	data32	534
8.16	CANTxFrame Struct Reference	534
8.16.1	Detailed Description	535
8.16.2	Field Documentation	535
8.16.2.1	DLC	535
8.16.2.2	RTR	535
8.16.2.3	IDE	535
8.16.2.4	SID	535
8.16.2.5	EID	535
8.16.2.6	data8	536
8.16.2.7	data16	536
8.16.2.8	data32	536
8.17	cdc_linecoding_t Struct Reference	536
8.17.1	Detailed Description	536
8.18	DACConfig Struct Reference	536
8.18.1	Detailed Description	537
8.19	DACConversionGroup Struct Reference	537
8.19.1	Detailed Description	538
8.19.2	Field Documentation	538
8.19.2.1	num_channels	538
8.19.2.2	end_cb	538
8.19.2.3	error_cb	538
8.20	DACDriver Struct Reference	538
8.20.1	Detailed Description	539
8.20.2	Field Documentation	540
8.20.2.1	state	540
8.20.2.2	grpp	540

8.20.2.3	samples	540
8.20.2.4	depth	540
8.20.2.5	config	540
8.20.2.6	thread	540
8.20.2.7	mutex	540
8.21	event_source Struct Reference	540
8.21.1	Detailed Description	541
8.21.2	Field Documentation	541
8.21.2.1	flags	541
8.21.2.2	cb	541
8.21.2.3	param	541
8.22	EXTChannelConfig Struct Reference	541
8.22.1	Detailed Description	542
8.22.2	Field Documentation	542
8.22.2.1	mode	542
8.22.2.2	cb	542
8.23	EXTConfig Struct Reference	543
8.23.1	Detailed Description	543
8.23.2	Field Documentation	543
8.23.2.1	channels	543
8.24	EXTDriver Struct Reference	544
8.24.1	Detailed Description	544
8.24.2	Field Documentation	544
8.24.2.1	state	544
8.24.2.2	config	545
8.25	FileStream Struct Reference	545
8.25.1	Detailed Description	546
8.25.2	Field Documentation	546
8.25.2.1	vmt	546
8.26	FileStreamVMT Struct Reference	546
8.26.1	Detailed Description	547
8.27	GPTConfig Struct Reference	547
8.27.1	Detailed Description	548
8.27.2	Field Documentation	548
8.27.2.1	frequency	548
8.27.2.2	callback	548
8.28	GPTDriver Struct Reference	549
8.28.1	Detailed Description	549
8.28.2	Field Documentation	549
8.28.2.1	state	549

8.28.2.2	config	549
8.29	I2CConfig Struct Reference	549
8.29.1	Detailed Description	550
8.30	I2CDriver Struct Reference	550
8.30.1	Detailed Description	551
8.30.2	Field Documentation	551
8.30.2.1	state	551
8.30.2.2	config	551
8.30.2.3	errors	551
8.31	I2SConfig Struct Reference	551
8.31.1	Detailed Description	552
8.31.2	Field Documentation	552
8.31.2.1	tx_buffer	552
8.31.2.2	rx_buffer	553
8.31.2.3	size	553
8.31.2.4	end_cb	553
8.32	I2SDriver Struct Reference	553
8.32.1	Detailed Description	554
8.32.2	Field Documentation	554
8.32.2.1	state	554
8.32.2.2	config	554
8.33	ICUConfig Struct Reference	554
8.33.1	Detailed Description	555
8.33.2	Field Documentation	555
8.33.2.1	mode	555
8.33.2.2	frequency	555
8.33.2.3	width_cb	555
8.33.2.4	period_cb	555
8.33.2.5	overflow_cb	555
8.34	ICUDriver Struct Reference	555
8.34.1	Detailed Description	556
8.34.2	Field Documentation	556
8.34.2.1	state	556
8.34.2.2	config	556
8.35	io_buffers_queue Struct Reference	556
8.35.1	Detailed Description	558
8.35.2	Field Documentation	558
8.35.2.1	waiting	558
8.35.2.2	bcounter	558
8.35.2.3	bwrptr	558

8.35.2.4	brdptr	558
8.35.2.5	btop	558
8.35.2.6	bsize	558
8.35.2.7	bn	558
8.35.2.8	buffers	558
8.35.2.9	ptr	558
8.35.2.10	top	559
8.35.2.11	notify	559
8.35.2.12	link	559
8.36	io_queue Struct Reference	559
8.36.1	Detailed Description	560
8.36.2	Field Documentation	560
8.36.2.1	q_waiting	560
8.36.2.2	q_counter	560
8.36.2.3	q_buffer	560
8.36.2.4	q_top	560
8.36.2.5	q_wptr	560
8.36.2.6	q_rdptr	560
8.36.2.7	q_notify	560
8.36.2.8	q_link	560
8.37	IOBus Struct Reference	561
8.37.1	Detailed Description	561
8.37.2	Field Documentation	561
8.37.2.1	portid	561
8.37.2.2	mask	561
8.37.2.3	offset	562
8.38	MACConfig Struct Reference	562
8.38.1	Detailed Description	562
8.38.2	Field Documentation	562
8.38.2.1	mac_address	562
8.39	MACDriver Struct Reference	562
8.39.1	Detailed Description	563
8.39.2	Field Documentation	563
8.39.2.1	state	563
8.39.2.2	config	563
8.39.2.3	tdqueue	563
8.39.2.4	rdqueue	564
8.39.2.5	rdevent	564
8.40	MACReceiveDescriptor Struct Reference	564
8.40.1	Detailed Description	564

---

8.40.2 Field Documentation . . . . .	564
8.40.2.1 offset . . . . .	564
8.40.2.2 size . . . . .	564
8.41 MACTransmitDescriptor Struct Reference . . . . .	565
8.41.1 Detailed Description . . . . .	565
8.41.2 Field Documentation . . . . .	565
8.41.2.1 offset . . . . .	565
8.41.2.2 size . . . . .	565
8.42 MMConfig Struct Reference . . . . .	565
8.42.1 Detailed Description . . . . .	566
8.42.2 Field Documentation . . . . .	566
8.42.2.1 spip . . . . .	566
8.42.2.2 lscfg . . . . .	567
8.42.2.3 hscfg . . . . .	567
8.43 MMCDriver Struct Reference . . . . .	567
8.43.1 Detailed Description . . . . .	568
8.43.2 Field Documentation . . . . .	568
8.43.2.1 vmt . . . . .	568
8.43.2.2 config . . . . .	569
8.44 MMCDriverVMT Struct Reference . . . . .	569
8.44.1 Detailed Description . . . . .	570
8.45 MMCSDBlockDevice Struct Reference . . . . .	570
8.45.1 Detailed Description . . . . .	572
8.45.2 Field Documentation . . . . .	572
8.45.2.1 vmt . . . . .	572
8.46 MMCSDBlockDeviceVMT Struct Reference . . . . .	572
8.46.1 Detailed Description . . . . .	573
8.47 PALConfig Struct Reference . . . . .	574
8.47.1 Detailed Description . . . . .	574
8.48 PWMChannelConfig Struct Reference . . . . .	574
8.48.1 Detailed Description . . . . .	575
8.48.2 Field Documentation . . . . .	575
8.48.2.1 mode . . . . .	575
8.48.2.2 callback . . . . .	575
8.49 PWMConfig Struct Reference . . . . .	576
8.49.1 Detailed Description . . . . .	577
8.49.2 Field Documentation . . . . .	577
8.49.2.1 frequency . . . . .	577
8.49.2.2 period . . . . .	577
8.49.2.3 callback . . . . .	577

8.49.2.4	channels	577
8.50	PWMDriver Struct Reference	577
8.50.1	Detailed Description	578
8.50.2	Field Documentation	578
8.50.2.1	state	579
8.50.2.2	config	579
8.50.2.3	period	579
8.50.2.4	enabled	579
8.50.2.5	channels	579
8.51	RTCAlarm Struct Reference	579
8.51.1	Detailed Description	579
8.52	RTCDateTime Struct Reference	579
8.52.1	Detailed Description	580
8.52.2	Field Documentation	580
8.52.2.1	year	580
8.52.2.2	month	580
8.52.2.3	dstflag	580
8.52.2.4	dayofweek	581
8.52.2.5	day	581
8.52.2.6	millisecond	581
8.53	RTCDriver Struct Reference	581
8.53.1	Detailed Description	582
8.53.2	Field Documentation	582
8.53.2.1	vmt	582
8.54	RTCDriverVMT Struct Reference	582
8.54.1	Detailed Description	583
8.55	SDCConfig Struct Reference	583
8.55.1	Detailed Description	584
8.55.2	Field Documentation	584
8.55.2.1	scratchpad	584
8.55.2.2	bus_width	584
8.56	SDCDriver Struct Reference	584
8.56.1	Detailed Description	585
8.56.2	Field Documentation	585
8.56.2.1	vmt	585
8.56.2.2	config	586
8.56.2.3	cardmode	586
8.56.2.4	errors	586
8.56.2.5	rca	586
8.57	SDCDriverVMT Struct Reference	586

8.57.1 Detailed Description . . . . .	587
8.58 SerialConfig Struct Reference . . . . .	587
8.58.1 Detailed Description . . . . .	588
8.58.2 Field Documentation . . . . .	588
8.58.2.1 speed . . . . .	588
8.59 SerialDriver Struct Reference . . . . .	588
8.59.1 Detailed Description . . . . .	589
8.59.2 Field Documentation . . . . .	589
8.59.2.1 vmt . . . . .	589
8.60 SerialDriverVMT Struct Reference . . . . .	589
8.60.1 Detailed Description . . . . .	591
8.61 SerialUSBConfig Struct Reference . . . . .	591
8.61.1 Detailed Description . . . . .	591
8.61.2 Field Documentation . . . . .	592
8.61.2.1 usbp . . . . .	592
8.61.2.2 bulk_in . . . . .	592
8.61.2.3 bulk_out . . . . .	592
8.61.2.4 int_in . . . . .	592
8.62 SerialUSBDriver Struct Reference . . . . .	592
8.62.1 Detailed Description . . . . .	594
8.62.2 Field Documentation . . . . .	594
8.62.2.1 vmt . . . . .	594
8.63 SerialUSBDriverVMT Struct Reference . . . . .	594
8.63.1 Detailed Description . . . . .	596
8.64 SPIConfig Struct Reference . . . . .	596
8.64.1 Detailed Description . . . . .	596
8.64.2 Field Documentation . . . . .	596
8.64.2.1 end_cb . . . . .	596
8.65 SPIDriver Struct Reference . . . . .	597
8.65.1 Detailed Description . . . . .	597
8.65.2 Field Documentation . . . . .	597
8.65.2.1 state . . . . .	597
8.65.2.2 config . . . . .	598
8.65.2.3 thread . . . . .	598
8.65.2.4 mutex . . . . .	598
8.66 threads_queue_t Struct Reference . . . . .	598
8.66.1 Detailed Description . . . . .	598
8.67 UARTConfig Struct Reference . . . . .	598
8.67.1 Detailed Description . . . . .	599
8.67.2 Field Documentation . . . . .	599

8.67.2.1	txend1_cb	600
8.67.2.2	txend2_cb	600
8.67.2.3	rxend_cb	600
8.67.2.4	rxchar_cb	600
8.67.2.5	rxerr_cb	600
8.68	UARTDriver Struct Reference	600
8.68.1	Detailed Description	601
8.68.2	Field Documentation	602
8.68.2.1	state	602
8.68.2.2	txstate	602
8.68.2.3	rxstate	602
8.68.2.4	config	602
8.68.2.5	early	602
8.68.2.6	threadrx	602
8.68.2.7	threadtx	602
8.68.2.8	mutex	602
8.69	unpacked_mmc_cid_t Struct Reference	602
8.69.1	Detailed Description	603
8.70	unpacked_mmc_csd_t Struct Reference	603
8.70.1	Detailed Description	603
8.71	unpacked_sdc_cid_t Struct Reference	604
8.71.1	Detailed Description	604
8.72	unpacked_sdc_csd_10_t Struct Reference	604
8.72.1	Detailed Description	605
8.73	unpacked_sdc_csd_20_t Struct Reference	605
8.73.1	Detailed Description	606
8.74	USBConfig Struct Reference	606
8.74.1	Detailed Description	608
8.74.2	Field Documentation	608
8.74.2.1	event_cb	608
8.74.2.2	get_descriptor_cb	608
8.74.2.3	requests_hook_cb	608
8.74.2.4	sof_cb	608
8.75	USBDescriptor Struct Reference	608
8.75.1	Detailed Description	609
8.75.2	Field Documentation	609
8.75.2.1	ud_size	609
8.75.2.2	ud_string	609
8.76	USBDriver Struct Reference	609
8.76.1	Detailed Description	610

---

8.76.2 Field Documentation . . . . .	610
8.76.2.1 state . . . . .	610
8.76.2.2 config . . . . .	610
8.76.2.3 transmitting . . . . .	610
8.76.2.4 receiving . . . . .	610
8.76.2.5 epc . . . . .	611
8.76.2.6 in_params . . . . .	611
8.76.2.7 out_params . . . . .	611
8.76.2.8 ep0state . . . . .	611
8.76.2.9 ep0next . . . . .	611
8.76.2.10 ep0n . . . . .	611
8.76.2.11 ep0endcb . . . . .	611
8.76.2.12 setup . . . . .	611
8.76.2.13 status . . . . .	611
8.76.2.14 address . . . . .	611
8.76.2.15 configuration . . . . .	611
8.77 USBEndpointConfig Struct Reference . . . . .	612
8.77.1 Detailed Description . . . . .	613
8.77.2 Field Documentation . . . . .	613
8.77.2.1 ep_mode . . . . .	613
8.77.2.2 setup_cb . . . . .	613
8.77.2.3 in_cb . . . . .	613
8.77.2.4 out_cb . . . . .	614
8.77.2.5 in_maxsize . . . . .	614
8.77.2.6 out_maxsize . . . . .	614
8.77.2.7 in_state . . . . .	614
8.77.2.8 out_state . . . . .	614
8.78 USBInEndpointState Struct Reference . . . . .	614
8.78.1 Detailed Description . . . . .	615
8.78.2 Field Documentation . . . . .	615
8.78.2.1 txsize . . . . .	615
8.78.2.2 txcnt . . . . .	615
8.78.2.3 txbuf . . . . .	615
8.78.2.4 thread . . . . .	615
8.79 USBOutEndpointState Struct Reference . . . . .	615
8.79.1 Detailed Description . . . . .	616
8.79.2 Field Documentation . . . . .	616
8.79.2.1 rxsize . . . . .	616
8.79.2.2 rxcnt . . . . .	616
8.79.2.3 rxbuf . . . . .	616

8.79.2.4	thread	616
8.80	WDGConfig Struct Reference	616
8.80.1	Detailed Description	617
8.81	WDGDriver Struct Reference	617
8.81.1	Detailed Description	617
8.81.2	Field Documentation	617
8.81.2.1	state	617
8.81.2.2	config	618
<b>9</b>	<b>File Documentation</b>	<b>619</b>
9.1	adc.c File Reference	619
9.1.1	Detailed Description	619
9.2	adc.h File Reference	620
9.2.1	Detailed Description	621
9.3	adc_Ild.c File Reference	621
9.3.1	Detailed Description	621
9.4	adc_Ild.h File Reference	621
9.4.1	Detailed Description	622
9.5	can.c File Reference	623
9.5.1	Detailed Description	623
9.6	can.h File Reference	623
9.6.1	Detailed Description	624
9.7	can_Ild.c File Reference	624
9.7.1	Detailed Description	625
9.8	can_Ild.h File Reference	625
9.8.1	Detailed Description	626
9.9	dac.c File Reference	626
9.9.1	Detailed Description	627
9.10	dac.h File Reference	627
9.10.1	Detailed Description	628
9.11	dac_Ild.c File Reference	629
9.11.1	Detailed Description	629
9.12	dac_Ild.h File Reference	629
9.12.1	Detailed Description	630
9.13	ext.c File Reference	631
9.13.1	Detailed Description	631
9.14	ext.h File Reference	631
9.14.1	Detailed Description	632
9.15	ext_Ild.c File Reference	632
9.15.1	Detailed Description	633

9.16 ext_lld.h File Reference . . . . .	633
9.16.1 Detailed Description . . . . .	634
9.17 gpt.c File Reference . . . . .	634
9.17.1 Detailed Description . . . . .	635
9.18 gpt.h File Reference . . . . .	635
9.18.1 Detailed Description . . . . .	636
9.19 gpt_lld.c File Reference . . . . .	636
9.19.1 Detailed Description . . . . .	636
9.20 gpt_lld.h File Reference . . . . .	636
9.20.1 Detailed Description . . . . .	637
9.21 hal.c File Reference . . . . .	637
9.21.1 Detailed Description . . . . .	638
9.22 hal.h File Reference . . . . .	638
9.22.1 Detailed Description . . . . .	639
9.23 hal_buffers.c File Reference . . . . .	639
9.23.1 Detailed Description . . . . .	640
9.24 hal_buffers.h File Reference . . . . .	640
9.24.1 Detailed Description . . . . .	642
9.25 hal_channels.h File Reference . . . . .	642
9.25.1 Detailed Description . . . . .	643
9.26 hal_files.h File Reference . . . . .	643
9.26.1 Detailed Description . . . . .	644
9.27 hal_ioblock.h File Reference . . . . .	644
9.27.1 Detailed Description . . . . .	645
9.28 hal_lld.c File Reference . . . . .	646
9.28.1 Detailed Description . . . . .	646
9.29 hal_lld.h File Reference . . . . .	646
9.29.1 Detailed Description . . . . .	646
9.30 hal_mmcisd.c File Reference . . . . .	646
9.30.1 Detailed Description . . . . .	647
9.31 hal_mmcisd.h File Reference . . . . .	647
9.31.1 Detailed Description . . . . .	651
9.32 hal_queues.c File Reference . . . . .	651
9.32.1 Detailed Description . . . . .	652
9.33 hal_queues.h File Reference . . . . .	652
9.33.1 Detailed Description . . . . .	653
9.34 hal_streams.h File Reference . . . . .	653
9.34.1 Detailed Description . . . . .	654
9.35 halconf.h File Reference . . . . .	654
9.35.1 Detailed Description . . . . .	656

9.36 i2c.c File Reference . . . . .	657
9.36.1 Detailed Description . . . . .	657
9.37 i2c.h File Reference . . . . .	657
9.37.1 Detailed Description . . . . .	659
9.38 i2c_lld.c File Reference . . . . .	659
9.38.1 Detailed Description . . . . .	659
9.39 i2c_lld.h File Reference . . . . .	659
9.39.1 Detailed Description . . . . .	660
9.40 i2s.c File Reference . . . . .	660
9.40.1 Detailed Description . . . . .	661
9.41 i2s.h File Reference . . . . .	661
9.41.1 Detailed Description . . . . .	662
9.42 i2s_lld.c File Reference . . . . .	662
9.42.1 Detailed Description . . . . .	662
9.43 i2s_lld.h File Reference . . . . .	662
9.43.1 Detailed Description . . . . .	663
9.44 icu.c File Reference . . . . .	663
9.44.1 Detailed Description . . . . .	664
9.45 icu.h File Reference . . . . .	664
9.45.1 Detailed Description . . . . .	665
9.46 icu_lld.c File Reference . . . . .	665
9.46.1 Detailed Description . . . . .	666
9.47 icu_lld.h File Reference . . . . .	666
9.47.1 Detailed Description . . . . .	667
9.48 mac.c File Reference . . . . .	667
9.48.1 Detailed Description . . . . .	668
9.49 mac.h File Reference . . . . .	668
9.49.1 Detailed Description . . . . .	669
9.50 mac_lld.c File Reference . . . . .	669
9.50.1 Detailed Description . . . . .	670
9.51 mac_lld.h File Reference . . . . .	670
9.51.1 Detailed Description . . . . .	671
9.52 mii.h File Reference . . . . .	671
9.52.1 Detailed Description . . . . .	673
9.53 mmc_spi.c File Reference . . . . .	673
9.53.1 Detailed Description . . . . .	674
9.54 mmc_spi.h File Reference . . . . .	674
9.54.1 Detailed Description . . . . .	676
9.55 osal.c File Reference . . . . .	676
9.55.1 Detailed Description . . . . .	677

9.56 osal.h File Reference . . . . .	677
9.56.1 Detailed Description . . . . .	681
9.57 pal.c File Reference . . . . .	681
9.57.1 Detailed Description . . . . .	681
9.58 pal.h File Reference . . . . .	681
9.58.1 Detailed Description . . . . .	683
9.59 pal_lld.c File Reference . . . . .	683
9.59.1 Detailed Description . . . . .	684
9.60 pal_lld.h File Reference . . . . .	684
9.60.1 Detailed Description . . . . .	685
9.61 pwm.c File Reference . . . . .	685
9.61.1 Detailed Description . . . . .	686
9.62 pwm.h File Reference . . . . .	686
9.62.1 Detailed Description . . . . .	688
9.63 pwm_lld.c File Reference . . . . .	688
9.63.1 Detailed Description . . . . .	689
9.64 pwm_lld.h File Reference . . . . .	689
9.64.1 Detailed Description . . . . .	690
9.65 rtc.c File Reference . . . . .	690
9.65.1 Detailed Description . . . . .	690
9.66 rtc.h File Reference . . . . .	691
9.66.1 Detailed Description . . . . .	692
9.67 rtc_lld.c File Reference . . . . .	692
9.67.1 Detailed Description . . . . .	692
9.68 rtc_lld.h File Reference . . . . .	692
9.68.1 Detailed Description . . . . .	694
9.69 sdc.c File Reference . . . . .	694
9.69.1 Detailed Description . . . . .	695
9.70 sdc.h File Reference . . . . .	695
9.70.1 Detailed Description . . . . .	697
9.71 sdc_lld.c File Reference . . . . .	697
9.71.1 Detailed Description . . . . .	698
9.72 sdc_lld.h File Reference . . . . .	698
9.72.1 Detailed Description . . . . .	699
9.73 serial.c File Reference . . . . .	699
9.73.1 Detailed Description . . . . .	700
9.74 serial.h File Reference . . . . .	700
9.74.1 Detailed Description . . . . .	701
9.75 serial_lld.c File Reference . . . . .	701
9.75.1 Detailed Description . . . . .	702

---

9.76 serial_Ild.h File Reference . . . . .	702
9.76.1 Detailed Description . . . . .	703
9.77 serial_usb.c File Reference . . . . .	703
9.77.1 Detailed Description . . . . .	703
9.78 serial_usb.h File Reference . . . . .	703
9.78.1 Detailed Description . . . . .	705
9.79 spi.c File Reference . . . . .	705
9.79.1 Detailed Description . . . . .	706
9.80 spi.h File Reference . . . . .	706
9.80.1 Detailed Description . . . . .	707
9.81 spi_Ild.c File Reference . . . . .	707
9.81.1 Detailed Description . . . . .	708
9.82 spi_Ild.h File Reference . . . . .	708
9.82.1 Detailed Description . . . . .	709
9.83 st.c File Reference . . . . .	709
9.83.1 Detailed Description . . . . .	710
9.84 st.h File Reference . . . . .	710
9.84.1 Detailed Description . . . . .	710
9.85 st_Ild.c File Reference . . . . .	710
9.85.1 Detailed Description . . . . .	711
9.86 st_Ild.h File Reference . . . . .	711
9.86.1 Detailed Description . . . . .	711
9.87 uart.c File Reference . . . . .	711
9.87.1 Detailed Description . . . . .	712
9.88 uart.h File Reference . . . . .	712
9.88.1 Detailed Description . . . . .	714
9.89 uart_Ild.c File Reference . . . . .	714
9.89.1 Detailed Description . . . . .	715
9.90 uart_Ild.h File Reference . . . . .	715
9.90.1 Detailed Description . . . . .	716
9.91 usb.c File Reference . . . . .	716
9.91.1 Detailed Description . . . . .	717
9.92 usb.h File Reference . . . . .	717
9.92.1 Detailed Description . . . . .	720
9.93 usb_cdc.h File Reference . . . . .	720
9.93.1 Detailed Description . . . . .	721
9.94 usb_Ild.c File Reference . . . . .	722
9.94.1 Detailed Description . . . . .	723
9.94.2 Variable Documentation . . . . .	723
9.94.2.1 in . . . . .	723

9.94.2.2	out	723
9.95	usb_lld.h File Reference	723
9.95.1	Detailed Description	725
9.96	wdg.c File Reference	725
9.96.1	Detailed Description	725
9.97	wdg.h File Reference	725
9.97.1	Detailed Description	726
9.98	wdg_lld.c File Reference	726
9.98.1	Detailed Description	726
9.99	wdg_lld.h File Reference	726
9.99.1	Detailed Description	727



# **Chapter 1**

## **ChibiOS/HAL**

### **1.1 Copyright**

Copyright (C) 2006..2015 Giovanni Di Sirio. All rights reserved.

Neither the whole nor any part of the information contained in this document may be adapted or reproduced in any form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by Giovanni Di Sirio in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Giovanni Di Sirio shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

### **1.2 Introduction**

This document is the Reference Manual for the ChibiOS/HAL hardware abstraction layer.

### **1.3 Related Documents**

- ChibiOS/HAL General Architecture



## Chapter 2

### Deprecated List

globalScope> Global **sdGetWouldBlock** (SerialDriver \*sdp)

globalScope> Global **sdPutWouldBlock** (SerialDriver \*sdp)



# Chapter 3

## Module Index

### 3.1 Modules

Here is a list of all modules:

HAL . . . . .	212
Configuration . . . . .	215
Normal Drivers . . . . .	222
ADC Driver . . . . .	17
CAN Driver . . . . .	35
DAC Driver . . . . .	51
EXT Driver . . . . .	68
GPT Driver . . . . .	78
HAL Driver . . . . .	95
I2C Driver . . . . .	156
I2S Driver . . . . .	170
ICU Driver . . . . .	178
MAC Driver . . . . .	196
PAL Driver . . . . .	292
PWM Driver . . . . .	318
RTC Driver . . . . .	340
SDC Driver . . . . .	350
Serial Driver . . . . .	379
SPI Driver . . . . .	405
ST Driver . . . . .	429
UART Driver . . . . .	435
USB Driver . . . . .	461
WDG Driver . . . . .	505
Complex Drivers . . . . .	223
MMC over SPI Driver . . . . .	260
Serial over USB Driver . . . . .	394
Interfaces . . . . .	224
Abstract I/O Channel . . . . .	118
Abstract Files . . . . .	124
Abstract I/O Block Device . . . . .	130
Abstract Streams . . . . .	153
Inner Code . . . . .	225
I/O Buffers Queues . . . . .	98
I/O Bytes Queues . . . . .	137
MMC/SD Block Device . . . . .	282
Support Code . . . . .	226
MII/RMII Header . . . . .	250

USB CDC Header . . . . .	503
OSAL . . . . .	227

# Chapter 4

## Hierarchical Index

### 4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ADCConfig . . . . .	511
ADCConversionGroup . . . . .	512
ADCDriver . . . . .	513
BaseBlockDevice . . . . .	519
MMCSDBlockDevice . . . . .	570
MMCDriver . . . . .	567
BaseBlockDeviceVMT . . . . .	521
MMCSDBlockDeviceVMT . . . . .	572
MMCDriverVMT . . . . .	569
SDCDriverVMT . . . . .	586
BaseSequentialStream . . . . .	526
BaseChannel . . . . .	522
BaseAsynchronousChannel . . . . .	515
SerialDriver . . . . .	588
SerialUSBDriver . . . . .	592
FileStream . . . . .	545
RTCDriverVMT . . . . .	582
BaseSequentialStreamVMT . . . . .	527
BaseChannelVMT . . . . .	524
BaseAsynchronousChannelVMT . . . . .	517
SerialDriverVMT . . . . .	589
SerialUSBDriverVMT . . . . .	594
FileStreamVMT . . . . .	546
BlockDeviceInfo . . . . .	528
CANConfig . . . . .	529
CANDriver . . . . .	530
CANRxFrame . . . . .	532
CANTxFrame . . . . .	534
cdc_linecoding_t . . . . .	536
DACConfig . . . . .	536
DACConversionGroup . . . . .	537
DACDriver . . . . .	538
event_source . . . . .	540
EXTChannelConfig . . . . .	541
EXTConfig . . . . .	543
EXTDriver . . . . .	544

GPTConfig . . . . .	547
GPTDriver . . . . .	549
I2CConfig . . . . .	549
I2CDriver . . . . .	550
I2SConfig . . . . .	551
I2SDriver . . . . .	553
ICUConfig . . . . .	554
ICUDriver . . . . .	555
io_buffers_queue . . . . .	556
io_queue . . . . .	559
IOBus . . . . .	561
MACConfig . . . . .	562
MACDriver . . . . .	562
MACReceiveDescriptor . . . . .	564
MACTransmitDescriptor . . . . .	565
MMCConfig . . . . .	565
PALConfig . . . . .	574
PWMChannelConfig . . . . .	574
PWMConfig . . . . .	576
PWMDriver . . . . .	577
RTCAlarm . . . . .	579
RTCDateTime . . . . .	579
RTCDriver . . . . .	581
SDCConfig . . . . .	583
SDCDriver . . . . .	584
SerialConfig . . . . .	587
SerialUSBConfig . . . . .	591
SPIConfig . . . . .	596
SPIDriver . . . . .	597
threads_queue_t . . . . .	598
UARTConfig . . . . .	598
UARTDriver . . . . .	600
unpacked_mmc_cid_t . . . . .	602
unpacked_mmc_csd_t . . . . .	603
unpacked_sdc_cid_t . . . . .	604
unpacked_sdc_csd_10_t . . . . .	604
unpacked_sdc_csd_20_t . . . . .	605
USBConfig . . . . .	606
USBDescriptor . . . . .	608
USBDriver . . . . .	609
USBEndpointConfig . . . . .	612
USBInEndpointState . . . . .	614
USBOutEndpointState . . . . .	615
WDGConfig . . . . .	616
WDGDriver . . . . .	617

## Chapter 5

# Data Structure Index

### 5.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">ADCConfig</a>	Driver configuration structure . . . . .	511
<a href="#">ADCConversionGroup</a>	Conversion group configuration structure . . . . .	512
<a href="#">ADCDriver</a>	Structure representing an ADC driver . . . . .	513
<a href="#">BaseAsynchronousChannel</a>	Base asynchronous channel class . . . . .	515
<a href="#">BaseAsynchronousChannelVMT</a>	<a href="#">BaseAsynchronousChannel</a> virtual methods table . . . . .	517
<a href="#">BaseBlockDevice</a>	Base block device class . . . . .	519
<a href="#">BaseBlockDeviceVMT</a>	<a href="#">BaseBlockDevice</a> virtual methods table . . . . .	521
<a href="#">BaseChannel</a>	Base channel class . . . . .	522
<a href="#">BaseChannelVMT</a>	<a href="#">BaseChannel</a> virtual methods table . . . . .	524
<a href="#">BaseSequentialStream</a>	Base stream class . . . . .	526
<a href="#">BaseSequentialStreamVMT</a>	<a href="#">BaseSequentialStream</a> virtual methods table . . . . .	527
<a href="#">BlockDeviceInfo</a>	Block device info . . . . .	528
<a href="#">CANConfig</a>	Driver configuration structure . . . . .	529
<a href="#">CANDriver</a>	Structure representing an CAN driver . . . . .	530
<a href="#">CANRxFrame</a>	CAN received frame . . . . .	532
<a href="#">CANTxFrame</a>	CAN transmission frame . . . . .	534
<a href="#">cdc_linecoding_t</a>	Type of Line Coding structure . . . . .	536
<a href="#">DACConfig</a>	Driver configuration structure . . . . .	536
<a href="#">DACConversionGroup</a>	DAC Conversion group structure . . . . .	537

DACDriver	Structure representing a DAC driver . . . . .	538
event_source	Events source object . . . . .	540
EXTChannelConfig	Channel configuration structure . . . . .	541
EXTConfig	Driver configuration structure . . . . .	543
EXTDriver	Structure representing an EXT driver . . . . .	544
FileStream	Base file stream class . . . . .	545
FileStreamVMT	FileStream virtual methods table . . . . .	546
GPTConfig	Driver configuration structure . . . . .	547
GPTDriver	Structure representing a GPT driver . . . . .	549
I2CConfig	Type of I2C driver configuration structure . . . . .	549
I2CDriver	Structure representing an I2C driver . . . . .	550
I2SConfig	Driver configuration structure . . . . .	551
I2SDriver	Structure representing an I2S driver . . . . .	553
ICUConfig	Driver configuration structure . . . . .	554
ICUDriver	Structure representing an ICU driver . . . . .	555
io_buffers_queue	Structure of a generic buffers queue . . . . .	556
io_queue	Generic I/O queue structure . . . . .	559
IOBus	I/O bus descriptor . . . . .	561
MACConfig	Driver configuration structure . . . . .	562
MACDriver	Structure representing a MAC driver . . . . .	562
MACReceiveDescriptor	Structure representing a receive descriptor . . . . .	564
MACTransmitDescriptor	Structure representing a transmit descriptor . . . . .	565
MMCConfig	MMC/SD over SPI driver configuration structure . . . . .	565
MMCDriver	Structure representing a MMC/SD over SPI driver . . . . .	567
MMCDriverVMT	MMCDriver virtual methods table . . . . .	569
MMCSDBlockDevice	MCC/SD block device class . . . . .	570
MMCSDBlockDeviceVMT	MMCSDBlockDevice virtual methods table . . . . .	572
PALConfig	Generic I/O ports static initializer . . . . .	574
PWMChannelConfig	Type of a PWM driver channel configuration structure . . . . .	574

<b>PWMConfig</b>	Type of a PWM driver configuration structure . . . . .	576
<b>PWMDriver</b>	Structure representing a PWM driver . . . . .	577
<b>RTCAlarm</b>	Type of a structure representing an RTC alarm time stamp . . . . .	579
<b>RTCDateTime</b>	Type of a structure representing an RTC date/time stamp . . . . .	579
<b>RTCDriver</b>	Structure representing an RTC driver . . . . .	581
<b>RTCDriverVMT</b>	<b>RTCDriver</b> virtual methods table . . . . .	582
<b>SDCConfig</b>	Driver configuration structure . . . . .	583
<b>SDCDriver</b>	Structure representing an SDC driver . . . . .	584
<b>SDCDriverVMT</b>	<b>SDCDriver</b> virtual methods table . . . . .	586
<b>SerialConfig</b>	PLATFORM Serial Driver configuration structure . . . . .	587
<b>SerialDriver</b>	Full duplex serial driver class . . . . .	588
<b>SerialDriverVMT</b>	<b>SerialDriver</b> virtual methods table . . . . .	589
<b>SerialUSBCConfig</b>	Serial over USB Driver configuration structure . . . . .	591
<b>SerialUSBDriver</b>	Full duplex serial driver class . . . . .	592
<b>SerialUSBDriverVMT</b>	<b>SerialDriver</b> virtual methods table . . . . .	594
<b>SPIConfig</b>	Driver configuration structure . . . . .	596
<b>SPIDriver</b>	Structure representing an SPI driver . . . . .	597
<b>threads_queue_t</b>	Type of a thread queue . . . . .	598
<b>UARTConfig</b>	Driver configuration structure . . . . .	598
<b>UARTDriver</b>	Structure representing an UART driver . . . . .	600
<b>unpacked_mmc_cid_t</b>	Unpacked CID register from MMC . . . . .	602
<b>unpacked_mmc_csd_t</b>	Unpacked CSD register from MMC . . . . .	603
<b>unpacked_sdc_cid_t</b>	Unpacked CID register from SDC . . . . .	604
<b>unpacked_sdc_csd_10_t</b>	Unpacked CSD v1.0 register from SDC . . . . .	604
<b>unpacked_sdc_csd_20_t</b>	Unpacked CSD v2.0 register from SDC . . . . .	605
<b>USBConfig</b>	Type of an USB driver configuration structure . . . . .	606
<b>USBDescriptor</b>	Type of an USB descriptor . . . . .	608
<b>USBDriver</b>	Structure representing an USB driver . . . . .	609
<b>USBEndpointConfig</b>	Type of an USB endpoint configuration structure . . . . .	612

<a href="#">USBInEndpointState</a>	Type of an IN endpoint state structure . . . . .	614
<a href="#">USBOutEndpointState</a>	Type of an OUT endpoint state structure . . . . .	615
<a href="#">WDGConfig</a>	Driver configuration structure . . . . .	616
<a href="#">WDGDriver</a>	Structure representing an WDG driver . . . . .	617

# Chapter 6

## File Index

### 6.1 File List

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

<a href="#">adc.c</a>	ADC Driver code . . . . .	619
<a href="#">adc.h</a>	ADC Driver macros and structures . . . . .	620
<a href="#">adc_lld.c</a>	PLATFORM ADC subsystem low level driver source . . . . .	621
<a href="#">adc_lld.h</a>	PLATFORM ADC subsystem low level driver header . . . . .	621
<a href="#">can.c</a>	CAN Driver code . . . . .	623
<a href="#">can.h</a>	CAN Driver macros and structures . . . . .	623
<a href="#">can_lld.c</a>	PLATFORM CAN subsystem low level driver source . . . . .	624
<a href="#">can_lld.h</a>	PLATFORM CAN subsystem low level driver header . . . . .	625
<a href="#">dac.c</a>	DAC Driver code . . . . .	626
<a href="#">dac.h</a>	DAC Driver macros and structures . . . . .	627
<a href="#">dac_lld.c</a>	PLATFORM DAC subsystem low level driver source . . . . .	629
<a href="#">dac_lld.h</a>	PLATFORM DAC subsystem low level driver header . . . . .	629
<a href="#">ext.c</a>	EXT Driver code . . . . .	631
<a href="#">ext.h</a>	EXT Driver macros and structures . . . . .	631
<a href="#">ext_lld.c</a>	PLATFORM EXT subsystem low level driver source . . . . .	632
<a href="#">ext_lld.h</a>	PLATFORM EXT subsystem low level driver header . . . . .	633
<a href="#">gpt.c</a>	GPT Driver code . . . . .	634
<a href="#">gpt.h</a>	GPT Driver macros and structures . . . . .	635
<a href="#">gpt_lld.c</a>	PLATFORM GPT subsystem low level driver source . . . . .	636

gpt_ll.h	PLATFORM GPT subsystem low level driver header . . . . .	636
hal.c	HAL subsystem code . . . . .	637
hal.h	HAL subsystem header . . . . .	638
hal_buffers.c	I/O Buffers code . . . . .	639
hal_buffers.h	I/O Buffers macros and structures . . . . .	640
hal_channels.h	I/O channels access . . . . .	642
hal_files.h	Data files . . . . .	643
hal_ioblock.h	I/O block devices access . . . . .	644
hal_ll.c	PLATFORM HAL subsystem low level driver source . . . . .	646
hal_ll.h	PLATFORM HAL subsystem low level driver header . . . . .	646
hal_mmcsrcd.c	MMC/SD cards common code . . . . .	646
hal_mmcsrcd.h	MMC/SD cards common header . . . . .	647
hal_queues.c	I/O Queues code . . . . .	651
hal_queues.h	I/O Queues macros and structures . . . . .	652
hal_streams.h	Data streams . . . . .	653
halconf.h	HAL configuration header . . . . .	654
i2c.c	I2C Driver code . . . . .	657
i2c.h	I2C Driver macros and structures . . . . .	657
i2c_ll.c	PLATFORM I2C subsystem low level driver source . . . . .	659
i2c_ll.h	PLATFORM I2C subsystem low level driver header . . . . .	659
i2s.c	I2S Driver code . . . . .	660
i2s.h	I2S Driver macros and structures . . . . .	661
i2s_ll.c	PLATFORM I2S subsystem low level driver source . . . . .	662
i2s_ll.h	PLATFORM I2S subsystem low level driver header . . . . .	662
icu.c	ICU Driver code . . . . .	663
icu.h	ICU Driver macros and structures . . . . .	664
icu_ll.c	PLATFORM ADC subsystem low level driver source . . . . .	665
icu_ll.h	PLATFORM ICU subsystem low level driver header . . . . .	666
mac.c	MAC Driver code . . . . .	667

mac.h	MAC Driver macros and structures . . . . .	668
mac_lld.c	PLATFORM MAC subsystem low level driver source . . . . .	669
mac_lld.h	PLATFORM MAC subsystem low level driver header . . . . .	670
mii.h	MII macros and structures . . . . .	671
mmc_spi.c	MMC over SPI driver code . . . . .	673
mmc_spi.h	MMC over SPI driver header . . . . .	674
osal.c	OSAL module code . . . . .	676
osal.h	OSAL module header . . . . .	677
pal.c	I/O Ports Abstraction Layer code . . . . .	681
pal.h	I/O Ports Abstraction Layer macros, types and structures . . . . .	681
pal_lld.c	PLATFORM PAL subsystem low level driver source . . . . .	683
pal_lld.h	PLATFORM PAL subsystem low level driver header . . . . .	684
pwm.c	PWM Driver code . . . . .	685
pwm.h	PWM Driver macros and structures . . . . .	686
pwm_lld.c	PLATFORM PWM subsystem low level driver source . . . . .	688
pwm_lld.h	PLATFORM PWM subsystem low level driver header . . . . .	689
rtc.c	RTC Driver code . . . . .	690
rtc.h	RTC Driver macros and structures . . . . .	691
rtc_lld.c	PLATFORM RTC subsystem low level driver source . . . . .	692
rtc_lld.h	PLATFORM RTC subsystem low level driver header . . . . .	692
sdc.c	SDC Driver code . . . . .	694
sdc.h	SDC Driver macros and structures . . . . .	695
sdc_lld.c	PLATFORM SDC subsystem low level driver source . . . . .	697
sdc_lld.h	PLATFORM SDC subsystem low level driver header . . . . .	698
serial.c	Serial Driver code . . . . .	699
serial.h	Serial Driver macros and structures . . . . .	700
serial_lld.c	PLATFORM serial subsystem low level driver source . . . . .	701
serial_lld.h	PLATFORM serial subsystem low level driver header . . . . .	702
serial_usb.c	Serial over USB Driver code . . . . .	703

<a href="#">serial_usb.h</a>	Serial over USB Driver macros and structures . . . . .	703
<a href="#">spi.c</a>	SPI Driver code . . . . .	705
<a href="#">spi.h</a>	SPI Driver macros and structures . . . . .	706
<a href="#">spi_lld.c</a>	PLATFORM SPI subsystem low level driver source . . . . .	707
<a href="#">spi_lld.h</a>	PLATFORM SPI subsystem low level driver header . . . . .	708
<a href="#">st.c</a>	ST Driver code . . . . .	709
<a href="#">st.h</a>	ST Driver macros and structures . . . . .	710
<a href="#">st_lld.c</a>	PLATFORM ST subsystem low level driver source . . . . .	710
<a href="#">st_lld.h</a>	PLATFORM ST subsystem low level driver header . . . . .	711
<a href="#">uart.c</a>	UART Driver code . . . . .	711
<a href="#">uart.h</a>	UART Driver macros and structures . . . . .	712
<a href="#">uart_lld.c</a>	PLATFORM UART subsystem low level driver source . . . . .	714
<a href="#">uart_lld.h</a>	PLATFORM UART subsystem low level driver header . . . . .	715
<a href="#">usb.c</a>	USB Driver code . . . . .	716
<a href="#">usb.h</a>	USB Driver macros and structures . . . . .	717
<a href="#">usb_cdc.h</a>	USB CDC macros and structures . . . . .	720
<a href="#">usb_lld.c</a>	PLATFORM USB subsystem low level driver source . . . . .	722
<a href="#">usb_lld.h</a>	PLATFORM USB subsystem low level driver header . . . . .	723
<a href="#">wdg.c</a>	WDG Driver code . . . . .	725
<a href="#">wdg.h</a>	WDG Driver macros and structures . . . . .	725
<a href="#">wdg_lld.c</a>	WDG Driver subsystem low level driver source template . . . . .	726
<a href="#">wdg_lld.h</a>	WDG Driver subsystem low level driver header template . . . . .	726

# Chapter 7

## Module Documentation

### 7.1 ADC Driver

Generic ADC Driver.

#### 7.1.1 Detailed Description

Generic ADC Driver.

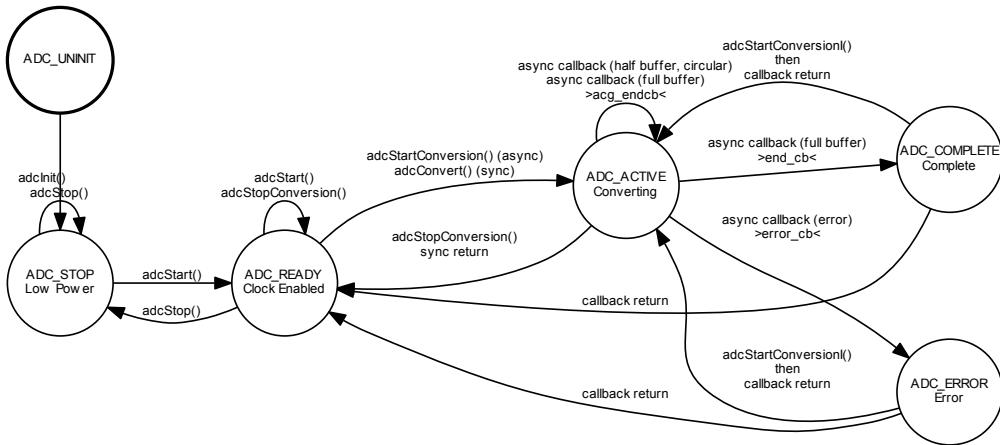
This module implements a generic ADC (Analog to Digital Converter) driver supporting a variety of buffer and conversion modes.

#### Precondition

In order to use the ADC driver the `HAL_USE_ADC` option must be enabled in `halconf.h`.

#### 7.1.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



### 7.1.3 ADC Operations

The ADC driver is quite complex, an explanation of the terminology and of the operational details follows.

#### 7.1.3.1 ADC Conversion Groups

The [ADCConversionGroup](#) is the objects that specifies a physical conversion operation. This structure contains some standard fields and several implementation-dependent fields.

The standard fields define the CG mode, the number of channels belonging to the CG and the optional callbacks. The implementation-dependent fields specify the physical ADC operation mode, the analog channels belonging to the group and any other implementation-specific setting. Usually the extra fields just mirror the physical ADC registers, please refer to the vendor's MCU Reference Manual for details about the available settings. Details are also available into the documentation of the ADC low level drivers and in the various sample applications.

#### 7.1.3.2 ADC Conversion Modes

The driver supports several conversion modes:

- **One Shot**, the driver performs a single group conversion then stops.
- **Linear Buffer**, the driver performs a series of group conversions then stops. This mode is like a one shot conversion repeated N times, the buffer pointer increases after each conversion. The buffer is organized as an S(CG)\*N samples matrix, when S(CG) is the conversion group size (number of channels) and N is the buffer depth (number of repeated conversions).
- **Circular Buffer**, much like the linear mode but the operation does not stop when the buffer is filled, it is automatically restarted with the buffer pointer wrapping back to the buffer base.

#### 7.1.3.3 ADC Callbacks

The driver is able to invoke callbacks during the conversion process. A callback is invoked when the operation has been completed or, in circular mode, when the buffer has been filled and the operation is restarted. In circular mode a callback is also invoked when the buffer is half filled.

The "half filled" and "filled" callbacks in circular mode allow to implement "streaming processing" of the sampled

data, while the driver is busy filling one half of the buffer the application can process the other half, this allows for continuous interleaved operations.

The driver is not thread safe for performance reasons, if you need to access the ADC bus from multiple threads then use the `adcAcquireBus()` and `adcReleaseBus()` APIs in order to gain exclusive access.

## ADC configuration options

- `#define ADC_USE_WAIT TRUE`  
*Enables synchronous APIs.*
- `#define ADC_USE_MUTUAL_EXCLUSION TRUE`  
*Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.*

## Low level driver helper macros

- `#define _adc_reset_i(adcp) osalThreadResumel(&(adcp)->thread, MSG_RESET)`  
*Resumes a thread waiting for a conversion completion.*
- `#define _adc_reset_s(adcp) osalThreadResumeS(&(adcp)->thread, MSG_RESET)`  
*Resumes a thread waiting for a conversion completion.*
- `#define _adc_wakeup_isr(adcp)`  
*Wakes up the waiting thread.*
- `#define _adc_timeout_isr(adcp)`  
*Wakes up the waiting thread with a timeout message.*
- `#define _adc_isr_half_code(adcp)`  
*Common ISR code, half buffer event.*
- `#define _adc_isr_full_code(adcp)`  
*Common ISR code, full buffer event.*
- `#define _adc_isr_error_code(adcp, err)`  
*Common ISR code, error event.*

## PLATFORM configuration options

- `#define PLATFORM_ADC_USE_ADC1 FALSE`  
*ADC1 driver enable switch.*

## Typedefs

- `typedef uint16_t adcsample_t`  
*ADC sample data type.*
- `typedef uint16_t adc_channels_num_t`  
*Channels number in a conversion group.*
- `typedef struct ADCDriver ADCDriver`  
*Type of a structure representing an ADC driver.*
- `typedef void(* adccallback_t) (ADCDriver *adcp, adcsample_t *buffer, size_t n)`  
*ADC notification callback type.*
- `typedef void(* adcerrorcallback_t) (ADCDriver *adcp, adcerror_t err)`  
*ADC error callback type.*

## Data Structures

- struct [ADCConversionGroup](#)  
*Conversion group configuration structure.*
- struct [ADCConfig](#)  
*Driver configuration structure.*
- struct [ADCDriver](#)  
*Structure representing an ADC driver.*

## Functions

- void [adclinit](#) (void)  
*ADC Driver initialization.*
- void [adcObjectInit](#) ([ADCDriver](#) \*adcp)  
*Initializes the standard part of a [ADCDriver](#) structure.*
- void [adcStart](#) ([ADCDriver](#) \*adcp, const [ADCConfig](#) \*config)  
*Configures and activates the ADC peripheral.*
- void [adcStop](#) ([ADCDriver](#) \*adcp)  
*Deactivates the ADC peripheral.*
- void [adcStartConversion](#) ([ADCDriver](#) \*adcp, const [ADCConversionGroup](#) \*grpp, [adcsample\\_t](#) \*samples, size\_t depth)  
*Starts an ADC conversion.*
- void [adcStartConversionI](#) ([ADCDriver](#) \*adcp, const [ADCConversionGroup](#) \*grpp, [adcsample\\_t](#) \*samples, size\_t depth)  
*Starts an ADC conversion.*
- void [adcStopConversion](#) ([ADCDriver](#) \*adcp)  
*Stops an ongoing conversion.*
- void [adcStopConversionI](#) ([ADCDriver](#) \*adcp)  
*Stops an ongoing conversion.*
- msg\_t [adcConvert](#) ([ADCDriver](#) \*adcp, const [ADCConversionGroup](#) \*grpp, [adcsample\\_t](#) \*samples, size\_t depth)  
*Performs an ADC conversion.*
- void [adcAcquireBus](#) ([ADCDriver](#) \*adcp)  
*Gains exclusive access to the ADC peripheral.*
- void [adcReleaseBus](#) ([ADCDriver](#) \*adcp)  
*Releases exclusive access to the ADC peripheral.*
- void [adc\\_lld\\_init](#) (void)  
*Low level ADC driver initialization.*
- void [adc\\_lld\\_start](#) ([ADCDriver](#) \*adcp)  
*Configures and activates the ADC peripheral.*
- void [adc\\_lld\\_stop](#) ([ADCDriver](#) \*adcp)  
*Deactivates the ADC peripheral.*
- void [adc\\_lld\\_start\\_conversion](#) ([ADCDriver](#) \*adcp)  
*Starts an ADC conversion.*
- void [adc\\_lld\\_stop\\_conversion](#) ([ADCDriver](#) \*adcp)  
*Stops an ongoing conversion.*

## Enumerations

- enum `adcstate_t` {
 `ADC_UNINIT` = 0, `ADC_STOP` = 1, `ADC_READY` = 2, `ADC_ACTIVE` = 3,  
`ADC_COMPLETE` = 4, `ADC_ERROR` = 5 }
   
*Driver state machine possible states.*
- enum `adcerror_t` { `ADC_ERR_DMAFAILURE` = 0, `ADC_ERR_OVERFLOW` = 1, `ADC_ERR_AWD` = 2 }
   
*Possible ADC failure causes.*

## Variables

- `ADCDriver ADCD1`  
*ADC1 driver identifier.*

### 7.1.4 Macro Definition Documentation

#### 7.1.4.1 `#define ADC_USE_WAIT TRUE`

Enables synchronous APIs.

##### Note

Disabling this option saves both code and data space.

#### 7.1.4.2 `#define ADC_USE_MUTUAL_EXCLUSION TRUE`

Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

##### Note

Disabling this option saves both code and data space.

#### 7.1.4.3 `#define _adc_reset_i( adcp ) osalThreadResumel(&(adcp)->thread, MSG_RESET)`

Resumes a thread waiting for a conversion completion.

##### Parameters

in	<code>adcp</code>	pointer to the <code>ADCDriver</code> object
----	-------------------	--

##### Function Class:

Not an API, this function is for internal use only.

#### 7.1.4.4 `#define _adc_reset_s( adcp ) osalThreadResumeS(&(adcp)->thread, MSG_RESET)`

Resumes a thread waiting for a conversion completion.

##### Parameters

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.1.4.5 #define \_adc\_wakeup\_isr( *adcp* )****Value:**

```
{
  osalSysLockFromISR();
  \
  osalThreadResumeI (& (adcp)->thread, MSG_OK);
  \
  osalSysUnlockFromISR();
}
```

Wakes up the waiting thread.

**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.1.4.6 #define \_adc\_timeout\_isr( *adcp* )****Value:**

```
{
  osalSysLockFromISR();
  \
  osalThreadResumeI (& (adcp)->thread, MSG_TIMEOUT);
  \
  osalSysUnlockFromISR();
}
```

Wakes up the waiting thread with a timeout message.

**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.1.4.7 #define \_adc\_isr\_half\_code( *adcp* )****Value:**

```
{
  if ((adcp)->grpp->end_cb != NULL) {
    (adcp)->grpp->end_cb(adcp, (adcp)->samples, (adcp)->depth / 2);
  }
}
```

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.1.4.8 #define \_adc\_isr\_full\_code( *adcp* )**

Common ISR code, full buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.1.4.9 #define \_adc\_isr\_error\_code( *adcp*, *err* )****Value:**

```
{
    adc_lld_stop_conversion(adcp);
    \
    if ((adcp)->grpp->error_cb != NULL) {
        \
        (adcp)->state = ADC_ERROR;
        (adcp)->grpp->error_cb(adcp, err);
        \
        if ((adcp)->state == ADC_ERROR)
            (adcp)->state = ADC_READY;
    }
    \
    (adcp)->grpp = NULL;
    \
    _adc_timeout_isr(adcp);
}
```

Common ISR code, error event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread timeout signaling, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
in	<i>err</i>	platform dependent error code

**Function Class:**

Not an API, this function is for internal use only.

**7.1.4.10 #define PLATFORM\_ADC\_USE\_ADC1 FALSE**

ADC1 driver enable switch.

If set to TRUE the support for ADC1 is included.

**Note**

The default is FALSE.

**7.1.5 Typedef Documentation****7.1.5.1 typedef uint16\_t adcsample\_t**

ADC sample data type.

**7.1.5.2 typedef uint16\_t adc\_channels\_num\_t**

Channels number in a conversion group.

**7.1.5.3 typedef struct ADCDriver ADCDriver**

Type of a structure representing an ADC driver.

**7.1.5.4 typedef void(\* adccallback\_t) (ADCDriver \*adcp, adcsample\_t \*buffer, size\_t n)**

ADC notification callback type.

**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object triggering the callback
in	<i>buffer</i>	pointer to the most recent samples data
in	<i>n</i>	number of buffer rows available starting from <i>buffer</i>

**7.1.5.5 typedef void(\* adcerrorcallback\_t) (ADCDriver \*adcp, adcerror\_t err)**

ADC error callback type.

### Parameters

in	<i>adcp</i>	pointer to the <a href="#">ADC Driver</a> object triggering the callback
in	<i>err</i>	ADC error code

## 7.1.6 Enumeration Type Documentation

### 7.1.6.1 enum adcstate\_t

Driver state machine possible states.

#### Enumerator

- ADC\_UNINIT** Not initialized.
- ADC\_STOP** Stopped.
- ADC\_READY** Ready.
- ADC\_ACTIVE** Converting.
- ADC\_COMPLETE** Conversion complete.
- ADC\_ERROR** Conversion complete.

### 7.1.6.2 enum adcerror\_t

Possible ADC failure causes.

#### Note

Error codes are architecture dependent and should not relied upon.

#### Enumerator

- ADC\_ERR\_DMAFAILURE** DMA operations failure.
- ADC\_ERR\_OVERFLOW** ADC overflow condition.
- ADC\_ERR\_AWD** Analog watchdog triggered.

## 7.1.7 Function Documentation

### 7.1.7.1 void adclnit( void )

ADC Driver initialization.

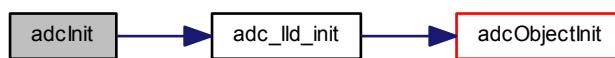
#### Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.1.7.2 void adcObjectInit ( ADCDriver \* *adcp* )

Initializes the standard part of a [ADCDriver](#) structure.

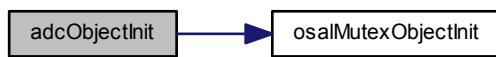
#### Parameters

out	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
-----	-------------	---

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.1.7.3 void adcStart ( ADCDriver \* *adcp*, const ADCConfig \* *config* )

Configures and activates the ADC peripheral.

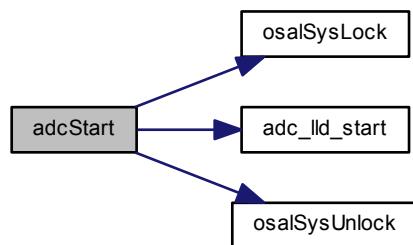
#### Parameters

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
in	<i>config</i>	pointer to the <a href="#">ADCConfig</a> object. Depending on the implementation the value can be NULL.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.1.7.4 void adcStop ( **ADCDriver** \* *adcp* )

Deactivates the ADC peripheral.

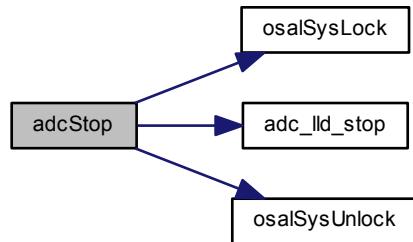
**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



**7.1.7.5 void adcStartConversion ( ADCDriver \* *adcp*, const ADCConversionGroup \* *grpp*, adcsample\_t \* *samples*, size\_t *depth* )**

Starts an ADC conversion.

Starts an asynchronous conversion operation.

**Note**

The buffer is organized as a matrix of M\*N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

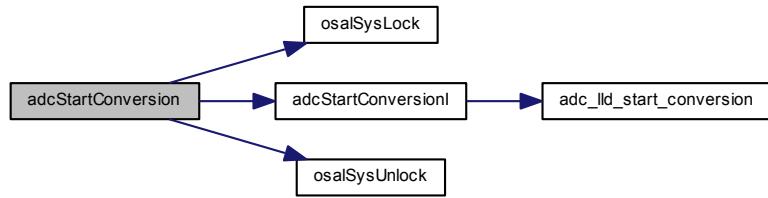
**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
in	<i>grpp</i>	pointer to a <a href="#">ADCConversionGroup</a> object
out	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



**7.1.7.6 void adcStartConversionl ( ADCDriver \* *adcp*, const ADCConversionGroup \* *grpp*, adcsample\_t \* *samples*, size\_t *depth* )**

Starts an ADC conversion.

Starts an asynchronous conversion operation.

#### Postcondition

The callbacks associated to the conversion group will be invoked on buffer fill and error events.

#### Note

The buffer is organized as a matrix of M\*N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

#### Parameters

in	<i>adcp</i>	pointer to the <code>ADCDriver</code> object
in	<i>grpp</i>	pointer to a <code>ADCConversionGroup</code> object
out	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.1.7.7 void adcStopConversion ( ADCDriver \* adcp )

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the ADC\_READY state. If there was no conversion being processed then the function does nothing.

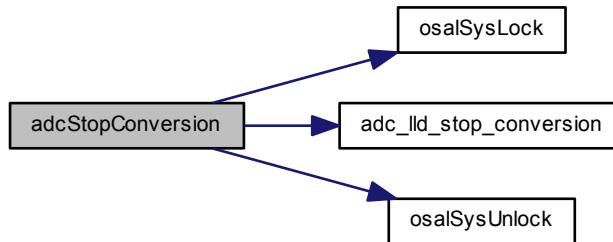
#### Parameters

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.1.7.8 void adcStopConversionI ( ADCDriver \* adcp )

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the ADC\_READY state. If there was no conversion being processed then the function does nothing.

#### Parameters

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.1.7.9 `msg_t adcConvert( ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth )`

Performs an ADC conversion.

Performs a synchronous conversion operation.

#### Note

The buffer is organized as a matrix of M\*N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

#### Parameters

in	<code>adcp</code>	pointer to the <code>ADCDriver</code> object
in	<code>grpp</code>	pointer to a <code>ADCConversionGroup</code> object
out	<code>samples</code>	pointer to the samples buffer
in	<code>depth</code>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

#### Returns

The operation result.

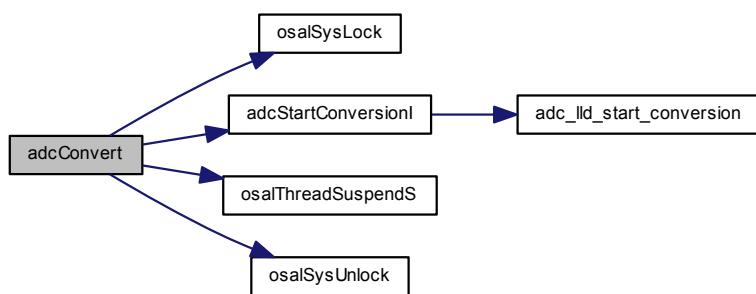
#### Return values

<code>MSG_OK</code>	Conversion finished.
<code>MSG_RESET</code>	The conversion has been stopped using <code>acdStopConversion()</code> or <code>acdStopConversionI()</code> , the result buffer may contain incorrect data.
<code>MSG_TIMEOUT</code>	The conversion has been stopped because an hardware error.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.1.7.10 `void adcAcquireBus( ADCDriver *adcp )`

Gains exclusive access to the ADC peripheral.

This function tries to gain ownership to the ADC bus, if the bus is already being used then the invoking thread is queued.

**Precondition**

In order to use this function the option ADC\_USE\_MUTUAL\_EXCLUSION must be enabled.

**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.1.7.11 void adcReleaseBus ( ADCDriver \* *adcp* )**

Releases exclusive access to the ADC peripheral.

**Precondition**

In order to use this function the option ADC\_USE\_MUTUAL\_EXCLUSION must be enabled.

**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

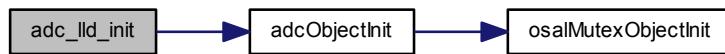
**7.1.7.12 void adc\_lld\_init ( void )**

Low level ADC driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.1.7.13 void adc\_lld\_start ( ADCDriver \* adcp )**

Configures and activates the ADC peripheral.

**Parameters**

in	adcp	pointer to the <a href="#">ADCDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.1.7.14 void adc\_lld\_stop ( ADCDriver \* adcp )**

Deactivates the ADC peripheral.

**Parameters**

in	adcp	pointer to the <a href="#">ADCDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.1.7.15 void adc\_lld\_start\_conversion ( ADCDriver \* adcp )**

Starts an ADC conversion.

**Parameters**

in	adcp	pointer to the <a href="#">ADCDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.1.7.16 void adc\_lld\_stop\_conversion ( ADCDriver \* adcp )**

Stops an ongoing conversion.

**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

## 7.1.8 Variable Documentation

### 7.1.8.1 ADCDriver ADCD1

ADC1 driver identifier.

## 7.2 CAN Driver

Generic CAN Driver.

### 7.2.1 Detailed Description

Generic CAN Driver.

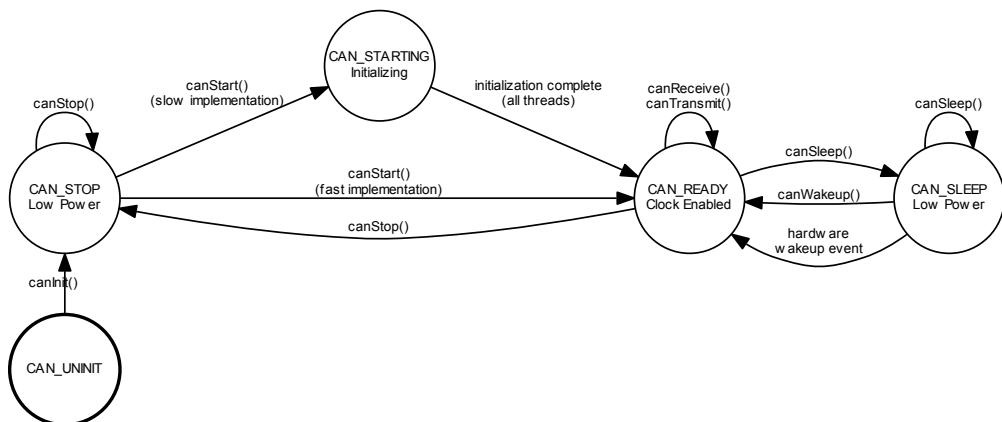
This module implements a generic CAN (Controller Area Network) driver allowing the exchange of information at frame level.

#### Precondition

In order to use the CAN driver the `HAL_USE_CAN` option must be enabled in `halconf.h`.

### 7.2.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



#### Macros

- `#define CAN_ANY_MAILBOX 0`  
*Special mailbox identifier.*
- `#define CAN_TX_MAILBOXES 1`  
*Number of transmit mailboxes.*
- `#define CAN_RX_MAILBOXES 1`  
*Number of receive mailboxes.*

#### CAN status flags

- `#define CAN_LIMIT_WARNING 1U`

- `#define CAN_LIMIT_ERROR 2U`  
*Errors rate warning.*
- `#define CAN_BUS_OFF_ERROR 4U`  
*Errors rate error.*
- `#define CAN_FRAMING_ERROR 8U`  
*Bus off condition reached.*
- `#define CAN_OVERFLOW_ERROR 16U`  
*Framing error of some kind on the CAN bus.*
- `#define CAN_OVERFLOW_ERROR 16U`  
*Overflow in receive queue.*

## CAN configuration options

- `#define CAN_USE_SLEEP_MODE TRUE`  
*Sleep mode related APIs inclusion switch.*

## Macro Functions

- `#define CAN_MAILBOX_TO_MASK(mbx) (1U << ((mbx) - 1U))`  
*Converts a mailbox index to a bit mask.*

## PLATFORM configuration options

- `#define PLATFORM_CAN_USE_CAN1 FALSE`  
*CAN1 driver enable switch.*

## Typedefs

- `typedef uint32_t canmbx_t`  
*Type of a transmission mailbox index.*

## Data Structures

- `struct CANTxFrame`  
*CAN transmission frame.*
- `struct CANRxFrame`  
*CAN received frame.*
- `struct CANConfig`  
*Driver configuration structure.*
- `struct CANDriver`  
*Structure representing an CAN driver.*

## Functions

- `void canInit (void)`  
*CAN Driver initialization.*
- `void canObjectInit (CANDriver *canc)`  
*Initializes the standard part of a `CANDriver` structure.*
- `void canStart (CANDriver *canc, const CANConfig *config)`  
*Configures and activates the CAN peripheral.*

- void `canStop (CANDriver *cancp)`  
*Deactivates the CAN peripheral.*
- bool `canTryTransmitl (CANDriver *cancp, canmbx_t mailbox, const CANTxFrame *ctfp)`  
*Can frame transmission attempt.*
- bool `canTryReceiveI (CANDriver *cancp, canmbx_t mailbox, CANRxFrame *crfp)`  
*Can frame receive attempt.*
- msg\_t `canTransmit (CANDriver *cancp, canmbx_t mailbox, const CANTxFrame *ctfp, systime_t timeout)`  
*Can frame transmission.*
- msg\_t `canReceive (CANDriver *cancp, canmbx_t mailbox, CANRxFrame *crfp, systime_t timeout)`  
*Can frame receive.*
- void `canSleep (CANDriver *cancp)`  
*Enters the sleep mode.*
- void `canWakeup (CANDriver *cancp)`  
*Enforces leaving the sleep mode.*
- void `can_lld_init (void)`  
*Low level CAN driver initialization.*
- void `can_lld_start (CANDriver *cancp)`  
*Configures and activates the CAN peripheral.*
- void `can_lld_stop (CANDriver *cancp)`  
*Deactivates the CAN peripheral.*
- bool `can_lld_is_tx_empty (CANDriver *cancp, canmbx_t mailbox)`  
*Determines whether a frame can be transmitted.*
- void `can_lld_transmit (CANDriver *cancp, canmbx_t mailbox, const CANTxFrame *ctfp)`  
*Inserts a frame into the transmit queue.*
- bool `can_lld_is_rx_nonempty (CANDriver *cancp, canmbx_t mailbox)`  
*Determines whether a frame has been received.*
- void `can_lld_receive (CANDriver *cancp, canmbx_t mailbox, CANRxFrame *crfp)`  
*Receives a frame from the input queue.*
- void `can_lld_sleep (CANDriver *cancp)`  
*Enters the sleep mode.*
- void `can_lld_wakeup (CANDriver *cancp)`  
*Enforces leaving the sleep mode.*

## Enumerations

- enum `canstate_t {`  
`CAN_UNINIT = 0, CAN_STOP = 1, CAN_STARTING = 2, CAN_READY = 3,`  
`CAN_SLEEP = 4 }`  
*Driver state machine possible states.*

## Variables

- CANDriver `CAND1`  
*CAN1 driver identifier.*

### 7.2.3 Macro Definition Documentation

#### 7.2.3.1 #define CAN\_LIMIT\_WARNING 1U

Errors rate warning.

**7.2.3.2 #define CAN\_LIMIT\_ERROR 2U**

Errors rate error.

**7.2.3.3 #define CAN\_BUS\_OFF\_ERROR 4U**

Bus off condition reached.

**7.2.3.4 #define CAN\_FRAMING\_ERROR 8U**

Framing error of some kind on the CAN bus.

**7.2.3.5 #define CAN\_OVERFLOW\_ERROR 16U**

Overflow in receive queue.

**7.2.3.6 #define CAN\_ANY\_MAILBOX 0**

Special mailbox identifier.

**7.2.3.7 #define CAN\_USE\_SLEEP\_MODE TRUE**

Sleep mode related APIs inclusion switch.

This option can only be enabled if the CAN implementation supports the sleep mode, see the macro `CAN_SUPPORTS_SLEEP` exported by the underlying implementation.

**7.2.3.8 #define CAN\_MAILBOX\_TO\_MASK( mbx )(1U << ((mbx) - 1U))**

Converts a mailbox index to a bit mask.

**7.2.3.9 #define CAN\_TX\_MAILBOXES 1**

Number of transmit mailboxes.

**7.2.3.10 #define CAN\_RX\_MAILBOXES 1**

Number of receive mailboxes.

**7.2.3.11 #define PLATFORM\_CAN\_USE\_CAN1 FALSE**

CAN1 driver enable switch.

If set to `TRUE` the support for CAN1 is included.

**Note**

The default is `FALSE`.

## 7.2.4 Typedef Documentation

### 7.2.4.1 `typedef uint32_t canmbx_t`

Type of a transmission mailbox index.

## 7.2.5 Enumeration Type Documentation

### 7.2.5.1 `enum canstate_t`

Driver state machine possible states.

#### Enumerator

**CAN\_UNINIT** Not initialized.

**CAN\_STOP** Stopped.

**CAN\_STARTING** Starting.

**CAN\_READY** Ready.

**CAN\_SLEEP** Sleep state.

## 7.2.6 Function Documentation

### 7.2.6.1 `void canInit( void )`

CAN Driver initialization.

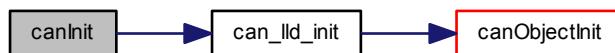
#### Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.2.6.2 `void canObjectInit( CANDriver * canp )`

Initializes the standard part of a `CANDriver` structure.

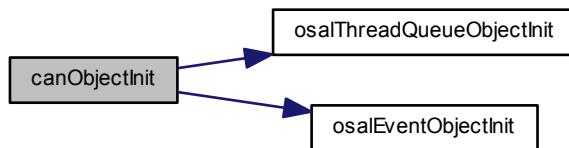
**Parameters**

out	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
-----	-------------	---

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

**7.2.6.3 void canStart( CANDriver \* *canp*, const CANConfig \* *config* )**

Configures and activates the CAN peripheral.

**Note**

Activating the CAN bus can be a slow operation.

Unlike other drivers it is not possible to restart the CAN driver without first stopping it using [canStop\(\)](#).

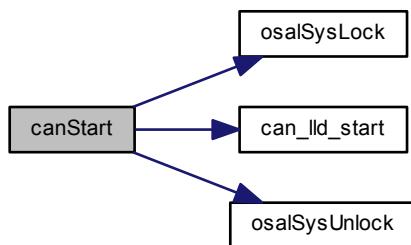
**Parameters**

in	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
in	<i>config</i>	pointer to the <a href="#">CANConfig</a> object. Depending on the implementation the value can be NULL.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.2.6.4 void canStop ( **CANDriver** \* *canp* )

Deactivates the CAN peripheral.

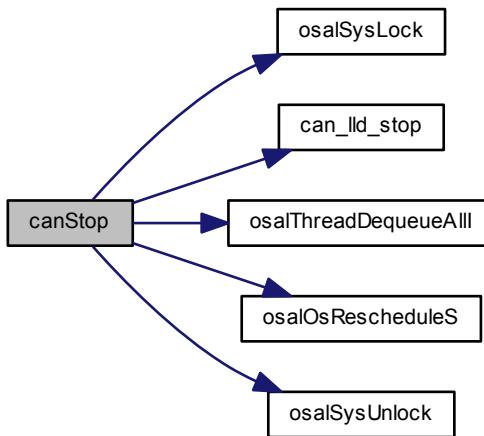
##### Parameters

in	<i>canp</i>	pointer to the <b>CANDriver</b> object
----	-------------	--

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.2.6.5 bool canTryTransmit( **CANDriver** \* *canp*, **canmbx\_t** *mailbox*, const **CANTxFFrame** \* *ctfp* )

Can frame transmission attempt.

The specified frame is queued for transmission, if the hardware queue is full then the function fails.

##### Parameters

in	<i>canp</i>	pointer to the <b>CANDriver</b> object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox
in	<i>ctfp</i>	pointer to the CAN frame to be transmitted

##### Returns

The operation result.

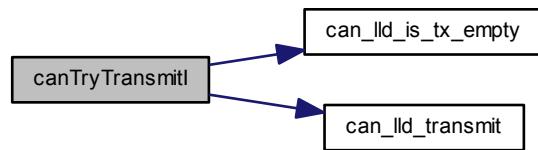
##### Return values

<i>false</i>	Frame transmitted.
<i>true</i>	Mailbox full.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.2.6.6 bool canTryReceive( CANDriver \* *canp*, canmbx\_t *mailbox*, CANRxFrame \* *crfp* )

Can frame receive attempt.

The function tries to fetch a frame from a mailbox.

**Parameters**

in	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox
out	<i>crfp</i>	pointer to the buffer where the CAN frame is copied

**Returns**

The operation result.

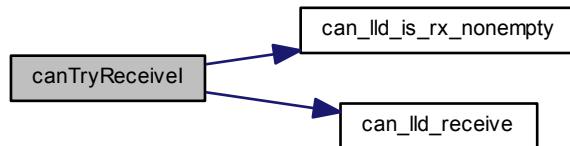
**Return values**

<i>false</i>	Frame fetched.
<i>true</i>	Mailbox empty.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.2.6.7 `msg_t canTransmit ( CANDriver * canp, canmbx_t mailbox, const CANTxFr ame * ctfp, systime_t timeout )`

Can frame transmission.

The specified frame is queued for transmission, if the hardware queue is full then the invoking thread is queued.

##### Note

Trying to transmit while in sleep mode simply enqueues the thread.

##### Parameters

in	<i>canp</i>	pointer to the <code>CANDriver</code> object
in	<i>mailbox</i>	mailbox number, <code>CAN_ANY_MAILBOX</code> for any mailbox
in	<i>ctfp</i>	pointer to the CAN frame to be transmitted
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

##### Returns

The operation result.

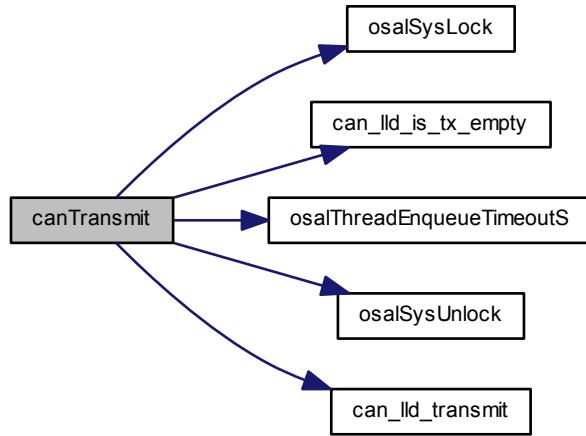
##### Return values

<code>MSG_OK</code>	the frame has been queued for transmission.
<code>MSG_TIMEOUT</code>	The operation has timed out.
<code>MSG_RESET</code>	The driver has been stopped while waiting.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.2.6.8 `msg_t canReceive ( CANDriver * canp, canmbx_t mailbox, CANRxFrame * crfp, systime_t timeout )`

Can frame receive.

The function waits until a frame is received.

#### Note

Trying to receive while in sleep mode simply enqueues the thread.

#### Parameters

in	<code>canp</code>	pointer to the <code>CANDriver</code> object
in	<code>mailbox</code>	mailbox number, <code>CAN_ANY_MAILBOX</code> for any mailbox
out	<code>crfp</code>	pointer to the buffer where the CAN frame is copied
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout (useful in an event driven scenario where a thread never blocks for I/O).</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

#### Returns

The operation result.

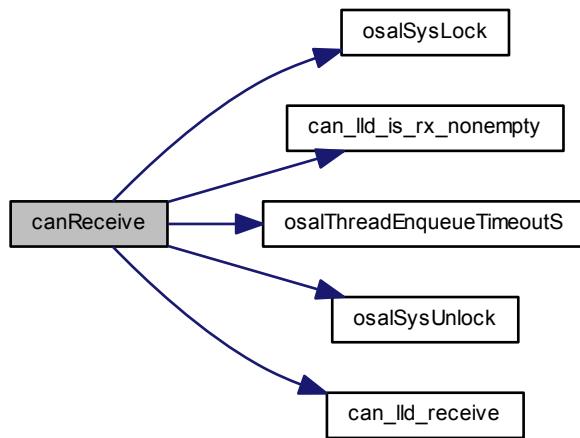
#### Return values

<code>MSG_OK</code>	a frame has been received and placed in the buffer.
<code>MSG_TIMEOUT</code>	The operation has timed out.
<code>MSG_RESET</code>	The driver has been stopped while waiting.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.2.6.9 void canSleep ( CANDriver \* *canp* )

Enters the sleep mode.

This function puts the CAN driver in sleep mode and broadcasts the `sleep_event` event source.

**Precondition**

In order to use this function the option `CAN_USE_SLEEP_MODE` must be enabled and the `CAN_SUPPORTS_SLEEP` mode must be supported by the low level driver.

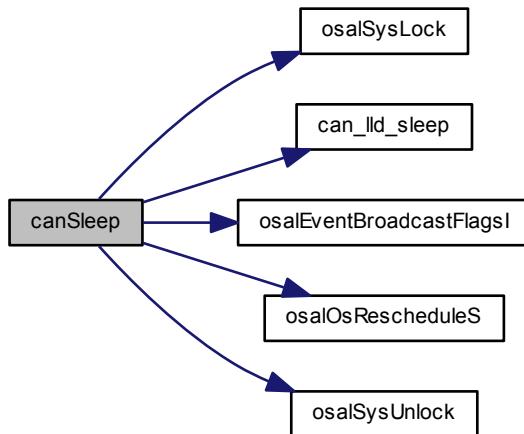
**Parameters**

in	<i>canp</i>	pointer to the <code>CANDriver</code> object
----	-------------	--

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.2.6.10 void canWakeup ( `CANDriver` \* *canp* )

Enforces leaving the sleep mode.

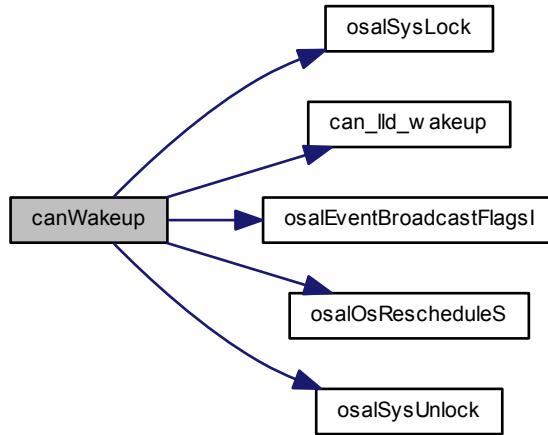
##### Note

The sleep mode is supposed to be usually exited automatically by an hardware event.

##### Parameters

in	<i>canp</i>	pointer to the <code>CANDriver</code> object
----	-------------	--

Here is the call graph for this function:



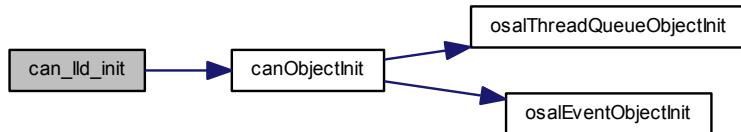
#### 7.2.6.11 void canIld\_init ( void )

Low level CAN driver initialization.

##### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.2.6.12 void canIld\_start ( CANDriver \* canp )

Configures and activates the CAN peripheral.

##### Parameters

in	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.2.6.13 void can\_lld\_stop ( CANDriver \* *canp* )**

Deactivates the CAN peripheral.

**Parameters**

in	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.2.6.14 bool can\_lld\_is\_tx\_empty ( CANDriver \* *canp*, canmbx\_t *mailbox* )**

Determines whether a frame can be transmitted.

**Parameters**

in	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox

**Returns**

The queue space availability.

**Return values**

<i>FALSE</i>	no space in the transmit queue.
<i>TRUE</i>	transmit slot available.

**Function Class:**

Not an API, this function is for internal use only.

**7.2.6.15 void can\_lld\_transmit ( CANDriver \* *canp*, canmbx\_t *mailbox*, const CANTxFrame \* *ctfp* )**

Inserts a frame into the transmit queue.

**Parameters**

in	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
in	<i>ctfp</i>	pointer to the CAN frame to be transmitted
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox

**Function Class:**

Not an API, this function is for internal use only.

**7.2.6.16 bool can\_lld\_is\_rx\_nonempty ( CANDriver \* *canp*, canmbx\_t *mailbox* )**

Determines whether a frame has been received.

**Parameters**

in	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox

**Returns**

The queue space availability.

**Return values**

<i>FALSE</i>	no space in the transmit queue.
<i>TRUE</i>	transmit slot available.

**Function Class:**

Not an API, this function is for internal use only.

**7.2.6.17 void can\_lld\_receive ( CANDriver \* *canp*, canmbx\_t *mailbox*, CANRxFrame \* *crfp* )**

Receives a frame from the input queue.

**Parameters**

in	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox
out	<i>crfp</i>	pointer to the buffer where the CAN frame is copied

**Function Class:**

Not an API, this function is for internal use only.

**7.2.6.18 void can\_lld\_sleep ( CANDriver \* *canp* )**

Enters the sleep mode.

**Parameters**

in	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.2.6.19 void can\_lld\_wakeup ( CANDriver \* *canp* )**

Enforces leaving the sleep mode.

**Parameters**

in	<i>canp</i>	pointer to the <a href="#">CANDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

## 7.2.7 Variable Documentation

### 7.2.7.1 CANDriver CAND1

CAN1 driver identifier.

## 7.3 DAC Driver

Generic DAC Driver.

### 7.3.1 Detailed Description

Generic DAC Driver.

This module implements a generic DAC (Digital to Analog Converter) driver.

#### Precondition

In order to use the MAC driver the `HAL_USE_DAC` option must be enabled in `halconf.h`.

#### Macros

- `#define DAC_MAX_CHANNELS 2`  
*Maximum number of DAC channels per unit.*

#### DAC configuration options

- `#define DAC_USE_WAIT TRUE`  
*Enables synchronous APIs.*
- `#define DAC_USE_MUTUAL_EXCLUSION TRUE`  
*Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.*

#### Low level driver helper macros

- `#define _dac_wait_s(dacp) osalThreadSuspendS(&(dacp)->thread)`  
*Waits for operation completion.*
- `#define _dac_reset_i(dacp) osalThreadResumeI(&(dacp)->thread, MSG_RESET)`  
*Resumes a thread waiting for a conversion completion.*
- `#define _dac_reset_s(dacp) osalThreadResumeS(&(dacp)->thread, MSG_RESET)`  
*Resumes a thread waiting for a conversion completion.*
- `#define _dac_wakeup_isr(dacp)`  
*Wakes up the waiting thread.*
- `#define _dac_timeout_isr(dacp)`  
*Wakes up the waiting thread with a timeout message.*
- `#define _dac_isr_half_code(dacp)`  
*Common ISR code, half buffer event.*
- `#define _dac_isr_full_code(dacp)`  
*Common ISR code, full buffer event.*
- `#define _dac_isr_error_code(dacp, err)`  
*Common ISR code, error event.*

#### Configuration options

- `#define PLATFORM_DAC_USE_DAC1 FALSE`  
*DAC1 CH1 driver enable switch.*

## Typedefs

- `typedef uint32_t dacchannel_t`  
*Type of a DAC channel index.*
- `typedef struct DACDriver DACDriver`  
*Type of a structure representing an DAC driver.*
- `typedef uint16_t dacsample_t`  
*Type representing a DAC sample.*
- `typedef void(* daccallback_t) (DACDriver *dapc, const dacsample_t *buffer, size_t n)`  
*DAC notification callback type.*
- `typedef void(* dacerrorcallback_t) (DACDriver *dapc, dacerror_t err)`  
*ADC error callback type.*

## Data Structures

- `struct DACConversionGroup`  
*DAC Conversion group structure.*
- `struct DACConfig`  
*Driver configuration structure.*
- `struct DACDriver`  
*Structure representing a DAC driver.*

## Functions

- `void dacInit (void)`  
*DAC Driver initialization.*
- `void dacObjectInit (DACDriver *dapc)`  
*Initializes the standard part of a `DACDriver` structure.*
- `void dacStart (DACDriver *dapc, const DACConfig *config)`  
*Configures and activates the DAC peripheral.*
- `void dacStop (DACDriver *dapc)`  
*Deactivates the DAC peripheral.*
- `void dacPutChannelX (DACDriver *dapc, dacchannel_t channel, dacsample_t sample)`  
*Outputs a value directly on a DAC channel.*
- `void dacStartConversion (DACDriver *dapc, const DACConversionGroup *grpp, const dacsample_t *samples, size_t depth)`  
*Starts a DAC conversion.*
- `void dacStartConversionl (DACDriver *dapc, const DACConversionGroup *grpp, const dacsample_t *samples, size_t depth)`  
*Starts a DAC conversion.*
- `void dacStopConversion (DACDriver *dapc)`  
*Stops an ongoing conversion.*
- `void dacStopConversionl (DACDriver *dapc)`  
*Stops an ongoing conversion.*
- `msg_t dacConvert (DACDriver *dapc, const DACConversionGroup *grpp, const dacsample_t *samples, size_t depth)`  
*Performs a DAC conversion.*
- `void dacAcquireBus (DACDriver *dapc)`  
*Gains exclusive access to the DAC bus.*
- `void dacReleaseBus (DACDriver *dapc)`  
*Releases exclusive access to the DAC bus.*

- void `dac_lld_init` (void)  
*Low level DAC driver initialization.*
- void `dac_lld_start` (DACDriver \*dacp)  
*Configures and activates the DAC peripheral.*
- void `dac_lld_stop` (DACDriver \*dacp)  
*Deactivates the DAC peripheral.*
- void `dac_lld_put_channel` (DACDriver \*dacp, dacchannel\_t channel, dacsample\_t sample)  
*Outputs a value directly on a DAC channel.*
- void `dac_lld_start_conversion` (DACDriver \*dacp)  
*Starts a DAC conversion.*
- void `dac_lld_stop_conversion` (DACDriver \*dacp)  
*Stops an ongoing conversion.*

## Enumerations

- enum `dacstate_t` {  
 DAC\_UNINIT = 0, DAC\_STOP = 1, DAC\_READY = 2, DAC\_ACTIVE = 3,  
 DAC\_COMPLETE = 4, DAC\_ERROR = 5 }  
*Driver state machine possible states.*
- enum `dacerror_t` { DAC\_ERR\_DMAFAILURE = 0, DAC\_ERR\_UNDERFLOW = 1 }  
*Possible DAC failure causes.*

## Variables

- DACDriver `DACD1`  
*DAC1 driver identifier.*

### 7.3.2 Macro Definition Documentation

#### 7.3.2.1 #define DAC\_USE\_WAIT TRUE

Enables synchronous APIs.

##### Note

Disabling this option saves both code and data space.

#### 7.3.2.2 #define DAC\_USE\_MUTUAL\_EXCLUSION TRUE

Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

##### Note

Disabling this option saves both code and data space.

#### 7.3.2.3 #define \_dac\_wait\_s( dacp ) osalThreadSuspendS(&(dacp)->thread)

Waits for operation completion.

This function waits for the driver to complete the current operation.

**Precondition**

An operation must be running while the function is invoked.

**Note**

No more than one thread can wait on a DAC driver using this function.

**Parameters**

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.3.2.4 #define \_dac\_reset\_i( *dacp* ) osalThreadResumeI(&(*dacp*)>thread, MSG\_RESET)**

Resumes a thread waiting for a conversion completion.

**Parameters**

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.3.2.5 #define \_dac\_reset\_s( *dacp* ) osalThreadResumeS(&(*dacp*)>thread, MSG\_RESET)**

Resumes a thread waiting for a conversion completion.

**Parameters**

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.3.2.6 #define \_dac\_wakeup\_isr( *dacp* )****Value:**

```
{
  osalSysLockFromISR();
  \
  osalThreadResumeI(&(dacp)>thread, MSG_OK);
  \
  osalSysUnlockFromISR();
}
```

Wakes up the waiting thread.

**Parameters**

in	dacp	pointer to the <a href="#">DACDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.3.2.7 #define \_dac\_timeout\_isr( dacp )****Value:**

```
{
    osalSysLockFromISR(); \
    osalThreadResumeI(&(dacp)->thread, MSG_TIMEOUT); \
    osalSysUnlockFromISR(); \
}
```

Wakes up the waiting thread with a timeout message.

**Parameters**

in	dacp	pointer to the <a href="#">DACDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.3.2.8 #define \_dac\_isr\_half\_code( dacp )****Value:**

```
{
    if ((dacp)->grpp->end_cb != NULL) { \
        (dacp)->grpp->end_cb(dacp, (dacp)->samples, (dacp)->depth / 2); \
    } \
}
```

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	dacp	pointer to the <a href="#">DACDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

### 7.3.2.9 #define \_dac\_isr\_full\_code( *dacp* )

#### Value:

```
{
    if ((dacp)->grpp->end_cb != NULL) {
        if ((dacp)->depth > 1) {
            /* Invokes the callback passing the 2nd half of the buffer.*/
            size_t half = (dacp)->depth / 2;
            size_t half_index = half * (dacp)->grpp->num_channels;
            (dacp)->grpp->end_cb(dacp, (dacp)->samples + half_index, half);
        }
        else {
            /* Invokes the callback passing the whole buffer.*/
            (dacp)->grpp->end_cb(dacp, (dacp)->samples, (dacp)->depth);
        }
    }
}
```

Common ISR code, full buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

#### Note

This macro is meant to be used in the low level drivers implementation only.

#### Parameters

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
----	-------------	---

#### Function Class:

Not an API, this function is for internal use only.

### 7.3.2.10 #define \_dac\_isr\_error\_code( *dacp, err* )

#### Value:

```
{
    dac_lld_stop_conversion(dacp);
    if ((dacp)->grpp->error_cb != NULL) {
        (dacp)->state = DAC_ERROR;
        (dacp)->grpp->error_cb(dacp, err);
        if ((dacp)->state == DAC_ERROR)
            (dacp)->state = DAC_READY;
    }
    (dacp)->grpp = NULL;
    \_dac\_timeout\_isr(dacp);
}
```

Common ISR code, error event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread timeout signaling, if any.
- Driver state transitions.

#### Note

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
in	<i>err</i>	platform dependent error code

**Function Class:**

Not an API, this function is for internal use only.

**7.3.2.11 #define DAC\_MAX\_CHANNELS 2**

Maximum number of DAC channels per unit.

**7.3.2.12 #define PLATFORM\_DAC\_USE\_DAC1 FALSE**

DAC1 CH1 driver enable switch.

If set to TRUE the support for DAC1 channel 1 is included.

**Note**

The default is FALSE.

**7.3.3 Typedef Documentation****7.3.3.1 typedef uint32\_t dacchannel\_t**

Type of a DAC channel index.

**7.3.3.2 typedef struct DACDriver DACDriver**

Type of a structure representing an DAC driver.

**7.3.3.3 typedef uint16\_t dacsample\_t**

Type representing a DAC sample.

**7.3.3.4 typedef void(\* daccallback\_t) (DACDriver \*dacp, const dacsample\_t \*buffer, size\_t n)**

DAC notification callback type.

**Parameters**

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object triggering the
in	<i>buffer</i>	pointer to the next semi-buffer to be filled
in	<i>n</i>	number of buffer rows available starting from <i>buffer</i> callback

**7.3.3.5 typedef void(\* dacerrorcallback\_t) (DACDriver \*dacp, dacerror\_t err)**

ADC error callback type.

### Parameters

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object triggering the callback
in	<i>err</i>	ADC error code

## 7.3.4 Enumeration Type Documentation

### 7.3.4.1 enum dacstate\_t

Driver state machine possible states.

#### Enumerator

- DAC\_UNINIT** Not initialized.
- DAC\_STOP** Stopped.
- DAC\_READY** Ready.
- DAC\_ACTIVE** Exchanging data.
- DAC\_COMPLETE** Asynchronous operation complete.
- DAC\_ERROR** Error.

### 7.3.4.2 enum dacerror\_t

Possible DAC failure causes.

#### Note

Error codes are architecture dependent and should not relied upon.

#### Enumerator

- DAC\_ERR\_DMAFAILURE** DMA operations failure.
- DAC\_ERR\_UNDERFLOW** DAC overflow condition.

## 7.3.5 Function Documentation

### 7.3.5.1 void dacInit( void )

DAC Driver initialization.

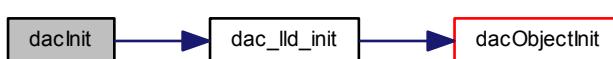
#### Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.3.5.2 void dacObjectInit ( **DACDriver** \* *dacp* )

Initializes the standard part of a **DACDriver** structure.

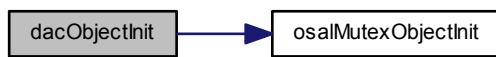
#### Parameters

out	<i>dacp</i>	pointer to the <b>DACDriver</b> object
-----	-------------	--

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.3.5.3 void dacStart ( **DACDriver** \* *dacp*, const **DACConfig** \* *config* )

Configures and activates the DAC peripheral.

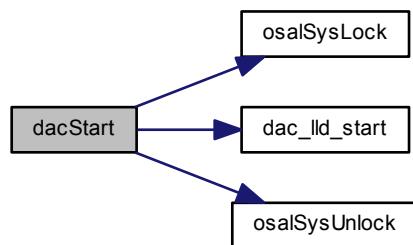
#### Parameters

in	<i>dacp</i>	pointer to the <b>DACDriver</b> object
in	<i>config</i>	pointer to the <b>DACConfig</b> object, it can be NULL if the low level driver implementation supports a default configuration

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.3.5.4 void dacStop ( DACDriver \* *dacp* )

Deactivates the DAC peripheral.

#### Note

Deactivating the peripheral also enforces a release of the slave select line.

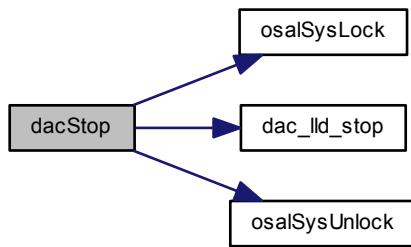
#### Parameters

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
----	-------------	---

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.3.5.5 void dacPutChannelX ( DACDriver \* *dacp*, dacchannel\_t *channel*, dacsample\_t *sample* )

Outputs a value directly on a DAC channel.

#### Parameters

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
in	<i>channel</i>	DAC channel number
in	<i>sample</i>	value to be output

#### Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:



```
7.3.5.6 void dacStartConversion ( DACDriver * dacp, const DACConversionGroup * grpp, const dacsample_t * samples, size_t depth )
```

Starts a DAC conversion.

Starts an asynchronous conversion operation.

#### Note

The buffer is organized as a matrix of M\*N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

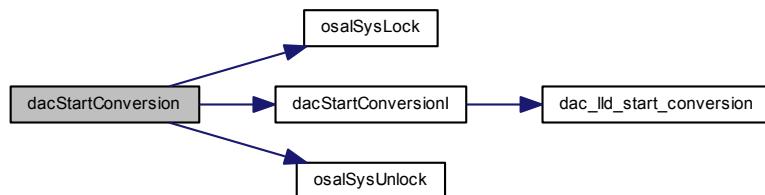
#### Parameters

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
in	<i>grpp</i>	pointer to a <a href="#">DACConversionGroup</a> object
in	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



```
7.3.5.7 void dacStartConversionI ( DACDriver * dacp, const DACConversionGroup * grpp, const dacsample_t * samples, size_t depth )
```

Starts a DAC conversion.

Starts an asynchronous conversion operation.

#### Postcondition

The callbacks associated to the conversion group will be invoked on buffer fill and error events.

#### Note

The buffer is organized as a matrix of M\*N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

#### Parameters

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
in	<i>grpp</i>	pointer to a <a href="#">DACConversionGroup</a> object
in	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.3.5.8 void dacStopConversion ( DACDriver \* *dacp* )

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the `DAC_READY` state. If there was no conversion being processed then the function does nothing.

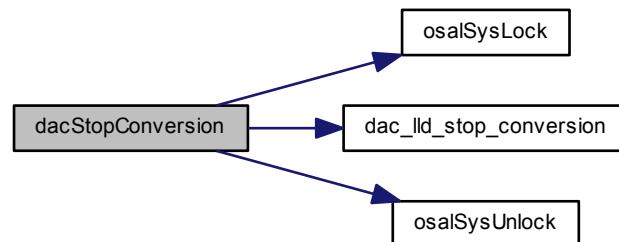
#### Parameters

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
----	-------------	---

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.3.5.9 void dacStopConversion( **DACDriver** \* *dacp* )

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the **DAC\_READY** state. If there was no conversion being processed then the function does nothing.

#### Parameters

in	<i>dacp</i>	pointer to the <b>DACDriver</b> object
----	-------------	--

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.3.5.10 msg\_t dacConvert( **DACDriver** \* *dacp*, const **DACConversionGroup** \* *grpp*, const **dacsample\_t** \* *samples*, size\_t *depth* )

Performs a DAC conversion.

Performs a synchronous conversion operation.

#### Note

The buffer is organized as a matrix of M\*N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

#### Parameters

in	<i>dacp</i>	pointer to the <b>DACDriver</b> object
in	<i>grpp</i>	pointer to a <b>DACConversionGroup</b> object
out	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

#### Returns

The operation result.

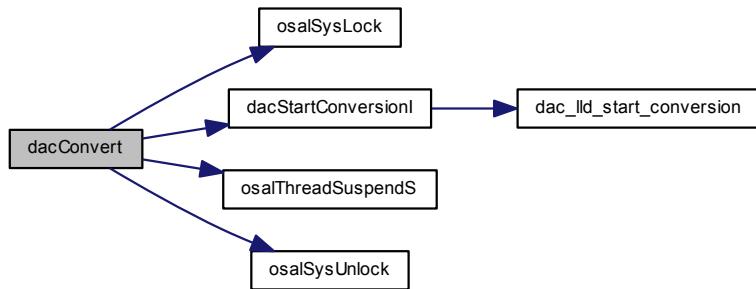
#### Return values

<i>MSG_OK</i>	Conversion finished.
<i>MSG_RESET</i>	The conversion has been stopped using <code>acdStopConversion()</code> or <code>acdStopConversionI()</code> , the result buffer may contain incorrect data.
<i>MSG_TIMEOUT</i>	The conversion has been stopped because an hardware error.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.3.5.11 void dacAcquireBus ( DACDriver \* *dacp* )**

Gains exclusive access to the DAC bus.

This function tries to gain ownership to the DAC bus, if the bus is already being used then the invoking thread is queued.

**Precondition**

In order to use this function the option `DAC_USE_MUTUAL_EXCLUSION` must be enabled.

**Parameters**

in	<i>dacp</i>	pointer to the <code>DACDriver</code> object
----	-------------	--

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



**7.3.5.12 void dacReleaseBus ( DACDriver \* *dacp* )**

Releases exclusive access to the DAC bus.

**Precondition**

In order to use this function the option `DAC_USE_MUTUAL_EXCLUSION` must be enabled.

**Parameters**

in	<i>dacp</i>	pointer to the <code>DACDriver</code> object
----	-------------	--

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

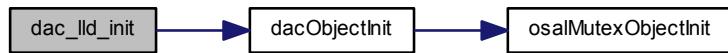
**7.3.5.13 void dac\_lld\_init ( void )**

Low level DAC driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.3.5.14 void dac\_lld\_start ( DACDriver \* *dacp* )**

Configures and activates the DAC peripheral.

**Parameters**

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.3.5.15 void dac\_lld\_stop ( DACDriver \* *dacp* )**

Deactivates the DAC peripheral.

**Parameters**

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.3.5.16 void dac\_lld\_put\_channel ( DACDriver \* *dacp*, dacchannel\_t *channel*, dacsample\_t *sample* )**

Outputs a value directly on a DAC channel.

**Parameters**

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
in	<i>channel</i>	DAC channel number
in	<i>sample</i>	value to be output

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.3.5.17 void dac\_lld\_start\_conversion ( DACDriver \* *dacp* )**

Starts a DAC conversion.

Starts an asynchronous conversion operation.

**Note**

In `DAC_DHRM_8BIT_RIGHT` mode the parameters passed to the callback are wrong because two samples are packed in a single `dacsample_t` element. This will not be corrected, do not rely on those parameters.

In `DAC_DHRM_8BIT_RIGHT_DUAL` mode two samples are treated as a single 16 bits sample and packed into a single `dacsample_t` element. The `num_channels` must be set to one in the group conversion configuration structure.

**Parameters**

in	<i>dacp</i>	pointer to the <a href="#">DACDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.3.5.18 void dac\_ll\_stop\_conversion ( DACDriver \* *dacp* )**

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the `DAC_READY` state. If there was no conversion being processed then the function does nothing.

**Parameters**

in	<i>dacp</i>	pointer to the <code>DACDriver</code> object
----	-------------	--

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

## 7.3.6 Variable Documentation

### 7.3.6.1 DACDriver DACD1

DAC1 driver identifier.

## 7.4 EXT Driver

Generic EXT Driver.

### 7.4.1 Detailed Description

Generic EXT Driver.

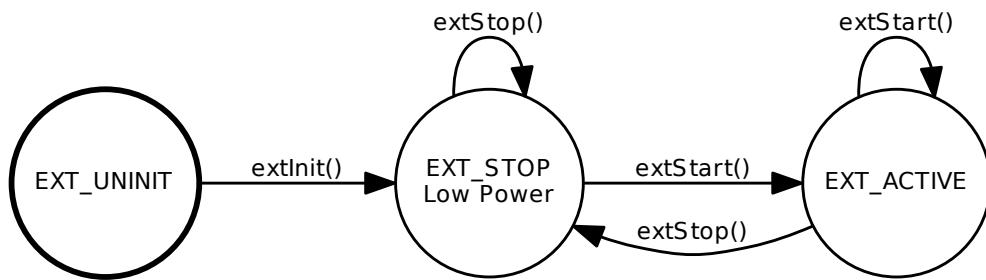
This module implements a generic EXT (EXTernal) driver.

#### Precondition

In order to use the EXT driver the `HAL_USE_EXT` option must be enabled in `halconf.h`.

### 7.4.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



### 7.4.3 EXT Operations.

This driver abstracts generic external interrupt sources, a callback is invoked when a programmable transition is detected on one of the configured channels. Several channel modes are possible.

- **`EXT_CH_MODE_DISABLED`**, channel not used.
- **`EXT_CH_MODE_RISING_EDGE`**, callback on a rising edge.
- **`EXT_CH_MODE_FALLING_EDGE`**, callback on a falling edge.
- **`EXT_CH_MODE_BOTH_EDGES`**, callback on a both edges.

#### Macros

- `#define EXT_MAX_CHANNELS 20`

*Available number of EXT channels.*

## EXT channel modes

- `#define EXT_CH_MODE_EDGES_MASK 3U`  
*Mask of edges field.*
- `#define EXT_CH_MODE_DISABLED 0U`  
*Channel disabled.*
- `#define EXT_CH_MODE_RISING_EDGE 1U`  
*Rising edge callback.*
- `#define EXT_CH_MODE_FALLING_EDGE 2U`  
*Falling edge callback.*
- `#define EXT_CH_MODE_BOTH_EDGES 3U`  
*Both edges callback.*
- `#define EXT_CH_MODE_AUTOSTART 4U`  
*Channel started automatically on driver start.*

## Macro Functions

- `#define extChannelEnable(extp, channel) ext_lld_channel_enable(extp, channel)`  
*Enables an EXT channel.*
- `#define extChannelDisable(extp, channel) ext_lld_channel_disable(extp, channel)`  
*Disables an EXT channel.*
- `#define extSetChannelMode(extp, channel, extcp)`  
*Changes the operation mode of a channel.*

## PLATFORM configuration options

- `#define PLATFORM_EXT_USE_EXT1 FALSE`  
*EXT driver enable switch.*

## Typedefs

- `typedef struct EXTDriver EXTDriver`  
*Type of a structure representing a EXT driver.*
- `typedef uint32_t expchannel_t`  
*EXT channel identifier.*
- `typedef void(* extcallback_t) (EXTDriver *extp, expchannel_t channel)`  
*Type of an EXT generic notification callback.*

## Data Structures

- `struct EXTChannelConfig`  
*Channel configuration structure.*
- `struct EXTConfig`  
*Driver configuration structure.*
- `struct EXTDriver`  
*Structure representing an EXT driver.*

## Functions

- void `extInit` (void)
 

*EXT Driver initialization.*
- void `extObjectInit` (EXTDriver \*extp)
 

*Initializes the standard part of a `EXTDriver` structure.*
- void `extStart` (EXTDriver \*extp, const EXTConfig \*config)
 

*Configures and activates the EXT peripheral.*
- void `extStop` (EXTDriver \*extp)
 

*Deactivates the EXT peripheral.*
- void `extChannelEnable` (EXTDriver \*extp, expchannel\_t channel)
 

*Enables an EXT channel.*
- void `extChannelDisable` (EXTDriver \*extp, expchannel\_t channel)
 

*Disables an EXT channel.*
- void `extSetChannelModel` (EXTDriver \*extp, expchannel\_t channel, const EXTChannelConfig \*extcp)
 

*Changes the operation mode of a channel.*
- void `ext_lld_init` (void)
 

*Low level EXT driver initialization.*
- void `ext_lld_start` (EXTDriver \*extp)
 

*Configures and activates the EXT peripheral.*
- void `ext_lld_stop` (EXTDriver \*extp)
 

*Deactivates the EXT peripheral.*
- void `ext_lld_channel_enable` (EXTDriver \*extp, expchannel\_t channel)
 

*Enables an EXT channel.*
- void `ext_lld_channel_disable` (EXTDriver \*extp, expchannel\_t channel)
 

*Disables an EXT channel.*

## Enumerations

- enum `extstate_t` { `EXT_UNINIT` = 0, `EXT_STOP` = 1, `EXT_ACTIVE` = 2 }
 

*Driver state machine possible states.*

## Variables

- EXTDriver `EXTD1`

*EXT1 driver identifier.*

### 7.4.4 Macro Definition Documentation

#### 7.4.4.1 #define EXT\_CH\_MODE\_EDGES\_MASK 3U

Mask of edges field.

#### 7.4.4.2 #define EXT\_CH\_MODE\_DISABLED 0U

Channel disabled.

#### 7.4.4.3 #define EXT\_CH\_MODE\_RISING\_EDGE 1U

Rising edge callback.

## 7.4.4.4 #define EXT\_CH\_MODE\_FALLING\_EDGE 2U

Falling edge callback.

## 7.4.4.5 #define EXT\_CH\_MODE\_BOTH\_EDGES 3U

Both edges callback.

## 7.4.4.6 #define EXT\_CH\_MODE\_AUTOSTART 4U

Channel started automatically on driver start.

## 7.4.4.7 #define extChannelEnable( extp, channel ) ext\_llid\_channel\_enable(extp, channel)

Enables an EXT channel.

## Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be enabled

## Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

## 7.4.4.8 #define extChannelDisable( extp, channel ) ext\_llid\_channel\_disable(extp, channel)

Disables an EXT channel.

## Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be disabled

## Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

## 7.4.4.9 #define extSetChannelMode( extp, channel, extcp )

## Value:

```
{
    osalSysLock();
    \
    extSetChannelModeI(extp, channel, extcp);
    \
    osalSysUnlock();
}
```

Changes the operation mode of a channel.

## Note

This function attempts to write over the current configuration structure that must have been not declared constant. This violates the `const` qualifier in `extStart()` but it is intentional. This function cannot be used if the configuration structure is declared `const`.

**Parameters**

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be changed
in	<i>extcp</i>	new configuration for the channel

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.4.4.10 #define EXT\_MAX\_CHANNELS 20**

Available number of EXT channels.

**7.4.4.11 #define PLATFORM\_EXT\_USE\_EXT1 FALSE**

EXT driver enable switch.

If set to `TRUE` the support for EXT1 is included.

**Note**

The default is `FALSE`.

**7.4.5 Typedef Documentation****7.4.5.1 typedef struct EXTDriver EXTDriver**

Type of a structure representing a EXT driver.

**7.4.5.2 typedef uint32\_t expchannel\_t**

EXT channel identifier.

**7.4.5.3 typedef void(\* extcallback\_t)(EXTDriver \*extp, expchannel\_t channel)**

Type of an EXT generic notification callback.

**Parameters**

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object triggering the callback
----	-------------	--

**7.4.6 Enumeration Type Documentation****7.4.6.1 enum extstate\_t**

Driver state machine possible states.

**Enumerator**

**`EXT_UNINIT`** Not initialized.

**`EXT_STOP`** Stopped.

**`EXT_ACTIVE`** Active.

### 7.4.7 Function Documentation

#### 7.4.7.1 void extInit( void )

EXT Driver initialization.

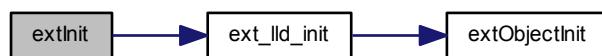
##### Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.4.7.2 void extObjectInit( EXTDriver \* extp )

Initializes the standard part of a `EXTDriver` structure.

##### Parameters

out	<code>extp</code>	pointer to the <code>EXTDriver</code> object
-----	-------------------	--

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

#### 7.4.7.3 void extStart( EXTDriver \* extp, const EXTConfig \* config )

Configures and activates the EXT peripheral.

##### Postcondition

After activation all EXT channels are in the disabled state, use `extChannelEnable()` in order to activate them.

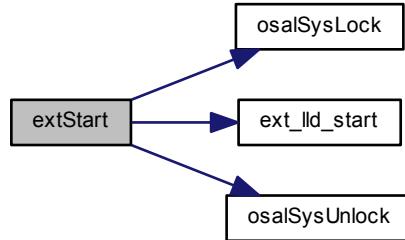
##### Parameters

in	<code>extp</code>	pointer to the <code>EXTDriver</code> object
in	<code>config</code>	pointer to the <code>EXTConfig</code> object

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.4.7.4 void extStop ( EXTDriver \* extp )

Deactivates the EXT peripheral.

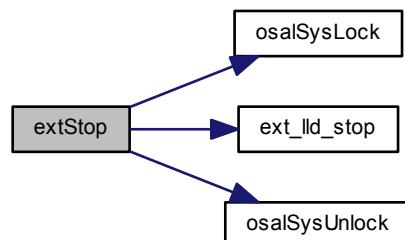
##### Parameters

in	<code>extp</code>	pointer to the <code>EXTDriver</code> object
----	-------------------	--

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.4.7.5 void extChannelEnable ( EXTDriver \* extp, expchannel\_t channel )

Enables an EXT channel.

##### Precondition

The channel must not be in `EXT_CH_MODE_DISABLED` mode.

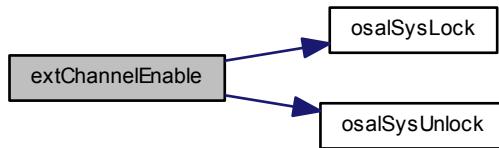
#### Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be enabled

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.4.7.6 void extChannelDisable ( `EXTDriver` \* *extp*, `expchannel_t` *channel* )

Disables an EXT channel.

#### Precondition

The channel must not be in `EXT_CH_MODE_DISABLED` mode.

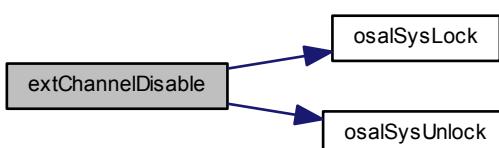
#### Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be disabled

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.4.7.7 void extSetChannelModel ( EXTDriver \* extp, expchannel\_t channel, const EXTChannelConfig \* extcp )

Changes the operation mode of a channel.

##### Note

This function attempts to write over the current configuration structure that must have been not declared constant. This violates the `const` qualifier in `extStart()` but it is intentional.

This function cannot be used if the configuration structure is declared `const`.

The effect of this function on constant configuration structures is not defined.

##### Parameters

in	<code>extp</code>	pointer to the <code>EXTDriver</code> object
in	<code>channel</code>	channel to be changed
in	<code>extcp</code>	new configuration for the channel

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.4.7.8 void ext\_lld\_init ( void )

Low level EXT driver initialization.

##### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.4.7.9 void ext\_lld\_start ( EXTDriver \* extp )

Configures and activates the EXT peripheral.

## Parameters

in	<i>extp</i>	pointer to the <a href="#">EXTDriver</a> object
----	-------------	---

## Function Class:

Not an API, this function is for internal use only.

7.4.7.10 void ext\_lld\_stop ( [EXTDriver](#) \* *extp* )

Deactivates the EXT peripheral.

## Parameters

in	<i>extp</i>	pointer to the <a href="#">EXTDriver</a> object
----	-------------	---

## Function Class:

Not an API, this function is for internal use only.

7.4.7.11 void ext\_lld\_channel\_enable ( [EXTDriver](#) \* *extp*, [expchannel\\_t](#) *channel* )

Enables an EXT channel.

## Parameters

in	<i>extp</i>	pointer to the <a href="#">EXTDriver</a> object
in	<i>channel</i>	channel to be enabled

## Function Class:

Not an API, this function is for internal use only.

7.4.7.12 void ext\_lld\_channel\_disable ( [EXTDriver](#) \* *extp*, [expchannel\\_t](#) *channel* )

Disables an EXT channel.

## Parameters

in	<i>extp</i>	pointer to the <a href="#">EXTDriver</a> object
in	<i>channel</i>	channel to be disabled

## Function Class:

Not an API, this function is for internal use only.

## 7.4.8 Variable Documentation

7.4.8.1 [EXTDriver](#) EXTD1

EXT1 driver identifier.

## 7.5 GPT Driver

Generic GPT Driver.

### 7.5.1 Detailed Description

Generic GPT Driver.

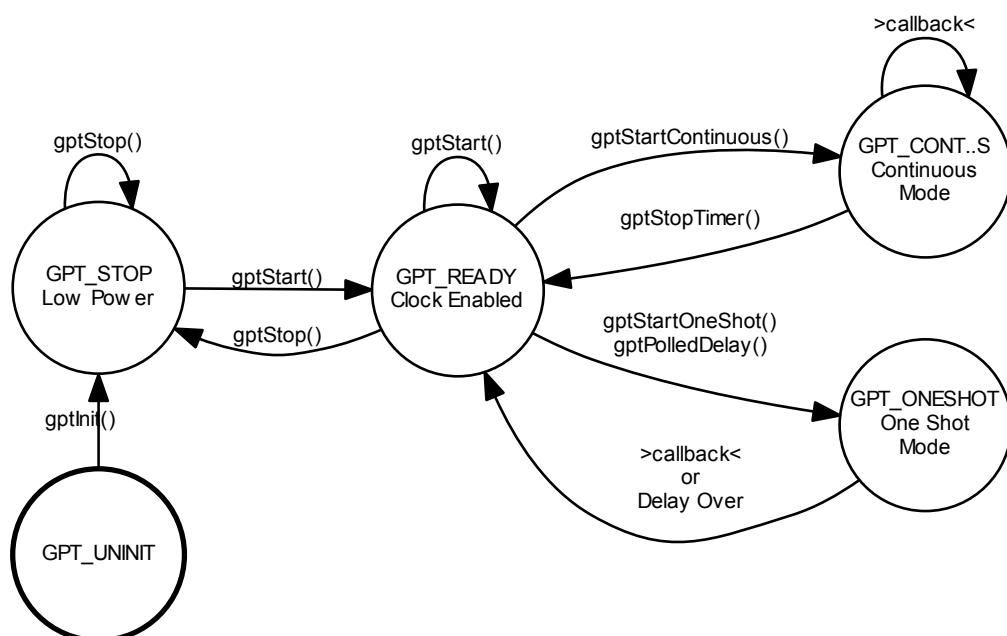
This module implements a generic GPT (General Purpose Timer) driver. The timer can be programmed in order to trigger callbacks after a specified time period or continuously with a specified interval.

#### Precondition

In order to use the GPT driver the `HAL_USE_GPT` option must be enabled in `halconf.h`.

### 7.5.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



### 7.5.3 GPT Operations.

This driver abstracts a generic timer composed of:

- A clock prescaler.
- A main up counter.

- A comparator register that resets the main counter to zero when the limit is reached. A callback is invoked when this happens.

The timer can operate in three different modes:

- **Continuous Mode**, a periodic callback is invoked until the driver is explicitly stopped.
- **One Shot Mode**, a callback is invoked after the programmed period and then the timer automatically stops.
- **Delay Mode**, the timer is used for inserting a brief delay into the execution flow, no callback is invoked in this mode.

## Macros

- `#define gptChangeInterval(gptp, interval)`  
*Changes the interval of GPT peripheral.*
- `#define gptGetIntervalX(gptp) gpt_lld_get_interval(gptp)`  
*Returns the interval of GPT peripheral.*
- `#define gptGetCounterX(gptp) gpt_lld_get_counter(gptp)`  
*Returns the counter value of GPT peripheral.*
- `#define gpt_lld_change_interval(gptp, interval)`  
*Changes the interval of GPT peripheral.*

## PLATFORM configuration options

- `#define PLATFORM_GPT_USE_GPT1 FALSE`  
*GPTD1 driver enable switch.*

## Typedefs

- `typedef struct GPTDriver GPTDriver`  
*Type of a structure representing a GPT driver.*
- `typedef void(* gptcallback_t) (GPTDriver *gptp)`  
*GPT notification callback type.*
- `typedef uint32_t gptfreq_t`  
*GPT frequency type.*
- `typedef uint16_t gptcnt_t`  
*GPT counter type.*

## Data Structures

- `struct GPTConfig`  
*Driver configuration structure.*
- `struct GPTDriver`  
*Structure representing a GPT driver.*

## Functions

- void `gptInit` (void)
 

*GPT Driver initialization.*
- void `gptObjectInit` (GPTDriver \*gptp)
 

*Initializes the standard part of a `GPTDriver` structure.*
- void `gptStart` (GPTDriver \*gptp, const GPTConfig \*config)
 

*Configures and activates the GPT peripheral.*
- void `gptStop` (GPTDriver \*gptp)
 

*Deactivates the GPT peripheral.*
- void `gptChangeInterval` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Changes the interval of GPT peripheral.*
- void `gptStartContinuous` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Starts the timer in continuous mode.*
- void `gptStartContinuousl` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Starts the timer in continuous mode.*
- void `gptStartOneShot` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Starts the timer in one shot mode.*
- void `gptStartOneShotl` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Starts the timer in one shot mode.*
- void `gptStopTimer` (GPTDriver \*gptp)
 

*Stops the timer.*
- void `gptStopTimerl` (GPTDriver \*gptp)
 

*Stops the timer.*
- void `gptPolledDelay` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Starts the timer in one shot mode and waits for completion.*
- void `gpt_lld_init` (void)
 

*Low level GPT driver initialization.*
- void `gpt_lld_start` (GPTDriver \*gptp)
 

*Configures and activates the GPT peripheral.*
- void `gpt_lld_stop` (GPTDriver \*gptp)
 

*Deactivates the GPT peripheral.*
- void `gpt_lld_start_timer` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Starts the timer in continuous mode.*
- void `gpt_lld_stop_timer` (GPTDriver \*gptp)
 

*Stops the timer.*
- void `gpt_lld_polled_delay` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Starts the timer in one shot mode and waits for completion.*

## Enumerations

- enum `gptstate_t` {
 

`GPT_UNINIT` = 0, `GPT_STOP` = 1, `GPT_READY` = 2, `GPT_CONTINUOUS` = 3,  
`GPT_ONESHOT` = 4 }

*Driver state machine possible states.*

## Variables

- GPTDriver GPTD1
 

*GPTD1 driver identifier.*

### 7.5.4 Macro Definition Documentation

#### 7.5.4.1 #define gptChangeInterval( *gptp*, *interval* )

##### Value:

```
{
    \gpt_lld_change_interval(gptp, interval);
}
```

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

##### Precondition

The GPT unit must be running in continuous mode.

##### Postcondition

The GPT unit interval is changed to the new value.

##### Parameters

in	<i>gptp</i>	pointer to a <a href="#">GPTDriver</a> object
in	<i>interval</i>	new cycle time in timer ticks

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

#### 7.5.4.2 #define gptGetIntervalX( *gptp* ) gpt\_lld\_get\_interval(*gptp*)

Returns the interval of GPT peripheral.

##### Precondition

The GPT unit must be running in continuous mode.

##### Parameters

in	<i>gptp</i>	pointer to a <a href="#">GPTDriver</a> object
----	-------------	---

##### Returns

The current interval.

##### Function Class:

This is an **X-Class** API, this function can be invoked from any context.

#### 7.5.4.3 #define gptGetCounterX( *gptp* ) gpt\_lld\_get\_counter(*gptp*)

Returns the counter value of GPT peripheral.

**Precondition**

The GPT unit must be running in continuous mode.

**Note**

The nature of the counter is not defined, it may count upward or downward, it could be continuously running or not.

**Parameters**

in	<i>gptp</i>	pointer to a <a href="#">GPTDriver</a> object
----	-------------	---

**Returns**

The current counter value.

**Function Class:**

This is an **X-Class** API, this function can be invoked from any context.

**7.5.4.4 #define PLATFORM\_GPT\_USE\_GPT1 FALSE**

GPTD1 driver enable switch.

If set to TRUE the support for GPTD1 is included.

**Note**

The default is FALSE.

**7.5.4.5 #define gpt\_lld\_change\_interval( *gptp*, *interval* )****Value:**

```
{
    (void)gptp;
    (void)interval;
}
```

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

**Precondition**

The GPT unit must have been activated using [gptStart\(\)](#).

The GPT unit must have been running in continuous mode using [gptStartContinuous\(\)](#).

**Postcondition**

The GPT unit interval is changed to the new value.

**Note**

The function has effect at the next cycle start.

**Parameters**

in	<i>gptp</i>	pointer to a <a href="#">GPTDriver</a> object
in	<i>interval</i>	new cycle time in timer ticks

**Function Class:**

Not an API, this function is for internal use only.

## 7.5.5 Typedef Documentation

### 7.5.5.1 `typedef struct GPTDriver GPTDriver`

Type of a structure representing a GPT driver.

### 7.5.5.2 `typedef void(* gptcallback_t)(GPTDriver *gptp)`

GPT notification callback type.

#### Parameters

in	<i>gptp</i>	pointer to a <code>GPTDriver</code> object
----	-------------	--

### 7.5.5.3 `typedef uint32_t gptfreq_t`

GPT frequency type.

### 7.5.5.4 `typedef uint16_t gptcnt_t`

GPT counter type.

## 7.5.6 Enumeration Type Documentation

### 7.5.6.1 `enum gptstate_t`

Driver state machine possible states.

#### Enumerator

***GPT\_UNINIT*** Not initialized.

***GPT\_STOP*** Stopped.

***GPT\_READY*** Ready.

***GPT\_CONTINUOUS*** Active in continuous mode.

***GPT\_ONESHOT*** Active in one shot mode.

## 7.5.7 Function Documentation

### 7.5.7.1 `void gptInit( void )`

GPT Driver initialization.

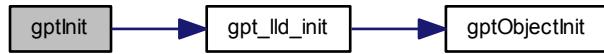
#### Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.5.7.2 void gptObjectInit ( **GPTDriver** \* *gptp* )

Initializes the standard part of a **GPTDriver** structure.

##### Parameters

out	<i>gptp</i>	pointer to the <b>GPTDriver</b> object
-----	-------------	--

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

#### 7.5.7.3 void gptStart ( **GPTDriver** \* *gptp*, const **GPTConfig** \* *config* )

Configures and activates the GPT peripheral.

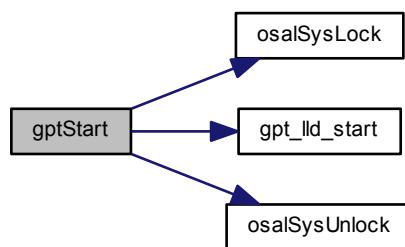
##### Parameters

in	<i>gptp</i>	pointer to the <b>GPTDriver</b> object
in	<i>config</i>	pointer to the <b>GPTConfig</b> object

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.5.7.4 void gptStop ( GPTDriver \* *gptp* )

Deactivates the GPT peripheral.

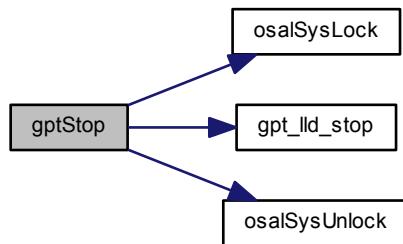
**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.5.7.5 void gptChangeInterval ( [GPTDriver](#) \* *gptp*, [gptcnt\\_t](#) *interval* )

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

**Precondition**

The GPT unit must be running in continuous mode.

**Postcondition**

The GPT unit interval is changed to the new value.

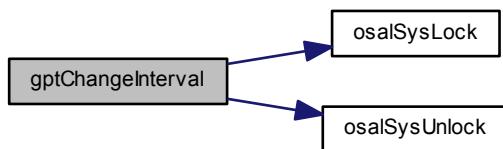
**Parameters**

in	<i>gptp</i>	pointer to a <a href="#">GPTDriver</a> object
in	<i>interval</i>	new cycle time in timer ticks

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.5.7.6 void gptStartContinuous ( GPTDriver \* *gptp*, gptcnt\_t *interval* )**

Starts the timer in continuous mode.

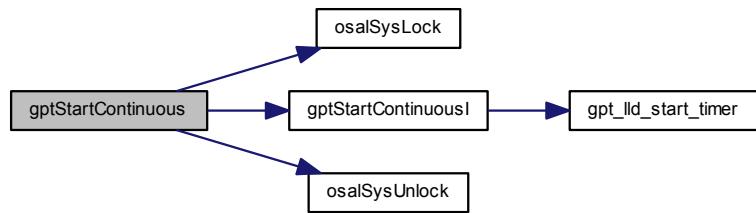
**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
in	<i>interval</i>	period in ticks

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



**7.5.7.7 void gptStartContinuous( GPTDriver \* *gptp*, gptcnt\_t *interval* )**

Starts the timer in continuous mode.

**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
in	<i>interval</i>	period in ticks

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**7.5.7.8 void gptStartOneShot ( GPTDriver \* *gptp*, gptcnt\_t *interval* )**

Starts the timer in one shot mode.

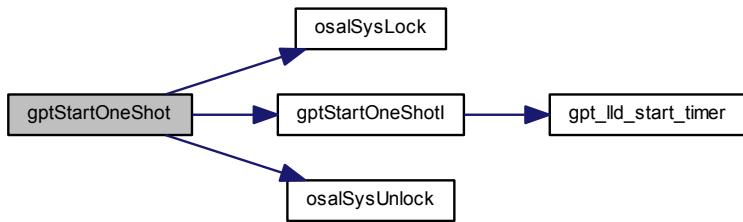
**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
in	<i>interval</i>	time interval in ticks

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.5.7.9 void gptStartOneShotI ( GPTDriver \* *gptp*, gptcnt\_t *interval* )**

Starts the timer in one shot mode.

**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
in	<i>interval</i>	time interval in ticks

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.5.7.10 void gptStopTimer ( GPTDriver \* *gptp* )**

Stops the timer.

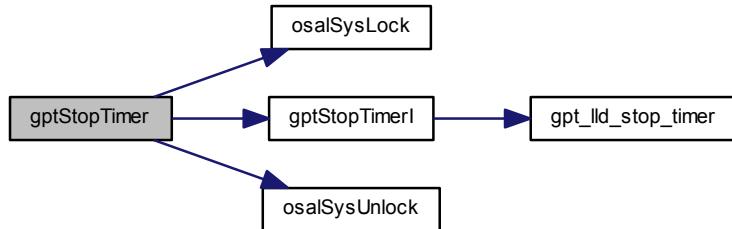
**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.5.7.11 void gptStopTimerI ( GPTDriver \* *gptp* )**

Stops the timer.

**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.5.7.12 void gptPolledDelay ( GPTDriver \* *gptp*, gptcnt\_t *interval* )**

Starts the timer in one shot mode and waits for completion.

This function specifically polls the timer waiting for completion in order to not have extra delays caused by interrupt servicing, this function is only recommended for short delays.

**Note**

The configured callback is not invoked when using this function.

**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
in	<i>interval</i>	time interval in ticks

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.5.7.13 void gpt\_lld\_init ( void )**

Low level GPT driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.5.7.14 void gpt\_lld\_start ( GPTDriver \* gptp )**

Configures and activates the GPT peripheral.

**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.5.7.15 void gpt\_lld\_stop ( GPTDriver \* gptp )**

Deactivates the GPT peripheral.

**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.5.7.16 void gpt\_lld\_start\_timer ( GPTDriver \* gptp, gptcnt\_t interval )**

Starts the timer in continuous mode.

**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
in	<i>interval</i>	period in ticks

**Function Class:**

Not an API, this function is for internal use only.

**7.5.7.17 void gpt\_lld\_stop\_timer ( GPTDriver \* gptp )**

Stops the timer.

**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.5.7.18 void gpt\_lld\_polled\_delay ( GPTDriver \* *gptp*, gptcnt\_t *interval* )**

Starts the timer in one shot mode and waits for completion.

This function specifically polls the timer waiting for completion in order to not have extra delays caused by interrupt servicing, this function is only recommended for short delays.

**Parameters**

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> object
in	<i>interval</i>	time interval in ticks

**Function Class:**

Not an API, this function is for internal use only.

## 7.5.8 Variable Documentation

### 7.5.8.1 GPTDriver GPTD1

GPTD1 driver identifier.

## 7.6 HAL Driver

Hardware Abstraction Layer.

### 7.6.1 Detailed Description

Hardware Abstraction Layer.

The HAL (Hardware Abstraction Layer) driver performs the system initialization and includes the platform support code shared by the other drivers. This driver does contain any API function except for a general initialization function `halInit()` that must be invoked before any HAL service can be used, usually the HAL initialization should be performed immediately before the kernel initialization.

Some HAL driver implementations also offer a custom early clock setup function that can be invoked before the C runtime initialization in order to accelerate the startup time.

### Macros

- `#define _CHIBIOS_HAL_`  
*ChibiOS/HAL identification macro.*
- `#define CH_HAL_STABLE 1`  
*Stable release flag.*

### ChibiOS/HAL version identification

- `#define HAL_VERSION "4.0.3"`  
*HAL version string.*
- `#define CH_HAL_MAJOR 4`  
*HAL version major number.*
- `#define CH_HAL_MINOR 0`  
*HAL version minor number.*
- `#define CH_HAL_PATCH 3`  
*HAL version patch number.*

### Return codes

- `#define HAL_SUCCESS false`
- `#define HAL_FAILED true`

### Platform identification macros

- `#define PLATFORM_NAME "templates"`

### Functions

- `void halInit (void)`  
*HAL initialization.*
- `void hal_lld_init (void)`  
*Low level HAL driver initialization.*

## 7.6.2 Macro Definition Documentation

7.6.2.1 `#define _CHIBIOS_HAL_`

ChibiOS/HAL identification macro.

7.6.2.2 `#define CH_HAL_STABLE 1`

Stable release flag.

7.6.2.3 `#define HAL_VERSION "4.0.3"`

HAL version string.

7.6.2.4 `#define CH_HAL_MAJOR 4`

HAL version major number.

7.6.2.5 `#define CH_HAL_MINOR 0`

HAL version minor number.

7.6.2.6 `#define CH_HAL_PATCH 3`

HAL version patch number.

## 7.6.3 Function Documentation

7.6.3.1 `void halInit( void )`

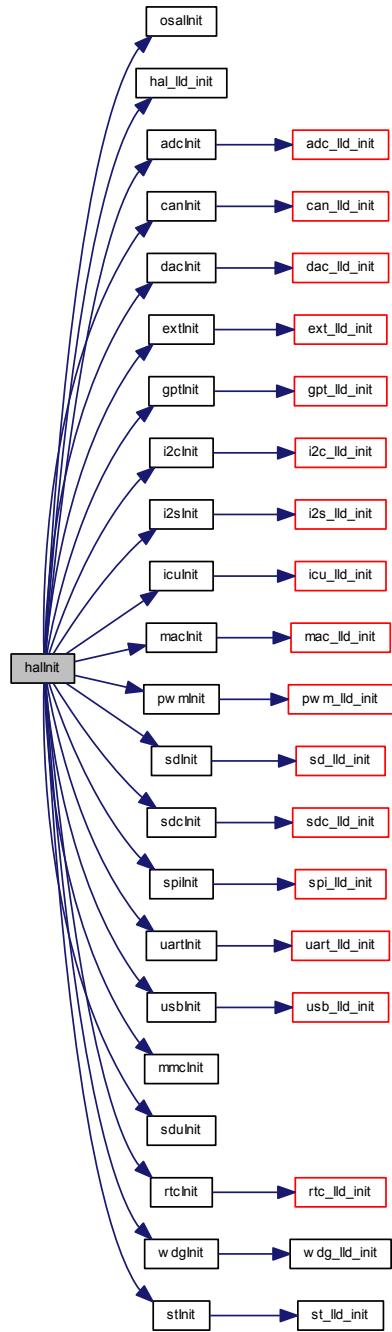
HAL initialization.

This function invokes the low level initialization code then initializes all the drivers enabled in the HAL. Finally the board-specific initialization is performed by invoking `boardInit()` (usually defined in `board.c`).

### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.6.3.2 void hal\_lld\_init ( void )

Low level HAL driver initialization.

#### Function Class:

Not an API, this function is for internal use only.

## 7.7 I/O Buffers Queues

### 7.7.1 Detailed Description

Buffers Queues are used when there is the need to exchange fixed-length data buffers between ISRs and threads. On the ISR side data can be exchanged only using buffers, on the thread side data can be exchanged both using buffers and/or using an emulation of regular byte queues. There are several kind of buffers queues:

- **Input queue**, unidirectional queue where the writer is the ISR side and the reader is the thread side.
- **Output queue**, unidirectional queue where the writer is the ISR side and the reader is the thread side.
- **Full duplex queue**, bidirectional queue. Full duplex queues are implemented by pairing an input queue and an output queue together.

### Macros

- `#define BQ_BUFFER_SIZE(n, size) (((size_t)(size) + sizeof (size_t)) * (size_t)(n))`  
*Computes the size of a buffers queue buffer size.*

### Macro Functions

- `#define bqSizeX(bqp) ((bqp)->bn)`  
*Returns the queue's number of buffers.*
- `#define bqSpacel(bqp) ((bqp)->bcounter)`  
*Return the ready buffers number.*
- `#define bqGetLinkX(bqp) ((bqp)->link)`  
*Returns the queue application-defined link.*
- `#define ibqlIsEmptyl(ibqp) ((bool)(bqSpacel(ibqp) == 0U))`  
*Evaluates to TRUE if the specified input buffers queue is empty.*
- `#define ibqlIsFulll(ibqp)`  
*Evaluates to TRUE if the specified input buffers queue is full.*
- `#define obqlIsEmptyl(obqp)`  
*Evaluates to true if the specified output buffers queue is empty.*
- `#define obqlIsFulll(obqp) ((bool)(bqSpacel(obqp) == 0U))`  
*Evaluates to true if the specified output buffers queue is full.*

### Typedefs

- `typedef struct io_buffers_queue io_buffers_queue_t`  
*Type of a generic queue of buffers.*
- `typedef void(* bqnotify_t) (io_buffers_queue_t *bqp)`  
*Double buffer notification callback type.*
- `typedef io_buffers_queue_t input_buffers_queue_t`  
*Type of an input buffers queue.*
- `typedef io_buffers_queue_t output_buffers_queue_t`  
*Type of an output buffers queue.*

### Data Structures

- `struct io_buffers_queue`  
*Structure of a generic buffers queue.*

## Functions

- `void ibqObjectInit (input_buffers_queue_t *ibqp, uint8_t *bp, size_t size, size_t n, bqnotify_t infy, void *link)`  
*Initializes an input buffers queue object.*
- `void ibqResetI (input_buffers_queue_t *ibqp)`  
*Resets an input buffers queue.*
- `uint8_t * ibqGetEmptyBufferI (input_buffers_queue_t *ibqp)`  
*Gets the next empty buffer from the queue.*
- `void ibqPostFullBufferI (input_buffers_queue_t *ibqp, size_t size)`  
*Posts a new filled buffer to the queue.*
- `msg_t ibqGetFullBufferTimeout (input_buffers_queue_t *ibqp, systime_t timeout)`  
*Gets the next filled buffer from the queue.*
- `msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t *ibqp, systime_t timeout)`  
*Gets the next filled buffer from the queue.*
- `void ibqReleaseEmptyBuffer (input_buffers_queue_t *ibqp)`  
*Releases the buffer back in the queue.*
- `void ibqReleaseEmptyBufferS (input_buffers_queue_t *ibqp)`  
*Releases the buffer back in the queue.*
- `msg_t ibqGetTimeout (input_buffers_queue_t *ibqp, systime_t timeout)`  
*Input queue read with timeout.*
- `size_t ibqReadTimeout (input_buffers_queue_t *ibqp, uint8_t *bp, size_t n, systime_t timeout)`  
*Input queue read with timeout.*
- `void obqObjectInit (output_buffers_queue_t *obqp, uint8_t *bp, size_t size, size_t n, bqnotify_t onfy, void *link)`  
*Initializes an output buffers queue object.*
- `void obqResetI (output_buffers_queue_t *obqp)`  
*Resets an output buffers queue.*
- `uint8_t * obqGetFullBufferI (output_buffers_queue_t *obqp, size_t *sizep)`  
*Gets the next filled buffer from the queue.*
- `void obqReleaseEmptyBufferI (output_buffers_queue_t *obqp)`  
*Releases the next filled buffer back in the queue.*
- `msg_t obqGetEmptyBufferTimeout (output_buffers_queue_t *obqp, systime_t timeout)`  
*Gets the next empty buffer from the queue.*
- `msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t *obqp, systime_t timeout)`  
*Gets the next empty buffer from the queue.*
- `void obqPostFullBuffer (output_buffers_queue_t *obqp, size_t size)`  
*Posts a new filled buffer to the queue.*
- `void obqPostFullBufferS (output_buffers_queue_t *obqp, size_t size)`  
*Posts a new filled buffer to the queue.*
- `msg_t obqPutTimeout (output_buffers_queue_t *obqp, uint8_t b, systime_t timeout)`  
*Output queue write with timeout.*
- `size_t obqWriteTimeout (output_buffers_queue_t *obqp, const uint8_t *bp, size_t n, systime_t timeout)`  
*Output queue write with timeout.*
- `bool obqTryFlushI (output_buffers_queue_t *obqp)`  
*Flushes the current, partially filled, buffer to the queue.*
- `void obqFlush (output_buffers_queue_t *obqp)`  
*Flushes the current, partially filled, buffer to the queue.*

### 7.7.2 Macro Definition Documentation

#### 7.7.2.1 `#define BQ_BUFFER_SIZE( n, size ) (((size_t)(size) + sizeof (size_t)) * (size_t)(n))`

Computes the size of a buffers queue buffer size.

**Parameters**

in	<i>n</i>	number of buffers in the queue
in	<i>size</i>	size of the buffers

**7.7.2.2 #define bqSizeX( *bqp* ) ((bqp)->bn)**

Returns the queue's number of buffers.

**Parameters**

in	<i>bqp</i>	pointer to an <i>io_buffers_queue_t</i> structure
----	------------	---

**Returns**

The number of buffers.

**Function Class:**

This is an **X-Class** API, this function can be invoked from any context.

**7.7.2.3 #define bqSpaceI( *bqp* ) ((bqp)->bcounter)**

Return the ready buffers number.

Returns the number of filled buffers if used on an input queue or the number of empty buffers if used on an output queue.

**Parameters**

in	<i>bqp</i>	pointer to an <i>io_buffers_queue_t</i> structure
----	------------	---

**Returns**

The number of ready buffers.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.7.2.4 #define bqGetLinkX( *bqp* ) ((bqp)->link)**

Returns the queue application-defined link.

**Parameters**

in	<i>bqp</i>	pointer to an <i>io_buffers_queue_t</i> structure
----	------------	---

**Returns**

The application-defined link.

**Function Class:**

Special function, this function has special requirements see the notes.

**7.7.2.5 #define ibqIsEmptyI( *ibqp* ) ((bool)(bqSpaceI(ibqp) == 0U))**

Evaluates to TRUE if the specified input buffers queue is empty.

**Parameters**

in	<i>ibqp</i>	pointer to an <code>input_buffers_queue_t</code> structure
----	-------------	--

**Returns**

The queue status.

**Return values**

<i>false</i>	if the queue is not empty.
<i>true</i>	if the queue is empty.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.7.2.6 #define ibqIsFull( *ibqp* )****Value:**

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/
((bool)((ibqp)->bwrptr == (ibqp)->brdptr) && ((ibqp)->bcounter != 0U)) \ \
/*lint -restore*/
```

Evaluates to TRUE if the specified input buffers queue is full.

**Parameters**

in	<i>ibqp</i>	pointer to an <code>input_buffers_queue_t</code> structure
----	-------------	--

**Returns**

The queue status.

**Return values**

<i>false</i>	if the queue is not full.
<i>true</i>	if the queue is full.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.7.2.7 #define obqIsEmpty( *obqp* )****Value:**

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/
((bool)((obqp)->bwrptr == (obqp)->brdptr) && ((obqp)->bcounter != 0U)) \ \
/*lint -restore*/
```

Evaluates to true if the specified output buffers queue is empty.

**Parameters**

in	<i>obqp</i>	pointer to an <code>output_buffers_queue_t</code> structure
----	-------------	---

**Returns**

The queue status.

**Return values**

<i>false</i>	if the queue is not empty.
<i>true</i>	if the queue is empty.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.7.2.8 #define obqIsFull( *obqp* ) ((bool)(bqSpaceI(*obqp*) == 0U))**

Evaluates to `true` if the specified output buffers queue is full.

**Parameters**

in	<i>obqp</i>	pointer to an <code>output_buffers_queue_t</code> structure
----	-------------	---

**Returns**

The queue status.

**Return values**

<i>false</i>	if the queue is not full.
<i>true</i>	if the queue is full.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.7.3 Typedef Documentation****7.7.3.1 typedef struct io\_buffers\_queue io\_buffers\_queue\_t**

Type of a generic queue of buffers.

**7.7.3.2 typedef void(\* bqnotify\_t) (io\_buffers\_queue\_t \*bqp)**

Double buffer notification callback type.

**Parameters**

in	<i>iodbp</i>	the buffers queue pointer
----	--------------	---------------------------

**7.7.3.3 typedef io\_buffers\_queue\_t input\_buffers\_queue\_t**

Type of an input buffers queue.

#### 7.7.3.4 `typedef io_buffers_queue_t output_buffers_queue_t`

Type of an output buffers queue.

#### 7.7.4 Function Documentation

##### 7.7.4.1 `void ibqObjectInit( input_buffers_queue_t *ibqp, uint8_t *bp, size_t size, size_t n, bqnotify_t infy, void *link )`

Initializes an input buffers queue object.

###### Parameters

out	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
in	<i>bp</i>	pointer to a memory area allocated for buffers
in	<i>size</i>	buffers size
in	<i>n</i>	number of buffers
in	<i>infy</i>	callback called when a buffer is returned to the queue
in	<i>link</i>	application defined pointer

###### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



##### 7.7.4.2 `void ibqReset( input_buffers_queue_t *ibqp )`

Resets an input buffers queue.

All the data in the input buffers queue is erased and lost, any waiting thread is resumed with status `MSG_RESET`.

###### Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

###### Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
----	-------------	--

###### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.7.4.3 uint8\_t \* ibqGetEmptyBufferl ( *input\_buffers\_queue\_t* \* *ibqp* )

Gets the next empty buffer from the queue.

##### Note

The function always returns the same buffer if called repeatedly.

##### Parameters

in	<i>ibqp</i>	pointer to the <i>input_buffers_queue_t</i> object
----	-------------	--

##### Returns

A pointer to the next buffer to be filled.

##### Return values

<i>NULL</i>	if the queue is full.
-------------	-----------------------

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

#### 7.7.4.4 void ibqPostFullBufferl ( *input\_buffers\_queue\_t* \* *ibqp*, *size\_t* *size* )

Posts a new filled buffer to the queue.

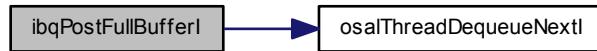
##### Parameters

in	<i>ibqp</i>	pointer to the <i>input_buffers_queue_t</i> object
in	<i>size</i>	used size of the buffer, cannot be zero

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.7.4.5 `msg_t ibqGetFullBufferTimeout( input_buffers_queue_t *ibqp, systime_t timeout )`

Gets the next filled buffer from the queue.

##### Note

The function always acquires the same buffer if called repeatedly.

##### Postcondition

After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

##### Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
in	<i>timeout</i>	<p>the number of ticks before the operation timeouts, the following special values are allowed:</p> <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

##### Returns

The operation status.

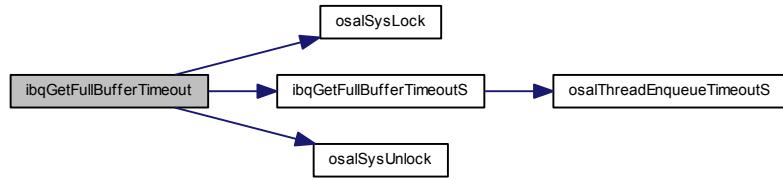
##### Return values

<code>MSG_OK</code>	if a buffer has been acquired.
<code>MSG_TIMEOUT</code>	if the specified time expired.
<code>MSG_RESET</code>	if the queue has been reset.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.7.4.6 `msg_t ibqGetFullBufferTimeoutS( input_buffers_queue_t *ibqp, systime_t timeout )`

Gets the next filled buffer from the queue.

##### Note

The function always acquires the same buffer if called repeatedly.

##### Postcondition

After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

##### Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

##### Returns

The operation status.

##### Return values

<code>MSG_OK</code>	if a buffer has been acquired.
<code>MSG_TIMEOUT</code>	if the specified time expired.
<code>MSG_RESET</code>	if the queue has been reset.

##### Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



#### 7.7.4.7 void ibqReleaseEmptyBuffer ( *input\_buffers\_queue\_t* \* *ibqp* )

Releases the buffer back in the queue.

##### Note

The object callback is called after releasing the buffer.

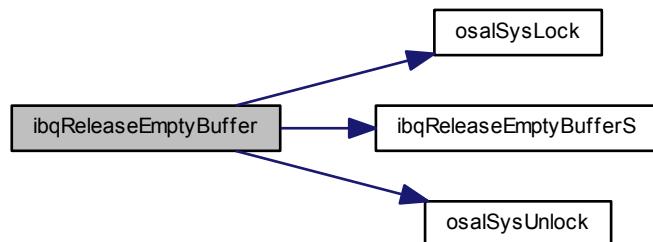
##### Parameters

in	<i>ibqp</i>	pointer to the <i>input_buffers_queue_t</i> object
----	-------------	--

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.7.4.8 void ibqReleaseEmptyBufferS ( *input\_buffers\_queue\_t* \* *ibqp* )

Releases the buffer back in the queue.

##### Note

The object callback is called after releasing the buffer.

**Parameters**

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
----	-------------	--

**Function Class:**

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

**7.7.4.9 `msg_t ibqGetTimeout( input_buffers_queue_t *ibqp, systime_t timeout )`**

Input queue read with timeout.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a new buffer arrives in the queue or a timeout occurs.

**Parameters**

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

**Returns**

A byte value from the queue.

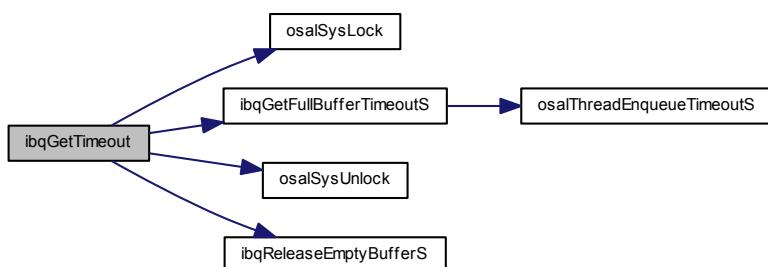
**Return values**

<code>MSG_TIMEOUT</code>	if the specified time expired.
<code>MSG_RESET</code>	if the queue has been reset.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.7.4.10 `size_t ibqReadTimeout( input_buffers_queue_t *ibqp, uint8_t *bp, size_t n, systime_t timeout )`**

Input queue read with timeout.

The function reads data from an input queue into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

#### Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred, the value 0 is reserved
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

#### Returns

The number of bytes effectively transferred.

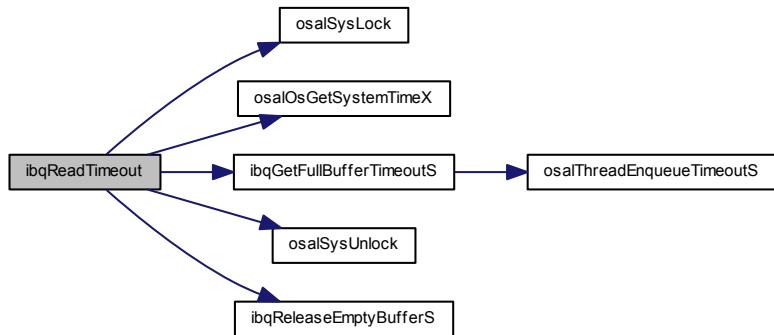
#### Return values

<i>0</i>	if a timeout occurred.
----------	------------------------

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.7.4.11 void obqObjectInit( `output_buffers_queue_t` \* *obqp*, `uint8_t` \* *bp*, `size_t` *size*, `size_t` *n*, `bqnotify_t` *only*, void \* *link* )

Initializes an output buffers queue object.

#### Parameters

out	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
in	<i>bp</i>	pointer to a memory area allocated for buffers
in	<i>size</i>	buffers size
in	<i>n</i>	number of buffers
in	<i>only</i>	callback called when a buffer is posted in the queue
in	<i>link</i>	application defined pointer

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.7.4.12 void obqResetI ( *output\_buffers\_queue\_t* \* *obqp* )

Resets an output buffers queue.

All the data in the output buffers queue is erased and lost, any waiting thread is resumed with status MSG\_RESET.

**Note**

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

**Parameters**

in	<i>obqp</i>	pointer to the <i>output_buffers_queue_t</i> object
----	-------------	---

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.7.4.13 uint8\_t \* obqGetFullBufferI ( *output\_buffers\_queue\_t* \* *obqp*, *size\_t* \* *sizep* )

Gets the next filled buffer from the queue.

**Note**

The function always returns the same buffer if called repeatedly.

**Parameters**

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
out	<i>sizep</i>	pointer to the filled buffer size

**Returns**

A pointer to the filled buffer.

**Return values**

<code>NULL</code>	if the queue is empty.
-------------------	------------------------

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.7.4.14 void obqReleaseEmptyBufferl( `output_buffers_queue_t` \* *obqp* )**

Releases the next filled buffer back in the queue.

**Parameters**

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
----	-------------	---

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**7.7.4.15 msg\_t obqGetEmptyBufferTimeout( `output_buffers_queue_t` \* *obqp*, `systime_t` *timeout* )**

Gets the next empty buffer from the queue.

**Note**

The function always acquires the same buffer if called repeatedly.

**Postcondition**

After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

**Parameters**

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
in	<i>timeout</i>	<p>the number of ticks before the operation timeouts, the following special values are allowed:</p> <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

**Returns**

The operation status.

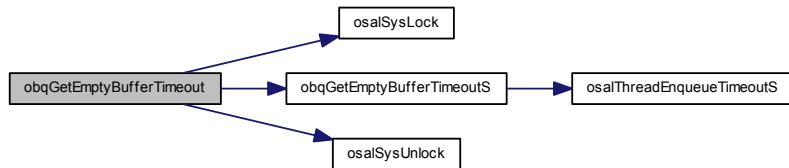
**Return values**

<code>MSG_OK</code>	if a buffer has been acquired.
<code>MSG_TIMEOUT</code>	if the specified time expired.
<code>MSG_RESET</code>	if the queue has been reset.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.7.4.16 `msg_t obqGetEmptyBufferTimeoutS ( output_buffers_queue_t * obqp, systime_t timeout )`

Gets the next empty buffer from the queue.

**Note**

The function always acquires the same buffer if called repeatedly.

**Postcondition**

After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

**Parameters**

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
in	<i>timeout</i>	<p>the number of ticks before the operation timeouts, the following special values are allowed:</p> <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

**Returns**

The operation status.

**Return values**

<i>MSG_OK</i>	if a buffer has been acquired.
<i>MSG_TIMEOUT</i>	if the specified time expired.
<i>MSG_RESET</i>	if the queue has been reset.

**Function Class:**

This is an **S-Class API**, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

**7.7.4.17 void obqPostFullBuffer( output\_buffers\_queue\_t \* obqp, size\_t size )**

Posts a new filled buffer to the queue.

**Note**

The object callback is called after releasing the buffer.

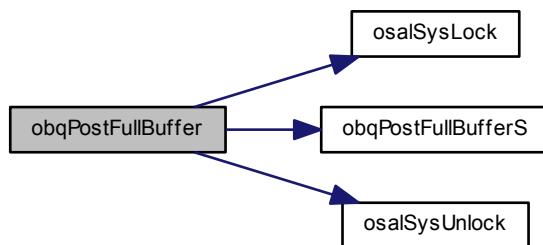
**Parameters**

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
in	<i>size</i>	used size of the buffer, cannot be zero

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.7.4.18 void obqPostFullBufferS ( *output\_buffers\_queue\_t* \* *obqp*, *size\_t* *size* )

Posts a new filled buffer to the queue.

#### Note

The object callback is called after releasing the buffer.

#### Parameters

in	<i>obqp</i>	pointer to the <i>output_buffers_queue_t</i> object
in	<i>size</i>	used size of the buffer, cannot be zero

#### Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

#### 7.7.4.19 *msg\_t* obqPutTimeout ( *output\_buffers\_queue\_t* \* *obqp*, *uint8\_t* *b*, *systime\_t* *timeout* )

Output queue write with timeout.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until a new buffer is freed in the queue or a timeout occurs.

#### Parameters

in	<i>obqp</i>	pointer to the <i>output_buffers_queue_t</i> object
in	<i>b</i>	byte value to be transferred
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <i>TIME_IMMEDIATE</i> immediate timeout.</li> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

#### Returns

A byte value from the queue.

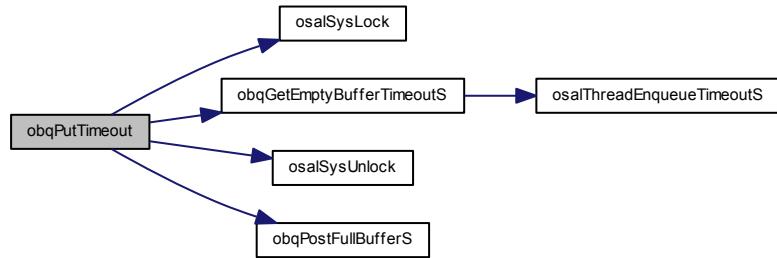
#### Return values

<i>MSG_TIMEOUT</i>	if the specified time expired.
<i>MSG_RESET</i>	if the queue has been reset.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.7.4.20 `size_t obqWriteTimeout( output_buffers_queue_t * obqp, const uint8_t * bp, size_t n, systime_t timeout )`

Output queue write with timeout.

The function writes data from a buffer to an output queue. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

##### Parameters

in	<code>obqp</code>	pointer to the <code>output_buffers_queue_t</code> object
in	<code>bp</code>	pointer to the data buffer
in	<code>n</code>	the maximum amount of data to be transferred, the value 0 is reserved
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

##### Returns

The number of bytes effectively transferred.

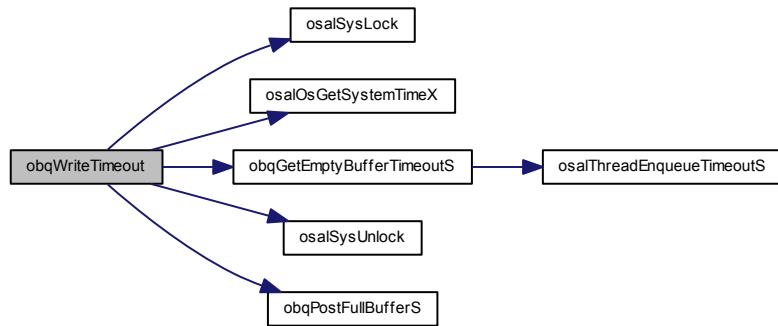
##### Return values

<code>0</code>	if a timeout occurred.
----------------	------------------------

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.7.4.21 bool obqTryFlush( output\_buffers\_queue\_t \* obqp )

Flushes the current, partially filled, buffer to the queue.

##### Note

The notification callback is not invoked because the function is meant to be called from ISR context. An operation status is returned instead.

##### Parameters

in	<code>obqp</code>	pointer to the <code>output_buffers_queue_t</code> object
----	-------------------	---

##### Returns

The operation status.

##### Return values

<code>false</code>	if no new filled buffer has been posted to the queue.
<code>true</code>	if a new filled buffer has been posted to the queue.

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

#### 7.7.4.22 void obqFlush( output\_buffers\_queue\_t \* obqp )

Flushes the current, partially filled, buffer to the queue.

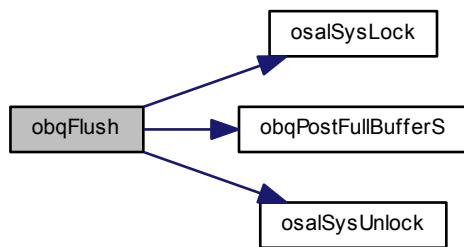
##### Parameters

in	<code>obqp</code>	pointer to the <code>output_buffers_queue_t</code> object
----	-------------------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



## 7.8 Abstract I/O Channel

### 7.8.1 Detailed Description

This module defines an abstract interface for I/O channels by extending the `BaseSequentialStream` interface. Note that no code is present, I/O channels are just abstract interface like structures, you should look at the systems as to a set of abstract C++ classes (even if written in C). Specific device drivers can use/extend the interface and implement them.

This system has the advantage to make the access to channels independent from the implementation logic.

### Macros

- `#define _base_channel_methods`  
`BaseChannel specific methods.`
- `#define _base_channel_data _base_sequential_stream_data`  
`BaseChannel specific data.`
- `#define _base_asynchronous_channel_methods _base_channel_methods \`  
`BaseAsynchronousChannel specific methods.`
- `#define _base_asynchronous_channel_data`  
`BaseAsynchronousChannel specific data.`

### Macro Functions (`BaseChannel`)

- `#define chnPutTimeout(ip, b, time) ((ip)->vmt->putt(ip, b, time))`  
`Channel blocking byte write with timeout.`
- `#define chnGetTimeout(ip, time) ((ip)->vmt->gett(ip, time))`  
`Channel blocking byte read with timeout.`
- `#define chnWrite(ip, bp, n) streamWrite(ip, bp, n)`  
`Channel blocking write.`
- `#define chnWriteTimeout(ip, bp, n, time) ((ip)->vmt->writet(ip, bp, n, time))`  
`Channel blocking write with timeout.`
- `#define chnRead(ip, bp, n) streamRead(ip, bp, n)`  
`Channel blocking read.`
- `#define chnReadTimeout(ip, bp, n, time) ((ip)->vmt->readt(ip, bp, n, time))`  
`Channel blocking read with timeout.`

### I/O status flags added to the event listener

- `#define CHN_NO_ERROR (eventflags_t)0`  
`No pending conditions.`
- `#define CHN_CONNECTED (eventflags_t)1`  
`Connection happened.`
- `#define CHN_DISCONNECTED (eventflags_t)2`  
`Disconnection happened.`
- `#define CHN_INPUT_AVAILABLE (eventflags_t)4`  
`Data available in the input queue.`
- `#define CHN_OUTPUT_EMPTY (eventflags_t)8`  
`Output queue empty.`
- `#define CHN_TRANSMISSION_END (eventflags_t)16`  
`Transmission end.`

## Macro Functions (`BaseAsynchronousChannel`)

- `#define chnGetEventSource(ip) (&((ip)->event))`  
*Returns the I/O condition event source.*
- `#define chnAddFlagsI(ip, flags)`  
*Adds status flags to the listeners's flags mask.*

## Data Structures

- struct `BaseChannelVMT`  
`BaseChannel` virtual methods table.
- struct `BaseChannel`  
`Base channel class.`
- struct `BaseAsynchronousChannelVMT`  
`BaseAsynchronousChannel` virtual methods table.
- struct `BaseAsynchronousChannel`  
`Base asynchronous channel class.`

### 7.8.2 Macro Definition Documentation

#### 7.8.2.1 `#define _base_channel_methods`

##### Value:

```
_base_sequential_stream_methods
/* Channel put method with timeout specification.*/
msg_t (*putt)(void *instance, uint8_t b, systime_t time);
/* Channel get method with timeout specification.*/
msg_t (*gett)(void *instance, systime_t time);
/* Channel write method with timeout specification.*/
size_t (*writet)(void *instance, const uint8_t *bp,
                  size_t n, systime_t time);
/* Channel read method with timeout specification.*/
size_t (*readt)(void *instance, uint8_t *bp, size_t n, systime_t time);
```



`BaseChannel` specific methods.

#### 7.8.2.2 `#define _base_channel_data _base_sequential_stream_data`

`BaseChannel` specific data.

##### Note

It is empty because `BaseChannel` is only an interface without implementation.

#### 7.8.2.3 `#define chnPutTimeout( ip, b, time ) ((ip)->vmt->putt(ip, b, time))`

Channel blocking byte write with timeout.

This function writes a byte value to a channel. If the channel is not ready to accept data then the calling thread is suspended.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseChannel</a> or derived class
in	<i>b</i>	the byte value to be written to the channel
in	<i>time</i>	<p>the number of ticks before the operation timeouts, the following special values are allowed:</p> <ul style="list-style-type: none"> <li>• <i>TIME_IMMEDIATE</i> immediate timeout.</li> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

**Returns**

The operation status.

**Return values**

<i>STM_OK</i>	if the operation succeeded.
<i>STM_TIMEOUT</i>	if the specified time expired.
<i>STM_RESET</i>	if the channel associated queue (if any) was reset.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.8.2.4 #define chnGetTimeout( *ip*, *time* ) ((ip)->vmt->gett(ip, time))**

Channel blocking byte read with timeout.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseChannel</a> or derived class
in	<i>time</i>	<p>the number of ticks before the operation timeouts, the following special values are allowed:</p> <ul style="list-style-type: none"> <li>• <i>TIME_IMMEDIATE</i> immediate timeout.</li> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

**Returns**

A byte value from the queue.

**Return values**

<i>STM_TIMEOUT</i>	if the specified time expired.
<i>STM_RESET</i>	if the channel associated queue (if any) has been reset.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.8.2.5 #define chnWrite( *ip*, *bp*, *n* ) streamWrite(ip, bp, n)**

Channel blocking write.

The function writes data from a buffer to a channel. If the channel is not ready to accept data then the calling thread is suspended.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseChannel</a> or derived class
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

**Returns**

The number of bytes transferred.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.8.2.6 #define chnWriteTimeout( *ip*, *bp*, *n*, *time* ) ((*ip*)>vmt->writet(*ip*, *bp*, *n*, *time*))**

Channel blocking write with timeout.

The function writes data from a buffer to a channel. If the channel is not ready to accept data then the calling thread is suspended.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseChannel</a> or derived class
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred
in	<i>time</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <i>TIME_IMMEDIATE</i> immediate timeout.</li> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

**Returns**

The number of bytes transferred.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.8.2.7 #define chnRead( *ip*, *bp*, *n* ) streamRead(*ip*, *bp*, *n*)**

Channel blocking read.

The function reads data from a channel into a buffer. If the data is not available then the calling thread is suspended.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseChannel</a> or derived class
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

**Returns**

The number of bytes transferred.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

---

7.8.2.8 #define chnReadTimeout( *ip*, *bp*, *n*, *time* ) ((*ip*)>vmt->readt(*ip*, *bp*, *n*, *time*))

Channel blocking read with timeout.

The function reads data from a channel into a buffer. If the data is not available then the calling thread is suspended.

#### Parameters

in	<i>ip</i>	pointer to a <a href="#">BaseChannel</a> or derived class
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred
in	<i>time</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <i>TIME_IMMEDIATE</i> immediate timeout.</li> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

#### Returns

The number of bytes transferred.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.8.2.9 #define CHN\_NO\_ERROR (eventflags\_t)0

No pending conditions.

7.8.2.10 #define CHN\_CONNECTED (eventflags\_t)1

Connection happened.

7.8.2.11 #define CHN\_DISCONNECTED (eventflags\_t)2

Disconnection happened.

7.8.2.12 #define CHN\_INPUT\_AVAILABLE (eventflags\_t)4

Data available in the input queue.

7.8.2.13 #define CHN\_OUTPUT\_EMPTY (eventflags\_t)8

Output queue empty.

7.8.2.14 #define CHN\_TRANSMISSION\_END (eventflags\_t)16

Transmission end.

7.8.2.15 #define \_base\_asynchronous\_channel\_methods \_base\_channel\_methods \

[BaseAsynchronousChannel](#) specific methods.

## 7.8.2.16 #define \_base\_asynchronous\_channel\_data

**Value:**

```
_base_channel_data \
/* I/O condition event source.*/
event_source_t     event;
```

[BaseAsynchronousChannel](#) specific data.

## 7.8.2.17 #define chnGetEventSource( ip ) (&amp;((ip)-&gt;event))

Returns the I/O condition event source.

The event source is broadcasted when an I/O condition happens.

**Parameters**

in	ip	pointer to a <a href="#">BaseAsynchronousChannel</a> or derived class
----	----	---

**Returns**

A pointer to an `EventSource` object.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

## 7.8.2.18 #define chnAddFlagsI( ip, flags )

**Value:**

```
{
    osalEventBroadcastFlagsI(&(ip)->event, flags);
}
```

Adds status flags to the listeners's flags mask.

This function is usually called from the I/O ISRs in order to notify I/O conditions such as data events, errors, signal changes etc.

**Parameters**

in	ip	pointer to a <a href="#">BaseAsynchronousChannel</a> or derived class
in	flags	condition flags to be added to the listener flags mask

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

## 7.9 Abstract Files

### 7.9.1 Detailed Description

This module define an abstract interface for generic data files by extending the `BaseSequentialStream` interface. Note that no code is present, data files are just abstract interface-like structures, you should look at the systems as to a set of abstract C++ classes (even if written in C). This system has the advantage to make the access to streams independent from the implementation logic.

The data files interface can be used as base class for high level object types such as an API for a File System implementation.

### Macros

- `#define _file_stream_methods`  
*FileStream specific methods.*
- `#define _file_stream_data_base_sequential_stream_data`  
*FileStream specific data.*

### Files return codes

- `#define FILE_OK STM_OK`  
*No error return code.*
- `#define FILE_ERROR STM_TIMEOUT`  
*Error code from the file stream methods.*
- `#define FILE_EOF STM_RESET`  
*End-of-file condition for file get/put methods.*

### Macro Functions (FileStream)

- `#define fileStreamWrite(ip, bp, n) streamWrite(ip, bp, n)`  
*File stream write.*
- `#define fileStreamRead(ip, bp, n) streamRead(ip, bp, n)`  
*File stream read.*
- `#define fileStreamPut(ip, b) streamPut(ip, b)`  
*File stream blocking byte write.*
- `#define fileStreamGet(ip) streamGet(ip)`  
*File stream blocking byte read.*
- `#define fileStreamClose(ip) ((ip)->vmt->close(ip))`  
*File Stream close.*
- `#define fileStreamGetError(ip) ((ip)->vmt->geterror(ip))`  
*Returns an implementation dependent error code.*
- `#define fileStreamGetSize(ip) ((ip)->vmt->getsize(ip))`  
*Returns the current file size.*
- `#define fileStreamGetPosition(ip) ((ip)->vmt->getposition(ip))`  
*Returns the current file pointer position.*
- `#define fileStreamSeek(ip, offset) ((ip)->vmt->lseek(ip, offset))`  
*Moves the file current pointer to an absolute position.*

### Typedefs

- `typedef uint32_t fileoffset_t`  
*File offset type.*

## Data Structures

- struct `FileStreamVMT`  
*FileStream virtual methods table.*
- struct `FileStream`  
*Base file stream class.*

### 7.9.2 Macro Definition Documentation

#### 7.9.2.1 #define FILE\_OK STM\_OK

No error return code.

#### 7.9.2.2 #define FILE\_ERROR STM\_TIMEOUT

Error code from the file stream methods.

#### 7.9.2.3 #define FILE\_EOF STM\_RESET

End-of-file condition for file get/put methods.

#### 7.9.2.4 #define \_file\_stream\_methods

##### Value:

```
_base_sequential_stream_methods
/* File close method.*/
msg_t (*close)(void *instance);
/* Get last error code method.*/
msg_t (*geterror)(void *instance);
/* File get size method.*/
msg_t (*getsize)(void *instance);
/* File get current position method.*/
msg_t (*getposition)(void *instance);
/* File seek method.*/
msg_t (*lseek)(void *instance, fileoffset_t offset);
```



`FileStream` specific methods.

#### 7.9.2.5 #define \_file\_stream\_data \_base\_sequential\_stream\_data

`FileStream` specific data.

##### Note

It is empty because `FileStream` is only an interface without implementation.

#### 7.9.2.6 #define fileStreamWrite( ip, bp, n ) streamWrite(ip, bp, n)

File stream write.

The function writes data from a buffer to a file stream.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">FileStream</a> or derived class
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

**Returns**

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

**Return values**

<a href="#">FILE_ERROR</a>	operation failed.
----------------------------	-------------------

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.9.2.7 #define fileStreamRead( *ip*, *bp*, *n* ) streamRead(*ip*, *bp*, *n*)**

File stream read.

The function reads data from a file stream into a buffer.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">FileStream</a> or derived class
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

**Returns**

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

**Return values**

<a href="#">FILE_ERROR</a>	operation failed.
----------------------------	-------------------

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.9.2.8 #define fileStreamPut( *ip*, *b* ) streamPut(*ip*, *b*)**

File stream blocking byte write.

This function writes a byte value to a channel. If the channel is not ready to accept data then the calling thread is suspended.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">FileStream</a> or derived class
in	<i>b</i>	the byte value to be written to the channel

**Returns**

The operation status.

## Return values

<i>FILE_OK</i>	if the operation succeeded.
<i>FILE_ERROR</i>	operation failed.
<i>FILE_EOF</i>	if an end-of-file condition has been met.

## Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.2.9 #define fileStreamGet( *ip* ) streamGet(*ip*)

File stream blocking byte read.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

## Parameters

in	<i>ip</i>	pointer to a <a href="#">FileStream</a> or derived class
----	-----------	--

## Returns

A byte value from the queue.

## Return values

<i>FILE_ERROR</i>	operation failed.
<i>FILE_EOF</i>	if an end-of-file condition has been met.

## Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.2.10 #define fileStreamClose( *ip* ) ((*ip*)->vmt->close(*ip*))

File Stream close.

The function closes a file stream.

## Parameters

in	<i>ip</i>	pointer to a <a href="#">FileStream</a> or derived class
----	-----------	--

## Returns

The operation status.

## Return values

<i>FILE_OK</i>	no error.
<i>FILE_ERROR</i>	operation failed.

## Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.9.2.11 #define fileStreamGetError( *ip* ) ((ip)->vmt->geterror(ip))**

Returns an implementation dependent error code.

**Precondition**

The previously called function must have returned FILE\_ERROR.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">FileStream</a> or derived class
----	-----------	--

**Returns**

Implementation dependent error code.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.9.2.12 #define fileStreamGetSize( *ip* ) ((ip)->vmt->getsize(ip))**

Returns the current file size.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">FileStream</a> or derived class
----	-----------	--

**Returns**

The file size.

**Return values**

FILE_ERROR	operation failed.
------------	-------------------

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.9.2.13 #define fileStreamGetPosition( *ip* ) ((ip)->vmt->getposition(ip))**

Returns the current file pointer position.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">FileStream</a> or derived class
----	-----------	--

**Returns**

The current position inside the file.

**Return values**

FILE_ERROR	operation failed.
------------	-------------------

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.2.14 #define fileStreamSeek( *ip*, *offset* ) ((*ip*)->vmt->lseek(*ip*, *offset*))

Moves the file current pointer to an absolute position.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">FileStream</a> or derived class
in	<i>offset</i>	new absolute position

**Returns**

The operation status.

**Return values**

<i>FILE_OK</i>	no error.
<i>FILE_ERROR</i>	operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.9.3 Typedef Documentation

#### 7.9.3.1 `typedef uint32_t fileoffset_t`

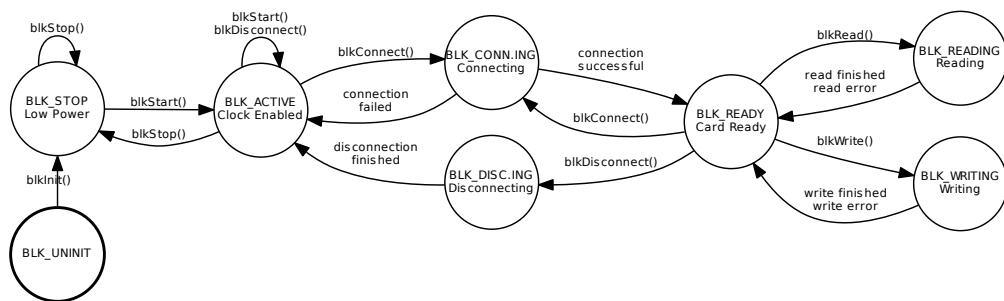
File offset type.

## 7.10 Abstract I/O Block Device

### 7.10.1 Detailed Description

### 7.10.2 Driver State Machine

The drivers implementing this interface shall implement the following state machine internally. Not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



This module defines an abstract interface for accessing generic block devices.

Note that no code is present, just abstract interfaces-like structures, you should look at the system as to a set of abstract C++ classes (even if written in C). This system has then advantage to make the access to block devices independent from the implementation logic.

### Macros

- `#define _base_block_device_methods`  
*BaseBlockDevice specific methods.*
- `#define _base_block_device_data`  
*BaseBlockDevice specific data.*

### Macro Functions (BaseBlockDevice)

- `#define blkGetDriverState(ip) ((ip)->state)`  
*Returns the driver state.*
- `#define blkIsTransferring(ip)`  
*Determines if the device is transferring data.*
- `#define blkIsInserted(ip) ((ip)->vmt->is_inserted(ip))`  
*Returns the media insertion status.*
- `#define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))`  
*Returns the media write protection status.*
- `#define blkConnect(ip) ((ip)->vmt->connect(ip))`  
*Performs the initialization procedure on the block device.*
- `#define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))`

- Terminates operations on the block device.*
- #define `blkRead(ip, startblk, buf, n)` ((ip)->vmt->read(ip, startblk, buf, n))  
*Reads one or more blocks.*
  - #define `blkWrite(ip, startblk, buf, n)` ((ip)->vmt->write(ip, startblk, buf, n))  
*Writes one or more blocks.*
  - #define `blkSync(ip)` ((ip)->vmt->`sync(ip)`)  
*Ensures write synchronization.*
  - #define `blkGetInfo(ip, bdip)` ((ip)->vmt->get\_info(ip, bdip))  
*Returns a media information structure.*

## Data Structures

- struct `BlockDeviceInfo`  
*Block device info.*
- struct `BaseBlockDeviceVMT`  
*BaseBlockDevice virtual methods table.*
- struct `BaseBlockDevice`  
*Base block device class.*

## Enumerations

- enum `blkstate_t` {
 `BLK_UNINIT` = 0, `BLK_STOP` = 1, `BLK_ACTIVE` = 2, `BLK_CONNECTING` = 3,  
`BLK_DISCONNECTING` = 4, `BLK_READY` = 5, `BLK_READING` = 6, `BLK_WRITING` = 7,  
`BLK_SYNCING` = 8 }

*Driver state machine possible states.*

### 7.10.3 Macro Definition Documentation

#### 7.10.3.1 #define \_base\_block\_device\_methods

##### Value:

```
/* Removable media detection.*/
bool (*is_inserted)(void *instance);
/* Removable write protection detection.*/
bool (*is_protected)(void *instance);
/* Connection to the block device.*/
bool (*connect)(void *instance);
/* Disconnection from the block device.*/
bool (*disconnect)(void *instance);
/* Reads one or more blocks.*/
bool (*read)(void *instance, uint32_t startblk,
            uint8_t *buffer, uint32_t n);
/* Writes one or more blocks.*/
bool (*write)(void *instance, uint32_t startblk,
              const uint8_t *buffer, uint32_t n);
/* Write operations synchronization.*/
bool (*sync)(void *instance);
/* Obtains info about the media.*/
bool (*get_info)(void *instance, BlockDeviceInfo *bdip);
```



`BaseBlockDevice` specific methods.

## 7.10.3.2 #define \_base\_block\_device\_data

**Value:**

```
/* Driver state */
blkstate_t state;
```

`BaseBlockDevice` specific data.

## 7.10.3.3 #define blkGetDriverState( ip ) ((ip)-&gt;state)

Returns the driver state.

**Note**

Can be called in ISR context.

**Parameters**

in	ip	pointer to a <code>BaseBlockDevice</code> or derived class
----	----	--

**Returns**

The driver state.

**Function Class:**

Special function, this function has special requirements see the notes.

## 7.10.3.4 #define blkIsTransferring( ip )

**Value:**

```
((((ip)->state) == BLK_CONNECTING) ||
    (((ip)->state) == BLK_DISCONNECTING) ||
    (((ip)->state) == BLK_READING) ||
    (((ip)->state) == BLK_WRITING))
```

Determines if the device is transferring data.

**Note**

Can be called in ISR context.

**Parameters**

in	ip	pointer to a <code>BaseBlockDevice</code> or derived class
----	----	--

**Returns**

The driver state.

**Return values**

<i>FALSE</i>	the device is not transferring data.
<i>TRUE</i>	the device not transferring data.

**Function Class:**

Special function, this function has special requirements see the notes.

7.10.3.5 #define blkIsInserted( *ip* ) ((ip)->vmt->is\_inserted(ip))

Returns the media insertion status.

**Note**

On some implementations this function can only be called if the device is not transferring data. The function [blkIsTransferring\(\)](#) should be used before calling this function.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseBlockDevice</a> or derived class
----	-----------	---

**Returns**

The media state.

**Return values**

<i>FALSE</i>	media not inserted.
<i>TRUE</i>	media inserted.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.3.6 #define blkIsWriteProtected( *ip* ) ((ip)->vmt->is\_protected(ip))

Returns the media write protection status.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseBlockDevice</a> or derived class
----	-----------	---

**Returns**

The media state.

**Return values**

<i>FALSE</i>	writable media.
<i>TRUE</i>	non writable media.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.3.7 #define blkConnect( *ip* ) ((ip)->vmt->connect(ip))

Performs the initialization procedure on the block device.

This function should be performed before I/O operations can be attempted on the block device and after insertion has been confirmed using [blkIsInserted\(\)](#).

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseBlockDevice</a> or derived class
----	-----------	---

**Returns**

The operation status.

**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.10.3.8 #define blkDisconnect( *ip* ) ((*ip*)->vmt->disconnect(*ip*))**

Terminates operations on the block device.

This operation safely terminates operations on the block device.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseBlockDevice</a> or derived class
----	-----------	---

**Returns**

The operation status.

**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.10.3.9 #define blkRead( *ip*, *startblk*, *buf*, *n* ) ((*ip*)->vmt->read(*ip*, *startblk*, *buf*, *n*))**

Reads one or more blocks.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseBlockDevice</a> or derived class
in	<i>startblk</i>	first block to read
out	<i>buf</i>	pointer to the read buffer
in	<i>n</i>	number of blocks to read

**Returns**

The operation status.

**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.3.10 #define blkWrite( *ip*, *startblk*, *buf*, *n* ) ((*ip*)->vmt->write(*ip*, *startblk*, *buf*, *n*))

Writes one or more blocks.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseBlockDevice</a> or derived class
in	<i>startblk</i>	first block to write
out	<i>buf</i>	pointer to the write buffer
in	<i>n</i>	number of blocks to write

**Returns**

The operation status.

**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.3.11 #define blkSync( *ip* ) ((*ip*)->vmt->**sync**(*ip*))

Ensures write synchronization.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseBlockDevice</a> or derived class
----	-----------	---

**Returns**

The operation status.

**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.3.12 #define blkGetInfo( *ip*, *bdip* ) ((*ip*)->vmt->**get\_info**(*ip*, *bdip*))

Returns a media information structure.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseBlockDevice</a> or derived class
out	<i>bdipl</i>	pointer to a <a href="#">BlockDeviceInfo</a> structure

**Returns**

The operation status.

**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.10.4 Enumeration Type Documentation****7.10.4.1 enum blkstate\_t**

Driver state machine possible states.

**Enumerator**

***BLK\_UNINIT*** Not initialized.

***BLK\_STOP*** Stopped.

***BLK\_ACTIVE*** Interface active.

***BLK\_CONNECTING*** Connection in progress.

***BLK\_DISCONNECTING*** Disconnection in progress.

***BLK\_READY*** Device ready.

***BLK\_READING*** Read operation in progress.

***BLK\_WRITING*** Write operation in progress.

***BLK\_SYNCING*** Sync. operation in progress.

## 7.11 I/O Bytes Queues

### 7.11.1 Detailed Description

Queues are mostly used in serial-like device drivers. Serial device drivers are usually designed to have a lower side (lower driver, it is usually an interrupt service routine) and an upper side (upper driver, accessed by the application threads).

There are several kind of queues:

- **Input queue**, unidirectional queue where the writer is the lower side and the reader is the upper side.
- **Output queue**, unidirectional queue where the writer is the upper side and the reader is the lower side.
- **Full duplex queue**, bidirectional queue. Full duplex queues are implemented by pairing an input queue and an output queue together.

#### Queue functions returned status value

- `#define Q_OK MSG_OK`  
*Operation successful.*
- `#define Q_TIMEOUT MSG_TIMEOUT`  
*Timeout condition.*
- `#define Q_RESET MSG_RESET`  
*Queue has been reset.*
- `#define Q_EMPTY (msg_t)-3`  
*Queue empty.*
- `#define Q_FULL (msg_t)-4`  
*Queue full. .*

#### Macro Functions

- `#define qSizeX(qp)`  
*Returns the queue's buffer size.*
- `#define qSpaceI(qp) ((qp)->q_counter)`  
*Queue space.*
- `#define qGetLink(qp) ((qp)->q_link)`  
*Returns the queue application-defined link.*
- `#define iqGetFullI(iqp) qSpaceI(iqp)`  
*Returns the filled space into an input queue.*
- `#define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpaceI(iqp))`  
*Returns the empty space into an input queue.*
- `#define iqIsEmptyI(iqp) ((bool)(qSpaceI(iqp) == 0U))`  
*Evaluates to true if the specified input queue is empty.*
- `#define iqIsFullI(iqp)`  
*Evaluates to true if the specified input queue is full.*
- `#define iqGet(iqp) iqGetTimeout(iqp, TIME_INFINITE)`  
*Input queue read.*
- `#define oqGetFullI(oqp) (qSizeX(oqp) - qSpaceI(oqp))`  
*Returns the filled space into an output queue.*
- `#define oqGetEmptyI(oqp) qSpaceI(oqp)`  
*Returns the empty space into an output queue.*

- `#define oqIsEmpty(oqp)`  
*Evaluates to true if the specified output queue is empty.*
- `#define oqIsFull(oqp) ((bool)(qSpace1(oqp) == 0U))`  
*Evaluates to true if the specified output queue is full.*
- `#define oqPut(oqp, b) oqPutTimeout(oqp, b, TIME_INFINITE)`  
*Output queue write.*

## Typedefs

- `typedef struct io_queue io_queue_t`  
*Type of a generic I/O queue structure.*
- `typedef void(* qnotify_t) (io_queue_t *qp)`  
*Queue notification callback type.*
- `typedef io_queue_t input_queue_t`  
*Type of an input queue structure.*
- `typedef io_queue_t output_queue_t`  
*Type of an output queue structure.*

## Data Structures

- `struct io_queue`  
*Generic I/O queue structure.*

## Functions

- `void iqObjectInit (input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)`  
*Initializes an input queue.*
- `void iqResetl (input_queue_t *iqp)`  
*Resets an input queue.*
- `msg_t iqPutl (input_queue_t *iqp, uint8_t b)`  
*Input queue write.*
- `msg_t iqGetTimeout (input_queue_t *iqp, systime_t timeout)`  
*Input queue read with timeout.*
- `size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, systime_t timeout)`  
*Input queue read with timeout.*
- `void oqObjectInit (output_queue_t *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)`  
*Initializes an output queue.*
- `void oqResetl (output_queue_t *oqp)`  
*Resets an output queue.*
- `msg_t oqPutTimeout (output_queue_t *oqp, uint8_t b, systime_t timeout)`  
*Output queue write with timeout.*
- `msg_t oqGetl (output_queue_t *oqp)`  
*Output queue read.*
- `size_t oqWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, systime_t timeout)`  
*Output queue write with timeout.*

### 7.11.2 Macro Definition Documentation

#### 7.11.2.1 #define Q\_OK MSG\_OK

Operation successful.

### 7.11.2.2 #define Q\_TIMEOUT MSG\_TIMEOUT

Timeout condition.

### 7.11.2.3 #define Q\_RESET MSG\_RESET

Queue has been reset.

### 7.11.2.4 #define Q\_EMPTY (msg\_t)-3

Queue empty.

### 7.11.2.5 #define Q\_FULL (msg\_t)-4

Queue full.,.

### 7.11.2.6 #define qSizeX( qp )

#### Value:

```
/*lint -save -e9033 [10.8] The cast is safe.*/
((size_t)((qp)->q_top - (qp)->q_buffer)) \
/*lint -restore*/\ \
```

Returns the queue's buffer size.

#### Parameters

in	qp	pointer to a <code>io_queue_t</code> structure
----	----	--

#### Returns

The buffer size.

#### Function Class:

This is an **X-Class** API, this function can be invoked from any context.

### 7.11.2.7 #define qSpaceI( qp ) ((qp)->q\_counter)

Queue space.

Returns the used space if used on an input queue or the empty space if used on an output queue.

#### Parameters

in	qp	pointer to a <code>io_queue_t</code> structure
----	----	--

#### Returns

The buffer space.

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.11.2.8 #define qGetLink( *qp* ) ((*qp*)->q\_link)

Returns the queue application-defined link.

#### Note

This function can be called in any context.

#### Parameters

in	<i>qp</i>	pointer to a <code>io_queue_t</code> structure
----	-----------	--

#### Returns

The application-defined link.

#### Function Class:

Special function, this function has special requirements see the notes.

7.11.2.9 #define iqGetFull( *iqp* ) qSpace(iqp)

Returns the filled space into an input queue.

#### Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

#### Returns

The number of full bytes in the queue.

#### Return values

<i>O</i>	if the queue is empty.
----------	------------------------

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.11.2.10 #define iqGetEmpty( *iqp* ) (qSizeX(iqp) - qSpace(iqp))

Returns the empty space into an input queue.

#### Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

#### Returns

The number of empty bytes in the queue.

**Return values**

<i>O</i>	if the queue is full.
----------	-----------------------

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.11.2.11 #define iqIsEmpty( *iqp* ) ((bool)(qSpace1(iqp) == 0U))**

Evaluates to `true` if the specified input queue is empty.

**Parameters**

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

**Returns**

The queue status.

**Return values**

<i>false</i>	if the queue is not empty.
<i>true</i>	if the queue is empty.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.11.2.12 #define iqIsFull( *iqp* )****Value:**

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/
((bool)((iqp)->q_wptr == (iqp)->q_rptr) && ((iqp)->q_counter != 0U)) \
/*lint -restore*/
```

Evaluates to `true` if the specified input queue is full.

**Parameters**

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

**Returns**

The queue status.

**Return values**

<i>false</i>	if the queue is not full.
<i>true</i>	if the queue is full.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.11.2.13 #define iqGet( *iqp* ) iqGetTimeout(iqp, TIME\_INFINITE)

Input queue read.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a byte arrives in the queue.

#### Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

#### Returns

A byte value from the queue.

#### Return values

<i>Q_RESET</i>	if the queue has been reset.
----------------	------------------------------

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.2.14 #define oqGetFull( *oqp* ) (qSizeX(oqp) - qSpaceI(oqp))

Returns the filled space into an output queue.

#### Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

#### Returns

The number of full bytes in the queue.

#### Return values

<i>0</i>	if the queue is empty.
----------	------------------------

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.11.2.15 #define oqGetEmpty( *oqp* ) qSpaceI(oqp)

Returns the empty space into an output queue.

#### Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

#### Returns

The number of empty bytes in the queue.

## Return values

<i>O</i>	if the queue is full.
----------	-----------------------

## Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.11.2.16 #define oqIsEmpty( *oqp* )

## Value:

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/
((bool) (((oqp)->q_wptr == (oqp)->q_rptr) && ((oqp)->q_counter != 0U))) \
/*lint -restore*/
```

Evaluates to `true` if the specified output queue is empty.

## Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

## Returns

The queue status.

## Return values

<i>false</i>	if the queue is not empty.
<i>true</i>	if the queue is empty.

## Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.11.2.17 #define oqIsFull( *oqp* ) ((bool)(qSpace(oqp) == 0U))

Evaluates to `true` if the specified output queue is full.

## Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

## Returns

The queue status.

## Return values

<i>false</i>	if the queue is not full.
<i>true</i>	if the queue is full.

## Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

---

7.11.2.18 #define oqPut( *oqp*, *b* ) oqPutTimeout(*oqp*, *b*, TIME\_INFINITE)

Output queue write.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until there is space in the queue.

#### Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
in	<i>b</i>	the byte value to be written in the queue

#### Returns

The operation status.

#### Return values

<i>Q_OK</i>	if the operation succeeded.
<i>Q_RESET</i>	if the queue has been reset.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.11.3 Typedef Documentation

7.11.3.1 `typedef struct io_queue io_queue_t`

Type of a generic I/O queue structure.

7.11.3.2 `typedef void(* qnotify_t)(io_queue_t *qp)`

Queue notification callback type.

#### Parameters

in	<i>qp</i>	the queue pointer
----	-----------	-------------------

7.11.3.3 `typedef io_queue_t input_queue_t`

Type of an input queue structure.

This structure represents a generic asymmetrical input queue. Writing to the queue is non-blocking and can be performed from interrupt handlers or from within a kernel lock zone. Reading the queue can be a blocking operation and is supposed to be performed by a system thread.

7.11.3.4 `typedef io_queue_t output_queue_t`

Type of an output queue structure.

This structure represents a generic asymmetrical output queue. Reading from the queue is non-blocking and can be performed from interrupt handlers or from within a kernel lock zone. Writing the queue can be a blocking operation and is supposed to be performed by a system thread.

### 7.11.4 Function Documentation

#### 7.11.4.1 void iqObjectInit ( *input\_queue\_t* \* *iqp*, *uint8\_t* \* *bp*, *size\_t* *size*, *qnotify\_t* *infy*, *void* \* *link* )

Initializes an input queue.

A Semaphore is internally initialized and works as a counter of the bytes contained in the queue.

##### Note

The callback is invoked from within the S-Locked system state.

##### Parameters

<i>out</i>	<i>iqp</i>	pointer to an <i>input_queue_t</i> structure
<i>in</i>	<i>bp</i>	pointer to a memory area allocated as queue buffer
<i>in</i>	<i>size</i>	size of the queue buffer
<i>in</i>	<i>infy</i>	pointer to a callback function that is invoked when data is read from the queue. The value can be NULL.
<i>in</i>	<i>link</i>	application defined pointer

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.11.4.2 void iqResetI ( *input\_queue\_t* \* *iqp* )

Resets an input queue.

All the data in the input queue is erased and lost, any waiting thread is resumed with status Q\_RESET.

##### Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

##### Parameters

<i>in</i>	<i>iqp</i>	pointer to an <i>input_queue_t</i> structure
-----------	------------	--

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.11.4.3 msg\_t iqPutl( input\_queue\_t \* iqp, uint8\_t b )

Input queue write.

A byte value is written into the low end of an input queue.

##### Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
in	<i>b</i>	the byte value to be written in the queue

##### Returns

The operation status.

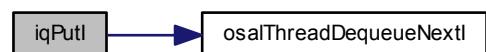
##### Return values

<i>Q_OK</i>	if the operation has been completed with success.
<i>Q_FULL</i>	if the queue is full and the operation cannot be completed.

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.11.4.4 msg\_t iqGetTimeout( input\_queue\_t \* iqp, systime\_t timeout )

Input queue read with timeout.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a byte arrives in the queue or a timeout occurs.

**Note**

The callback is invoked before reading the character from the buffer or before entering the state THD\_STA←TE\_WTQUEUE.

**Parameters**

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

**Returns**

A byte value from the queue.

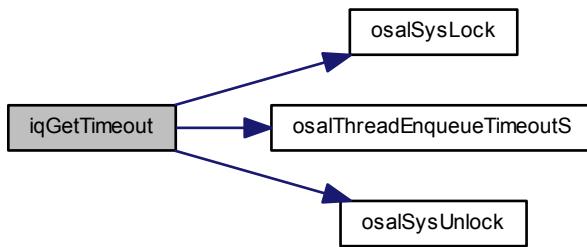
**Return values**

<code>Q_TIMEOUT</code>	if the specified time expired.
<code>Q_RESET</code>	if the queue has been reset.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.11.4.5 `size_t iqReadTimeout( input_queue_t * iq, uint8_t * bp, size_t n, systime_t timeout )`

Input queue read with timeout.

The function reads data from an input queue into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

**Note**

The function is not atomic, if you need atomicity it is suggested to use a semaphore or a mutex for mutual exclusion.

The callback is invoked before reading each character from the buffer or before entering the state THD\_STA←TE\_WTQUEUE.

**Parameters**

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred, the value 0 is reserved
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

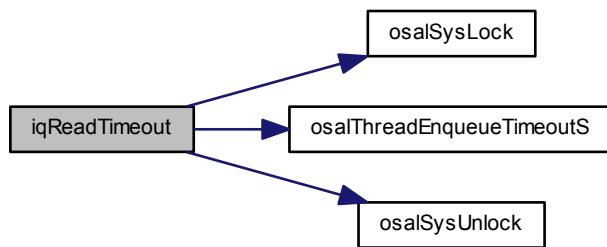
**Returns**

The number of bytes effectively transferred.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.11.4.6 void oqObjectInit ( `output_queue_t * oqp`, `uint8_t * bp`, `size_t size`, `qnotify_t onfy`, `void * link` )**

Initializes an output queue.

A Semaphore is internally initialized and works as a counter of the free bytes in the queue.

**Note**

The callback is invoked from within the S-Locked system state.

**Parameters**

out	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
in	<i>bp</i>	pointer to a memory area allocated as queue buffer
in	<i>size</i>	size of the queue buffer

in	<i>only</i>	pointer to a callback function that is invoked when data is written to the queue. The value can be <code>NULL</code> .
in	<i>link</i>	application defined pointer

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

**7.11.4.7 void oqResetI ( output\_queue\_t \* oqp )**

Resets an output queue.

All the data in the output queue is erased and lost, any waiting thread is resumed with status `Q_RESET`.

**Note**

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

**Parameters**

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**7.11.4.8 msg\_t oqPutTimeout ( output\_queue\_t \* oqp, uint8\_t b, systime\_t timeout )**

Output queue write with timeout.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until there is space in the queue or a timeout occurs.

**Note**

The callback is invoked after writing the character into the buffer.

**Parameters**

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
in	<i>b</i>	the byte value to be written in the queue
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"><li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li><li>• <code>TIME_INFINITE</code> no timeout.</li></ul>

**Returns**

The operation status.

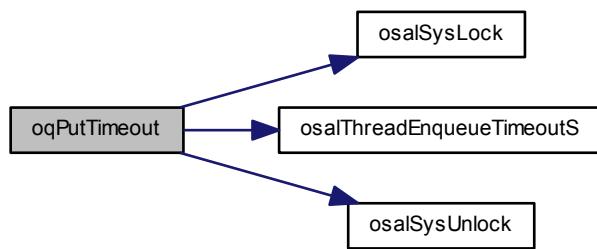
**Return values**

<code>Q_OK</code>	if the operation succeeded.
<code>Q_TIMEOUT</code>	if the specified time expired.
<code>Q_RESET</code>	if the queue has been reset.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.11.4.9 `msg_t oqGet( output_queue_t * oqp )`**

Output queue read.

A byte value is read from the low end of an output queue.

**Parameters**

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

**Returns**

The byte value from the queue.

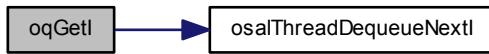
**Return values**

<i>Q_EMPTY</i>	if the queue is empty.
----------------	------------------------

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.11.4.10 `size_t oqWriteTimeout( output_queue_t * oqp, const uint8_t * bp, size_t n, systime_t timeout )`

Output queue write with timeout.

The function writes data from a buffer to an output queue. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

**Note**

The function is not atomic, if you need atomicity it is suggested to use a semaphore or a mutex for mutual exclusion.

The callback is invoked after writing each character into the buffer.

**Parameters**

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred, the value 0 is reserved
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <i>TIME_IMMEDIATE</i> immediate timeout.</li> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

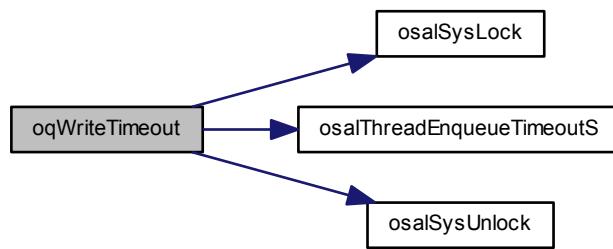
**Returns**

The number of bytes effectively transferred.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



## 7.12 Abstract Streams

### 7.12.1 Detailed Description

This module define an abstract interface for generic data streams. Note that no code is present, just abstract interfaces-like structures, you should look at the system as to a set of abstract C++ classes (even if written in C). This system has then advantage to make the access to data streams independent from the implementation logic. The stream interface can be used as base class for high level object types such as files, sockets, serial ports, pipes etc.

#### Macros

- `#define _base_sequential_stream_methods`  
*BaseSequentialStream specific methods.*
- `#define _base_sequential_stream_data`  
*BaseSequentialStream specific data.*

#### Streams return codes

- `#define STM_OK MSG_OK`
- `#define STM_TIMEOUT MSG_TIMEOUT`
- `#define STM_RESET MSG_RESET`

#### Macro Functions (BaseSequentialStream)

- `#define streamWrite(ip, bp, n) ((ip)->vmt->write(ip, bp, n))`  
*Sequential Stream write.*
- `#define streamRead(ip, bp, n) ((ip)->vmt->read(ip, bp, n))`  
*Sequential Stream read.*
- `#define streamPut(ip, b) ((ip)->vmt->put(ip, b))`  
*Sequential Stream blocking byte write.*
- `#define streamGet(ip) ((ip)->vmt->get(ip))`  
*Sequential Stream blocking byte read.*

#### Data Structures

- struct `BaseSequentialStreamVMT`  
*BaseSequentialStream virtual methods table.*
- struct `BaseSequentialStream`  
*Base stream class.*

### 7.12.2 Macro Definition Documentation

#### 7.12.2.1 `#define _base_sequential_stream_methods`

##### Value:

```
/* Stream write buffer method.*/
size_t (*write)(void *instance, const uint8_t *bp, size_t n);
/* Stream read buffer method.*/
size_t (*read)(void *instance, uint8_t *bp, size_t n);
/* Channel put method, blocking.*/
msg_t (*put)(void *instance, uint8_t b);
/* Channel get method, blocking.*/
msg_t (*get)(void *instance);
```



[BaseSequentialStream](#) specific methods.

#### 7.12.2.2 #define \_base\_sequential\_stream\_data

[BaseSequentialStream](#) specific data.

##### Note

It is empty because [BaseSequentialStream](#) is only an interface without implementation.

#### 7.12.2.3 #define streamWrite( ip, bp, n ) ((ip)->vmt->write(ip, bp, n))

Sequential Stream write.

The function writes data from a buffer to a stream.

##### Parameters

in	ip	pointer to a <a href="#">BaseSequentialStream</a> or derived class
in	bp	pointer to the data buffer
in	n	the maximum amount of data to be transferred

##### Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.12.2.4 #define streamRead( ip, bp, n ) ((ip)->vmt->read(ip, bp, n))

Sequential Stream read.

The function reads data from a stream into a buffer.

##### Parameters

in	ip	pointer to a <a href="#">BaseSequentialStream</a> or derived class
out	bp	pointer to the data buffer
in	n	the maximum amount of data to be transferred

##### Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.12.2.5 #define streamPut( ip, b ) ((ip)->vmt->put(ip, b))

Sequential Stream blocking byte write.

This function writes a byte value to a channel. If the channel is not ready to accept data then the calling thread is suspended.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseChannel</a> or derived class
in	<i>b</i>	the byte value to be written to the channel

**Returns**

The operation status.

**Return values**

<i>STM_OK</i>	if the operation succeeded.
<i>STM_RESET</i>	if an end-of-file condition has been met.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.12.2.6 #define streamGet( *ip* ) ((ip)->vmt->get(ip))**

Sequential Stream blocking byte read.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

**Parameters**

in	<i>ip</i>	pointer to a <a href="#">BaseChannel</a> or derived class
----	-----------	---

**Returns**

A byte value from the queue.

**Return values**

<i>STM_RESET</i>	if an end-of-file condition has been met.
------------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

## 7.13 I2C Driver

Generic I2C Driver.

### 7.13.1 Detailed Description

Generic I2C Driver.

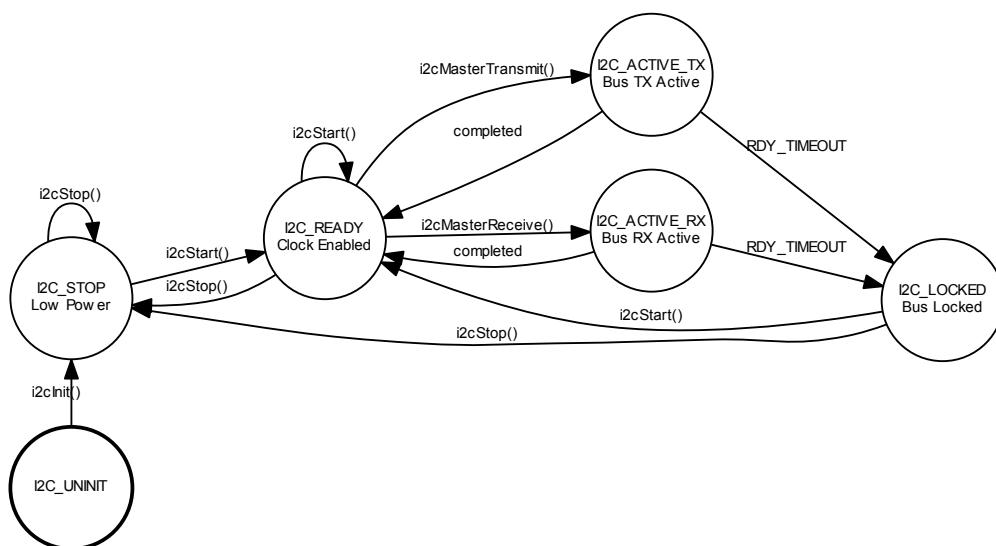
This module implements a generic I2C (Inter-Integrated Circuit) driver.

#### Precondition

In order to use the I2C driver the `HAL_USE_I2C` option must be enabled in `halconf.h`.

### 7.13.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



The driver is not thread safe for performance reasons, if you need to access the I2C bus from multiple threads then use the `i2cAcquireBus()` and `i2cReleaseBus()` APIs in order to gain exclusive access.

#### Macros

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`  
*Enables the mutual exclusion APIs on the I2C bus.*
- `#define _i2c_wakeup_isr(i2cp)`  
*Wakes up the waiting thread notifying no errors.*
- `#define _i2c_wakeup_error_isr(i2cp)`

- `#define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)`  
*Wrap i2cMasterTransmit function with TIME\_INFINITE timeout.*
- `#define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))`  
*Wrap i2cMasterReceiveTimeout function with TIME\_INFINITE timeout.*
- `#define i2c_lld_get_errors(i2cp) ((i2cp)->errors)`  
*Get errors from I2C driver.*

## I2C bus error conditions

- `#define I2C_NO_ERROR 0x00`  
*No error.*
- `#define I2C_BUS_ERROR 0x01`  
*Bus Error.*
- `#define I2C_ARBITRATION_LOST 0x02`  
*Arbitration Lost.*
- `#define I2C_ACK_FAILURE 0x04`  
*Acknowledge Failure.*
- `#define I2C_OVERRUN 0x08`  
*Overrun/Underrun.*
- `#define I2C_PEC_ERROR 0x10`  
*PEC Error in reception.*
- `#define I2C_TIMEOUT 0x20`  
*Hardware timeout.*
- `#define I2C_SMB_ALERT 0x40`  
*SMBus Alert.*

## PLATFORM configuration options

- `#define PLATFORM_I2C_USE_I2C1 FALSE`  
*I2C1 driver enable switch.*

## Typedefs

- `typedef uint16_t i2caddr_t`  
*Type representing an I2C address.*
- `typedef uint32_t i2cflags_t`  
*Type of I2C Driver condition flags.*
- `typedef struct I2CDriver I2CDriver`  
*Type of a structure representing an I2C driver.*

## Data Structures

- `struct I2CConfig`  
*Type of I2C driver configuration structure.*
- `struct I2CDriver`  
*Structure representing an I2C driver.*

## Functions

- void `i2cInit (void)`  
*I2C Driver initialization.*
- void `i2cObjectInit (I2CDriver *i2cp)`  
*Initializes the standard part of a `I2CDriver` structure.*
- void `i2cStart (I2CDriver *i2cp, const I2CConfig *config)`  
*Configures and activates the I2C peripheral.*
- void `i2cStop (I2CDriver *i2cp)`  
*Deactivates the I2C peripheral.*
- `i2cflags_t i2cGetErrors (I2CDriver *i2cp)`  
*Returns the errors mask associated to the previous operation.*
- `msg_t i2cMasterTimeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`  
*Sends data via the I2C bus.*
- `msg_t i2cMasterReceiveTimeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`  
*Receives data from the I2C bus.*
- void `i2cAcquireBus (I2CDriver *i2cp)`  
*Gains exclusive access to the I2C bus.*
- void `i2cReleaseBus (I2CDriver *i2cp)`  
*Releases exclusive access to the I2C bus.*
- void `i2c_lld_init (void)`  
*Low level I2C driver initialization.*
- void `i2c_lld_start (I2CDriver *i2cp)`  
*Configures and activates the I2C peripheral.*
- void `i2c_lld_stop (I2CDriver *i2cp)`  
*Deactivates the I2C peripheral.*
- `msg_t i2c_lld_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`  
*Receives data via the I2C bus as master.*
- `msg_t i2c_lld_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`  
*Transmits data via the I2C bus as master.*

## Enumerations

- enum `i2cstate_t` {
 `I2C_UNINIT` = 0, `I2C_STOP` = 1, `I2C_READY` = 2, `I2C_ACTIVE_TX` = 3,  
`I2C_ACTIVE_RX` = 4
 }
- Driver state machine possible states.*

## Variables

- `I2CDriver I2CD1`  
*I2C1 driver identifier.*

### 7.13.3 Macro Definition Documentation

#### 7.13.3.1 #define I2C\_NO\_ERROR 0x00

No error.

## 7.13.3.2 #define I2C\_BUS\_ERROR 0x01

Bus Error.

## 7.13.3.3 #define I2C\_ARBITRATION\_LOST 0x02

Arbitration Lost.

## 7.13.3.4 #define I2C\_ACK\_FAILURE 0x04

Acknowledge Failure.

## 7.13.3.5 #define I2C\_OVERRUN 0x08

Overrun/Underrun.

## 7.13.3.6 #define I2C\_PEC\_ERROR 0x10

PEC Error in reception.

## 7.13.3.7 #define I2C\_TIMEOUT 0x20

Hardware timeout.

## 7.13.3.8 #define I2C\_SMB\_ALERT 0x40

SMBus Alert.

## 7.13.3.9 #define I2C\_USE\_MUTUAL\_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

## 7.13.3.10 #define \_i2c\_wakeup\_isr( i2cp )

**Value:**

```
do {
    osalSysLockFromISR();
    \
    osalThreadResumeI(&(i2cp)->thread, MSG_OK);
    \
    osalSysUnlockFromISR();
} while(0)
```

Wakes up the waiting thread notifying no errors.

**Parameters**

in	i2cp	pointer to the <a href="#">I2CDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

7.13.3.11 #define \_i2c\_wakeup\_error\_isr( *i2cp* )

**Value:**

```
do {
    osalSysLockFromISR();
    \
    osalThreadResumeI (& (i2cp) ->thread, MSG_RESET);
    \
    osalSysUnlockFromISR();
} while(0)
```

Wakes up the waiting thread notifying errors.

**Parameters**

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

7.13.3.12 #define i2cMasterTransmit( *i2cp*, *addr*, *txbuf*, *txbytes*, *rxbuf*, *rxbytes* )

**Value:**

```
(i2cMasterTransmitTimeout(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes,
TIME_INFINITE)) \
```

Wrap i2cMasterTransmitTimeout function with TIME\_INFINITE timeout.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.13.3.13 #define i2cMasterReceive( *i2cp*, *addr*, *rxbuf*, *rxbytes* )(i2cMasterReceiveTimeout(*i2cp*, *addr*, *rxbuf*, *rxbytes*, TIME\_INFINITE))

Wrap i2cMasterReceiveTimeout function with TIME\_INFINITE timeout.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.13.3.14 #define PLATFORM\_I2C\_USE\_I2C1 FALSE

I2C1 driver enable switch.

If set to TRUE the support for I2C1 is included.

**Note**

The default is FALSE.

7.13.3.15 #define i2c\_lld\_get\_errors( *i2cp* ) ((i2cp)->errors)

Get errors from I2C driver.

**Parameters**

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.13.4 TYPEDOC Documentation****7.13.4.1 `typedef uint16_t i2caddr_t`**

Type representing an I2C address.

**7.13.4.2 `typedef uint32_t i2cflags_t`**

Type of I2C Driver condition flags.

**7.13.4.3 `typedef struct I2CDriver I2CDriver`**

Type of a structure representing an I2C driver.

**7.13.5 ENUMERATION Documentation****7.13.5.1 `enum i2cstate_t`**

Driver state machine possible states.

**Enumerator**

***I2C\_UNINIT*** Not initialized.

***I2C\_STOP*** Stopped.

***I2C\_READY*** Ready.

***I2C\_ACTIVE\_TX*** Transmitting.

***I2C\_ACTIVE\_RX*** Receiving.

**7.13.6 FUNCTION Documentation****7.13.6.1 `void i2clinit( void )`**

I2C Driver initialization.

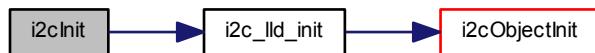
**Note**

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.13.6.2 void i2cObjectInit ( I2CDriver \* *i2cp* )

Initializes the standard part of a [I2CDriver](#) structure.

##### Parameters

out	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
-----	-------------	---

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.13.6.3 void i2cStart ( I2CDriver \* *i2cp*, const I2CConfig \* *config* )

Configures and activates the I2C peripheral.

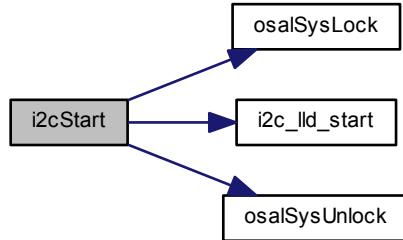
##### Parameters

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
in	<i>config</i>	pointer to the <a href="#">I2CConfig</a> object

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.13.6.4 void i2cStop ( I2CDriver \* *i2cp* )

Deactivates the I2C peripheral.

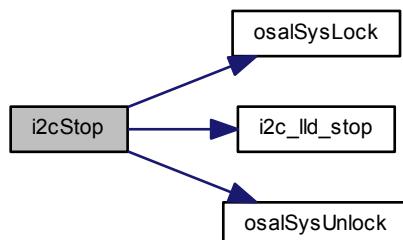
##### Parameters

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
----	-------------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.13.6.5 i2cflags\_t i2cGetErrors ( I2CDriver \* *i2cp* )

Returns the errors mask associated to the previous operation.

**Parameters**

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
----	-------------	---

**Returns**

The errors mask.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.13.6.6 *msg\_t i2cMasterTransmitTimeout( I2CDriver \*i2cp, i2caddr\_t addr, const uint8\_t \*txbuf, size\_t txbytes, uint8\_t \*rxbuf, size\_t rxbytes, systime\_t timeout )***

Sends data via the I2C bus.

Function designed to realize "read-through-write" transfer paradigm. If you want transmit data without any further read, than set **rxbytes** field to 0.

**Parameters**

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
in	<i>addr</i>	slave device address (7 bits) without R/W bit
in	<i>txbuf</i>	pointer to transmit buffer
in	<i>txbytes</i>	number of bytes to be transmitted
out	<i>rxbuf</i>	pointer to receive buffer
in	<i>rxbytes</i>	number of bytes to be received, set it to 0 if you want transmit only
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

**Returns**

The operation status.

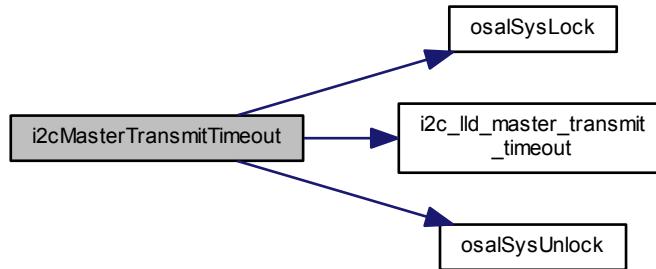
**Return values**

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more I2C errors occurred, the errors can be retrieved using <a href="#">i2cGetErrors()</a> .
<i>MSG_TIMEOUT</i>	if a timeout occurred before operation end.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.13.6.7 `msg_t i2cMasterReceiveTimeout( I2CDriver * i2cp, i2caddr_t addr, uint8_t * rdbuf, size_t rxbytes, systime_t timeout )`

Receives data from the I2C bus.

##### Parameters

in	<code>i2cp</code>	pointer to the <a href="#">I2CDriver</a> object
in	<code>addr</code>	slave device address (7 bits) without R/W bit
out	<code>rdbuf</code>	pointer to receive buffer
in	<code>rxbytes</code>	number of bytes to be received
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

##### Returns

The operation status.

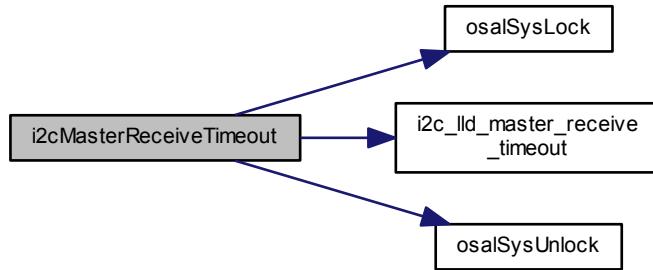
##### Return values

<code>MSG_OK</code>	if the function succeeded.
<code>MSG_RESET</code>	if one or more I2C errors occurred, the errors can be retrieved using <a href="#">i2cGetErrors()</a> .
<code>MSG_TIMEOUT</code>	if a timeout occurred before operation end.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.13.6.8 void i2cAcquireBus ( I2CDriver \* i2cp )

Gains exclusive access to the I2C bus.

This function tries to gain ownership to the I2C bus, if the bus is already being used then the invoking thread is queued.

##### Precondition

In order to use this function the option `I2C_USE_MUTUAL_EXCLUSION` must be enabled.

##### Parameters

in	<code>i2cp</code>	pointer to the <code>I2CDriver</code> object
----	-------------------	--

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.13.6.9 void i2cReleaseBus ( I2CDriver \* i2cp )

Releases exclusive access to the I2C bus.

##### Precondition

In order to use this function the option `I2C_USE_MUTUAL_EXCLUSION` must be enabled.

**Parameters**

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.13.6.10 void i2c\_lld\_init( void )**

Low level I2C driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.13.6.11 void i2c\_lld\_start( I2CDriver \* i2cp )**

Configures and activates the I2C peripheral.

**Parameters**

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.13.6.12 void i2c\_lld\_stop( I2CDriver \* i2cp )**

Deactivates the I2C peripheral.

## Parameters

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
----	-------------	---

## Function Class:

Not an API, this function is for internal use only.

**7.13.6.13 msg\_t i2c\_lld\_master\_receive\_timeout ( I2CDriver \* *i2cp*, i2caddr\_t *addr*, uint8\_t \* *rxbuf*, size\_t *rxbytes*, systime\_t *timeout* )**

Receives data via the I2C bus as master.

## Parameters

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
in	<i>addr</i>	slave device address
out	<i>rxbuf</i>	pointer to the receive buffer
in	<i>rxbytes</i>	number of bytes to be received
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

## Returns

The operation status.

## Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more I2C errors occurred, the errors can be retrieved using <a href="#">i2cGetErrors()</a> .
<i>MSG_TIMEOUT</i>	if a timeout occurred before operation end. <b>After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.</b>

## Function Class:

Not an API, this function is for internal use only.

**7.13.6.14 msg\_t i2c\_lld\_master\_transmit\_timeout ( I2CDriver \* *i2cp*, i2caddr\_t *addr*, const uint8\_t \* *txbuf*, size\_t *txbytes*, uint8\_t \* *rxbuf*, size\_t *rxbytes*, systime\_t *timeout* )**

Transmits data via the I2C bus as master.

## Parameters

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
in	<i>addr</i>	slave device address
in	<i>txbuf</i>	pointer to the transmit buffer
in	<i>txbytes</i>	number of bytes to be transmitted
out	<i>rxbuf</i>	pointer to the receive buffer

in	<i>rxbytes</i>	number of bytes to be received
in	<i>timeout</i>	<p>the number of ticks before the operation timeouts, the following special values are allowed:</p> <ul style="list-style-type: none"> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

**Returns**

The operation status.

**Return values**

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more I2C errors occurred, the errors can be retrieved using <a href="#"><i>i2cGetErrors()</i></a> .
<i>MSG_TIMEOUT</i>	if a timeout occurred before operation end. <b>After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.</b>

**Function Class:**

Not an API, this function is for internal use only.

## 7.13.7 Variable Documentation

### 7.13.7.1 I2CDriver I2CD1

I2C1 driver identifier.

## 7.14 I2S Driver

Generic I2S Driver.

### 7.14.1 Detailed Description

Generic I2S Driver.

This module implements a generic I2S driver.

#### Precondition

In order to use the I2S driver the `HAL_USE_I2S` option must be enabled in `halconf.h`.

### 7.14.2 Driver State Machine

#### I2S modes

- `#define I2S_MODE_SLAVE 0`
- `#define I2S_MODE_MASTER 1`

#### Macro Functions

- `#define i2sStartExchange(i2sp)`  
*Starts a I2S data exchange.*
- `#define i2sStopExchange(i2sp)`  
*Stops the ongoing data exchange.*
- `#define _i2s_isr_half_code(i2sp)`  
*Common ISR code, half buffer event.*
- `#define _i2s_isr_full_code(i2sp)`  
*Common ISR code.*

#### PLATFORM configuration options

- `#define PLATFORM_I2S_USE_I2S1 FALSE`  
*I2SD1 driver enable switch.*

#### Typedefs

- `typedef struct I2SDriver I2SDriver`  
*Type of a structure representing an I2S driver.*
- `typedef void(* i2scallback_t) (I2SDriver *i2sp, size_t offset, size_t n)`  
*I2S notification callback type.*

#### Data Structures

- `struct I2SConfig`  
*Driver configuration structure.*
- `struct I2SDriver`  
*Structure representing an I2S driver.*

## Functions

- void `i2sInit` (void)
 

*I2S Driver initialization.*
- void `i2sObjectInit` (`I2SDriver` \*`i2sp`)
 

*Initializes the standard part of a `I2SDriver` structure.*
- void `i2sStart` (`I2SDriver` \*`i2sp`, const `I2SConfig` \*`config`)
 

*Configures and activates the I2S peripheral.*
- void `i2sStop` (`I2SDriver` \*`i2sp`)
 

*Deactivates the I2S peripheral.*
- void `i2sStartExchange` (`I2SDriver` \*`i2sp`)
 

*Starts a I2S data exchange.*
- void `i2sStopExchange` (`I2SDriver` \*`i2sp`)
 

*Stops the ongoing data exchange.*
- void `i2s_lld_init` (void)
 

*Low level I2S driver initialization.*
- void `i2s_lld_start` (`I2SDriver` \*`i2sp`)
 

*Configures and activates the I2S peripheral.*

## Enumerations

- enum `i2sstate_t` {
 

`I2S_UNINIT` = 0, `I2S_STOP` = 1, `I2S_READY` = 2, `I2S_ACTIVE` = 3,  
`I2S_COMPLETE` = 4 }

*Driver state machine possible states.*

## Variables

- `I2SDriver` `I2SD1`

*I2S2 driver identifier.*

### 7.14.3 Macro Definition Documentation

#### 7.14.3.1 #define `i2sStartExchange`( `i2sp` )

##### Value:

```
{
  i2s_lld_start_exchange(i2sp);
  (i2sp)->state = I2S_ACTIVE;
}
```

Starts a I2S data exchange.

##### Parameters

in	<code>i2sp</code>	pointer to the <code>I2SDriver</code> object
----	-------------------	--

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.14.3.2 #define i2sStopExchange( *i2sp* )**Value:**

```
{
    i2s_lld_stop_exchange(i2sp);
    (i2sp)->state = I2S_READY;
}
```

Stops the ongoing data exchange.

The ongoing data exchange, if any, is stopped, if the driver was not active the function does nothing.

**Parameters**

in	<i>i2sp</i>	pointer to the <a href="#">I2SDriver</a> object
----	-------------	---

**Function Class:**

This is an **I-Class API**, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.14.3.3 #define \_i2s\_isr\_half\_code( *i2sp* )**Value:**

```
{
    if ((i2sp)>config->end_cb != NULL) {
        (i2sp)>config->end_cb(i2sp, 0, (i2sp)>config->size / 2);
    }
}
```

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>i2sp</i>	pointer to the <a href="#">I2CDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

7.14.3.4 #define \_i2s\_isr\_full\_code( *i2sp* )**Value:**

```
{
    if ((i2sp)>config->end_cb) {
        (i2sp)>state = I2S_COMPLETE;
        (i2sp)>config->end_cb(i2sp,
            (i2sp)>config->size / 2,
            (i2sp)>config->size / 2);
        if ((i2sp)>state == I2S_COMPLETE)
            (i2sp)>state = I2S_READY;
    }
    else
        (i2sp)>state = I2S_READY;
}
```

Common ISR code.

This code handles the portable part of the ISR code:

- Callback invocation.
- Driver state transitions.

#### Note

This macro is meant to be used in the low level drivers implementation only.

#### Parameters

in	<i>i2sp</i>	pointer to the <a href="#">I2CDriver</a> object
----	-------------	---

#### Function Class:

Not an API, this function is for internal use only.

### 7.14.3.5 #define PLATFORM\_I2S\_USE\_I2S1 FALSE

I2SD1 driver enable switch.

If set to TRUE the support for I2S1 is included.

#### Note

The default is FALSE.

### 7.14.4 Typedef Documentation

#### 7.14.4.1 typedef struct I2SDriver I2SDriver

Type of a structure representing an I2S driver.

#### 7.14.4.2 typedef void(\* i2scallback\_t) (I2SDriver \*i2sp, size\_t offset, size\_t n)

I2S notification callback type.

#### Parameters

in	<i>i2sp</i>	pointer to the <a href="#">I2SDriver</a> object
in	<i>offset</i>	offset in buffers of the data to read/write
in	<i>n</i>	number of samples to read/write

### 7.14.5 Enumeration Type Documentation

#### 7.14.5.1 enum i2sstate\_t

Driver state machine possible states.

#### Enumerator

***I2S\_UNINIT*** Not initialized.

***I2S\_STOP*** Stopped.

***I2S\_READY*** Ready.

***I2S\_ACTIVE*** Active.

***I2S\_COMPLETE*** Transmission complete.

## 7.14.6 Function Documentation

### 7.14.6.1 void i2sInit( void )

I2S Driver initialization.

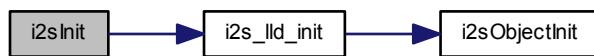
#### Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.14.6.2 void i2sObjectInit( I2SDriver \* i2sp )

Initializes the standard part of a `I2SDriver` structure.

#### Parameters

out	<i>i2sp</i>	pointer to the <code>I2SDriver</code> object
-----	-------------	--

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

### 7.14.6.3 void i2sStart( I2SDriver \* i2sp, const I2SConfig \* config )

Configures and activates the I2S peripheral.

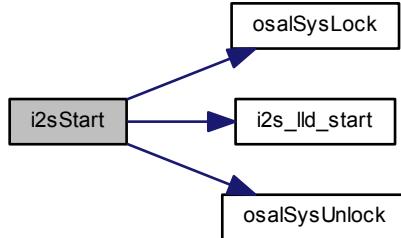
#### Parameters

in	<i>i2sp</i>	pointer to the <code>I2SDriver</code> object
in	<i>config</i>	pointer to the <code>I2SConfig</code> object

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.14.6.4 void i2sStop ( I2SDriver \* i2sp )

Deactivates the I2S peripheral.

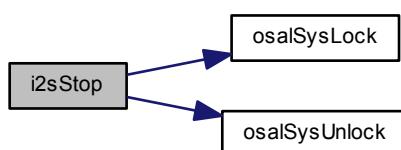
##### Parameters

in	i2sp	pointer to the <a href="#">I2SDriver</a> object
----	------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.14.6.5 void i2sStartExchange ( I2SDriver \* i2sp )

Starts a I2S data exchange.

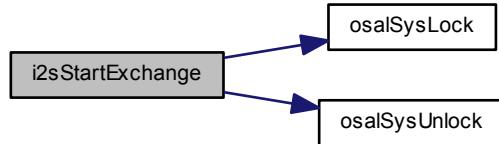
##### Parameters

in	i2sp	pointer to the <a href="#">I2SDriver</a> object
----	------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.14.6.6 void i2sStopExchange ( I2SDriver \* i2sp )

Stops the ongoing data exchange.

The ongoing data exchange, if any, is stopped, if the driver was not active the function does nothing.

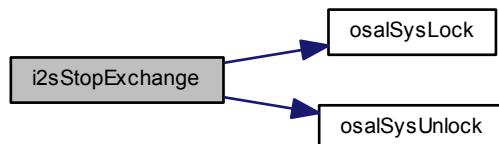
##### Parameters

in	i2sp	pointer to the <a href="#">I2SDriver</a> object
----	------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.14.6.7 void i2s\_lld\_init ( void )

Low level I2S driver initialization.

##### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.14.6.8 void i2s\_lld\_start ( I2SDriver \* i2sp )

Configures and activates the I2S peripheral.

**Parameters**

in	i2sp	pointer to the <a href="#">I2SDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

Deactivates the I2S peripheral.

**Parameters**

in	i2sp	pointer to the <a href="#">I2SDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

Starts a I2S data exchange.

**Parameters**

in	i2sp	pointer to the <a href="#">I2SDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

Stops the ongoing data exchange.

The ongoing data exchange, if any, is stopped, if the driver was not active the function does nothing.

**Parameters**

in	i2sp	pointer to the <a href="#">I2SDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

#### 7.14.7 Variable Documentation

##### 7.14.7.1 I2SDriver I2SD1

I2S2 driver identifier.

## 7.15 ICU Driver

Generic ICU Driver.

### 7.15.1 Detailed Description

Generic ICU Driver.

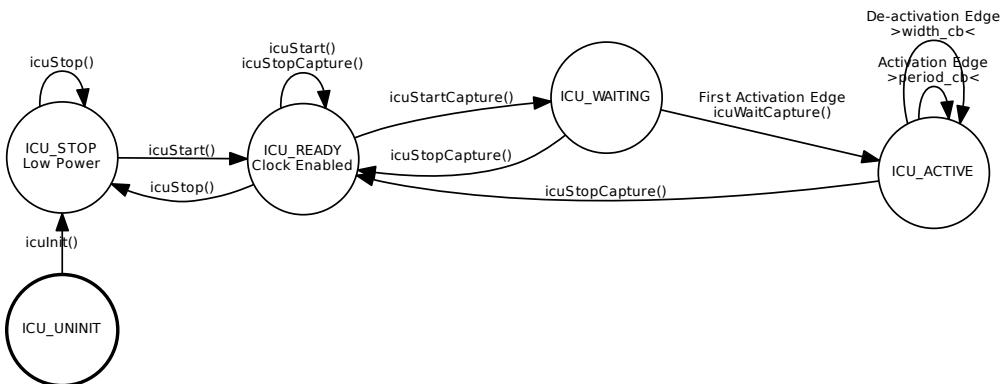
This module implements a generic ICU (Input Capture Unit) driver. The purpose of the driver is to measure period and duty cycle of an input digital signal (PWM input).

#### Precondition

In order to use the ICU driver the `HAL_USE_ICU` option must be enabled in `halconf.h`.

### 7.15.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



### 7.15.3 ICU Operations.

This driver abstracts a generic Input Capture Unit composed of:

- A clock prescaler.
- A main up counter.
- Two capture registers triggered by the rising and falling edges on the sampled input.

The ICU unit can be programmed to synchronize on the rising or falling edge of the sample input:

- **ICU\_INPUT\_ACTIVE\_HIGH**, a rising edge is the start signal.

- **ICU\_INPUT\_ACTIVE\_LOW**, a falling edge is the start signal.

Callbacks are optionally invoked when:

- On the PWM de-activation edge.
- On the PWM activation edge, measurements for the previous cycle are available from this callback and can be retrieved using `icuGetPeriodX()` and `icuGetWidthX()`.

## Macros

- `#define icu_lld_get_width(icup) 0`  
*Returns the width of the latest pulse.*
- `#define icu_lld_get_period(icup) 0`  
*Returns the width of the latest cycle.*
- `#define icu_lld_are_notifications_enabled(icup) false`  
*Check on notifications status.*

## Macro Functions

- `#define icuStartCapture1(icup)`  
*Starts the input capture.*
- `#define icuStopCapture1(icup)`  
*Stops the input capture.*
- `#define icuEnableNotifications1(icup) icu_lld_enable_notifications(icup)`  
*Enables notifications.*
- `#define icuDisableNotifications1(icup) icu_lld_disable_notifications(icup)`  
*Disables notifications.*
- `#define icuAreNotificationsEnabledX(icup) icu_lld_are_notifications_enabled(icup)`  
*Check on notifications status.*
- `#define icuGetWidthX(icup) icu_lld_get_width(icup)`  
*Returns the width of the latest pulse.*
- `#define icuGetPeriodX(icup) icu_lld_get_period(icup)`  
*Returns the width of the latest cycle.*

## Low level driver helper macros

- `#define _icu_isr_invoke_width_cb(icup)`  
*Common ISR code, ICU width event.*
- `#define _icu_isr_invoke_period_cb(icup)`  
*Common ISR code, ICU period event.*
- `#define _icu_isr_invoke_overflow_cb(icup)`  
*Common ISR code, ICU timer overflow event.*

## PLATFORM configuration options

- `#define PLATFORM_ICU_USE_ICU1 FALSE`  
*ICUD1 driver enable switch.*

## Typedefs

- **typedef struct ICUDriver ICUDriver**  
*Type of a structure representing an ICU driver.*
- **typedef void(\* icucallback\_t) (ICUDriver \*icup)**  
*ICU notification callback type.*
- **typedef uint32\_t icufreq\_t**  
*ICU frequency type.*
- **typedef uint32\_t icucnt\_t**  
*ICU counter type.*

## Data Structures

- **struct ICUConfig**  
*Driver configuration structure.*
- **struct ICUDriver**  
*Structure representing an ICU driver.*

## Functions

- **void icuInit (void)**  
*ICU Driver initialization.*
- **void icuObjectInit (ICUDriver \*icup)**  
*Initializes the standard part of a `ICUDriver` structure.*
- **void icuStart (ICUDriver \*icup, const ICUConfig \*config)**  
*Configures and activates the ICU peripheral.*
- **void icuStop (ICUDriver \*icup)**  
*Deactivates the ICU peripheral.*
- **void icuStartCapture (ICUDriver \*icup)**  
*Starts the input capture.*
- **bool icuWaitCapture (ICUDriver \*icup)**  
*Waits for a completed capture.*
- **void icuStopCapture (ICUDriver \*icup)**  
*Stops the input capture.*
- **void icuEnableNotifications (ICUDriver \*icup)**  
*Enables notifications.*
- **void icuDisableNotifications (ICUDriver \*icup)**  
*Disables notifications.*
- **void icu\_lld\_init (void)**  
*Low level ICU driver initialization.*
- **void icu\_lld\_start (ICUDriver \*icup)**  
*Configures and activates the ICU peripheral.*
- **void icu\_lld\_stop (ICUDriver \*icup)**  
*Deactivates the ICU peripheral.*
- **void icu\_lld\_start\_capture (ICUDriver \*icup)**  
*Starts the input capture.*
- **bool icu\_lld\_wait\_capture (ICUDriver \*icup)**  
*Waits for a completed capture.*
- **void icu\_lld\_stop\_capture (ICUDriver \*icup)**  
*Stops the input capture.*

- void `icu_lld_enable_notifications` (`ICUDriver` \*`icup`)  
*Enables notifications.*
- void `icu_lld_disable_notifications` (`ICUDriver` \*`icup`)  
*Disables notifications.*

## Enumerations

- enum `icustate_t` {
   
`ICU_UNINIT` = 0, `ICU_STOP` = 1, `ICU_READY` = 2, `ICU_WAITING` = 3,
   
`ICU_ACTIVE` = 4 }
   
*Driver state machine possible states.*
- enum `icemode_t` { `ICU_INPUT_ACTIVE_HIGH` = 0, `ICU_INPUT_ACTIVE_LOW` = 1 }
   
*ICU driver mode.*

## Variables

- `ICUDriver` `ICUD1`  
*ICUD1 driver identifier.*

### 7.15.4 Macro Definition Documentation

#### 7.15.4.1 #define `icuStartCapture`( `icup` )

##### Value:

```
do {
    icu_lld_start_capture(icup);
    (icup)>state = ICU_WAITING;
} while (false)
```

Starts the input capture.

##### Parameters

in	<code>icup</code>	pointer to the <code>ICUDriver</code> object
----	-------------------	--

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

#### 7.15.4.2 #define `icuStopCapture`( `icup` )

##### Value:

```
do {
    icu_lld_stop_capture(icup);
    (icup)>state = ICU_READY;
} while (false)
```

Stops the input capture.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.15.4.3 #define icuEnableNotifications( *icup* ) icu\_lld\_enable\_notifications(*icup*)**

Enables notifications.

**Precondition**

The ICU unit must have been activated using [icuStart\(\)](#).

**Note**

If the notification is already enabled then the call has no effect.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.15.4.4 #define icuDisableNotifications( *icup* ) icu\_lld\_disable\_notifications(*icup*)**

Disables notifications.

**Precondition**

The ICU unit must have been activated using [icuStart\(\)](#).

**Note**

If the notification is already disabled then the call has no effect.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.15.4.5 #define icuAreNotificationsEnabledX( *icup* ) icu\_lld\_are\_notifications\_enabled(*icup*)**

Check on notifications status.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Returns**

The notifications status.

**Return values**

<i>false</i>	if notifications are not enabled.
<i>true</i>	if notifications are enabled.

**Function Class:**

Not an API, this function is for internal use only.

**7.15.4.6 #define icuGetWidthX( *icup* ) icu\_lld\_get\_width(*icup*)**

Returns the width of the latest pulse.

The pulse width is defined as number of ticks between the start edge and the stop edge.

**Note**

This function is meant to be invoked from the width capture callback.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Returns**

The number of ticks.

**Function Class:**

This is an **X-Class** API, this function can be invoked from any context.

**7.15.4.7 #define icuGetPeriodX( *icup* ) icu\_lld\_get\_period(*icup*)**

Returns the width of the latest cycle.

The cycle width is defined as number of ticks between a start edge and the next start edge.

**Note**

This function is meant to be invoked from the width capture callback.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Returns**

The number of ticks.

**Function Class:**

This is an **X-Class** API, this function can be invoked from any context.

7.15.4.8 #define \_icu\_isr\_invoke\_width\_cb( *icup* )**Value:**

```
do {
    if (((icup)->state == ICU_ACTIVE) &&
        ((icup)->config->width_cb != NULL))
        (icup)->config->width_cb(icup);
} while (0)
```

Common ISR code, ICU width event.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

7.15.4.9 #define \_icu\_isr\_invoke\_period\_cb( *icup* )**Value:**

```
do {
    if (((icup)->state == ICU_ACTIVE) &&
        ((icup)->config->period_cb != NULL))
        (icup)->config->period_cb(icup);
    (icup)->state = ICU_ACTIVE;
} while (0)
```

Common ISR code, ICU period event.

**Note**

A period event brings the driver into the `ICU_ACTIVE` state.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

7.15.4.10 #define \_icu\_isr\_invoke\_overflow\_cb( *icup* )**Value:**

```
do {
    (icup)->config->overflow_cb(icup);
    (icup)->state = ICU_WAITING;
} while (0)
```

Common ISR code, ICU timer overflow event.

**Note**

An overflow always brings the driver back to the `ICU_WAITING` state.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.15.4.11 #define PLATFORM\_ICU\_USE\_ICU1 FALSE**

ICUD1 driver enable switch.

If set to TRUE the support for ICUD1 is included.

**Note**

The default is FALSE.

**7.15.4.12 #define icu\_lld\_get\_width( *icup* ) 0**

Returns the width of the latest pulse.

The pulse width is defined as number of ticks between the start edge and the stop edge.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Returns**

The number of ticks.

**Function Class:**

Not an API, this function is for internal use only.

**7.15.4.13 #define icu\_lld\_get\_period( *icup* ) 0**

Returns the width of the latest cycle.

The cycle width is defined as number of ticks between a start edge and the next start edge.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Returns**

The number of ticks.

**Function Class:**

Not an API, this function is for internal use only.

**7.15.4.14 #define icu\_lld\_are\_notifications\_enabled( *icup* ) false**

Check on notifications status.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Returns**

The notifications status.

**Return values**

<i>false</i>	if notifications are not enabled.
<i>true</i>	if notifications are enabled.

**Function Class:**

Not an API, this function is for internal use only.

**7.15.5 Typedef Documentation****7.15.5.1 `typedef struct ICUDriver ICUDriver`**

Type of a structure representing an ICU driver.

**7.15.5.2 `typedef void(* icucallback_t)(ICUDriver *icup)`**

ICU notification callback type.

**Parameters**

in	<i>icup</i>	pointer to a <a href="#">ICUDriver</a> object
----	-------------	---

**7.15.5.3 `typedef uint32_t icufreq_t`**

ICU frequency type.

**7.15.5.4 `typedef uint32_t icucnt_t`**

ICU counter type.

**7.15.6 Enumeration Type Documentation****7.15.6.1 `enum icustate_t`**

Driver state machine possible states.

**Enumerator**

***ICU\_UNINIT*** Not initialized.

***ICU\_STOP*** Stopped.

***ICU\_READY*** Ready.

***ICU\_WAITING*** Waiting for first front.

***ICU\_ACTIVE*** First front detected.

### 7.15.6.2 enum icumode\_t

ICU driver mode.

Enumerator

***ICU\_INPUT\_ACTIVE\_HIGH*** Trigger on rising edge.

***ICU\_INPUT\_ACTIVE\_LOW*** Trigger on falling edge.

## 7.15.7 Function Documentation

### 7.15.7.1 void iculinit( void )

ICU Driver initialization.

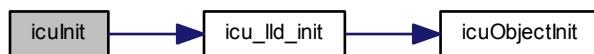
Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.15.7.2 void icuObjectInit( ICUDriver \* icup )

Initializes the standard part of a [ICUDriver](#) structure.

Parameters

out	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
-----	-------------	---

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

### 7.15.7.3 void icuStart( ICUDriver \* icup, const ICUConfig \* config )

Configures and activates the ICU peripheral.

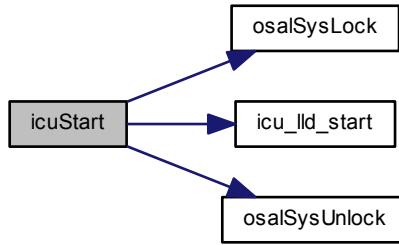
Parameters

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
in	<i>config</i>	pointer to the <a href="#">ICUConfig</a> object

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.15.7.4 void icuStop ( ICUDriver \* *icup* )**

Deactivates the ICU peripheral.

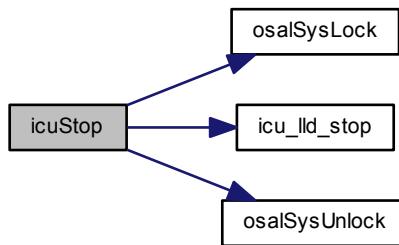
**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.15.7.5 void icuStartCapture ( **ICUDriver** \* *icup* )

Starts the input capture.

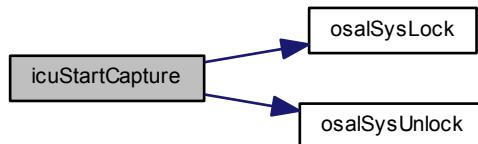
**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.15.7.6 bool icuWaitCapture ( [ICUDriver](#) \* *icup* )**

Waits for a completed capture.

**Note**

The operation could be performed in polled mode depending on.  
In order to use this function notifications must be disabled.

**Precondition**

The driver must be in `ICU_WAITING` or `ICU_ACTIVE` states.

**Postcondition**

After the capture is available the driver is in `ICU_ACTIVE` state. If a capture fails then the driver is in `ICU_WAITING` state.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Returns**

The capture status.

**Return values**

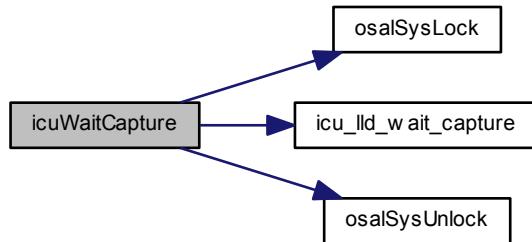
<i>false</i>	if the capture is successful.
--------------	-------------------------------

<code>true</code>	if a timer overflow occurred.
-------------------	-------------------------------

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.15.7.7 void icuStopCapture ( ICUDriver \* *icup* )**

Stops the input capture.

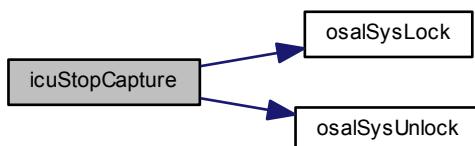
**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.15.7.8 void icuEnableNotifications ( ICUDriver \* *icup* )**

Enables notifications.

**Precondition**

The ICU unit must have been activated using `icuStart()`.

**Note**

If the notification is already enabled then the call has no effect.

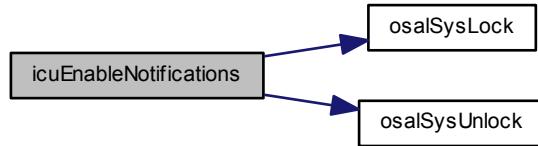
**Parameters**

in	<i>icup</i>	pointer to the <code>ICUDriver</code> object
----	-------------	--

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.15.7.9 void icuDisableNotifications ( ICUDriver \* *icup* )**

Disables notifications.

**Precondition**

The ICU unit must have been activated using `icuStart()`.

**Note**

If the notification is already disabled then the call has no effect.

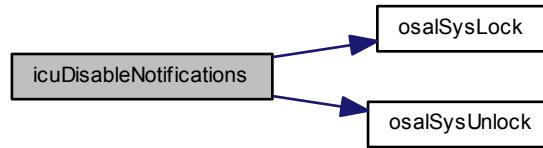
**Parameters**

in	<i>icup</i>	pointer to the <code>ICUDriver</code> object
----	-------------	--

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.15.7.10 void icu\_lld\_init( void )

Low level ICU driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.15.7.11 void icu\_lld\_start( ICUDriver \* icup )

Configures and activates the ICU peripheral.

**Parameters**

in	icup	pointer to the <a href="#">ICUDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

#### 7.15.7.12 void icu\_lld\_stop( ICUDriver \* icup )

Deactivates the ICU peripheral.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.15.7.13 void icu\_lld\_start\_capture ( ICUDriver \* *icup* )**

Starts the input capture.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.15.7.14 bool icu\_lld\_wait\_capture ( ICUDriver \* *icup* )**

Waits for a completed capture.

**Note**

The operation is performed in polled mode.

In order to use this function notifications must be disabled.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Returns**

The capture status.

**Return values**

<i>false</i>	if the capture is successful.
<i>true</i>	if a timer overflow occurred.

**Function Class:**

Not an API, this function is for internal use only.

**7.15.7.15 void icu\_lld\_stop\_capture ( ICUDriver \* *icup* )**

Stops the input capture.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.15.7.16 void icu\_lld\_enable\_notifications ( ICUDriver \* *icup* )**

Enables notifications.

**Precondition**

The ICU unit must have been activated using [icuStart \(\)](#).

**Note**

If the notification is already enabled then the call has no effect.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.15.7.17 void icu\_lld\_disable\_notifications ( ICUDriver \* *icup* )**

Disables notifications.

**Precondition**

The ICU unit must have been activated using [icuStart \(\)](#).

**Note**

If the notification is already disabled then the call has no effect.

**Parameters**

in	<i>icup</i>	pointer to the <a href="#">ICUDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.15.8 Variable Documentation****7.15.8.1 ICUDriver ICUD1**

ICUD1 driver identifier.

**Note**

The driver ICUD1 allocates the complex timer TIM1 when enabled.

## 7.16 MAC Driver

Generic MAC Driver.

### 7.16.1 Detailed Description

Generic MAC Driver.

This module implements a generic MAC (Media Access Control) driver for Ethernet controllers.

#### Precondition

In order to use the MAC driver the `HAL_USE_MAC` option must be enabled in `halconf.h`.

#### Macros

- `#define MAC_SUPPORTS_ZERO_COPY TRUE`  
*This implementation supports the zero-copy mode API.*

#### MAC configuration options

- `#define MAC_USE_ZERO_COPY FALSE`  
*Enables an event sources for incoming packets.*
- `#define MAC_USE_EVENTS TRUE`  
*Enables an event sources for incoming packets.*

#### Macro Functions

- `#define macGetReceiveEventSource(macp) (&(macp)->rdevent)`  
*Returns the received frames event source.*
- `#define macWriteTransmitDescriptor(tdp, buf, size) mac_lld_write_transmit_descriptor(tdp, buf, size)`  
*Writes to a transmit descriptor's stream.*
- `#define macReadReceiveDescriptor(rdp, buf, size) mac_lld_read_receive_descriptor(rdp, buf, size)`  
*Reads from a receive descriptor's stream.*
- `#define macGetNextTransmitBuffer(tdp, size, sizep) mac_lld_get_next_transmit_buffer(tdp, size, sizep)`  
*Returns a pointer to the next transmit buffer in the descriptor chain.*
- `#define macGetNextReceiveBuffer(rdp, sizep) mac_lld_get_next_receive_buffer(rdp, sizep)`  
*Returns a pointer to the next receive buffer in the descriptor chain.*

#### PLATFORM configuration options

- `#define PLATFORM_MAC_USE_MAC1 FALSE`  
*MAC driver enable switch.*

#### Typedefs

- `typedef struct MACDriver MACDriver`  
*Type of a structure representing a MAC driver.*

## Data Structures

- struct **MACConfig**  
*Driver configuration structure.*
- struct **MACDriver**  
*Structure representing a MAC driver.*
- struct **MACTransmitDescriptor**  
*Structure representing a transmit descriptor.*
- struct **MACReceiveDescriptor**  
*Structure representing a receive descriptor.*

## Functions

- void **macInit** (void)  
*MAC Driver initialization.*
- void **macObjectInit** (**MACDriver** \*macp)  
*Initialize the standard part of a **MACDriver** structure.*
- void **macStart** (**MACDriver** \*macp, const **MACConfig** \*config)  
*Configures and activates the MAC peripheral.*
- void **macStop** (**MACDriver** \*macp)  
*Deactivates the MAC peripheral.*
- **msg\_t** **macWaitTransmitDescriptor** (**MACDriver** \*macp, **MACTransmitDescriptor** \*tdp, **systime\_t** timeout)  
*Allocates a transmission descriptor.*
- void **macReleaseTransmitDescriptor** (**MACTransmitDescriptor** \*tdp)  
*Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.*
- **msg\_t** **macWaitReceiveDescriptor** (**MACDriver** \*macp, **MACReceiveDescriptor** \*rdp, **systime\_t** timeout)  
*Waits for a received frame.*
- void **macReleaseReceiveDescriptor** (**MACReceiveDescriptor** \*rdp)  
*Releases a receive descriptor.*
- bool **macPollLinkStatus** (**MACDriver** \*macp)  
*Updates and returns the link status.*
- void **mac\_lld\_init** (void)  
*Low level MAC initialization.*
- void **mac\_lld\_start** (**MACDriver** \*macp)  
*Configures and activates the MAC peripheral.*
- void **mac\_lld\_stop** (**MACDriver** \*macp)  
*Deactivates the MAC peripheral.*
- **msg\_t** **mac\_lld\_get\_transmit\_descriptor** (**MACDriver** \*macp, **MACTransmitDescriptor** \*tdp)  
*Returns a transmission descriptor.*
- void **mac\_lld\_release\_transmit\_descriptor** (**MACTransmitDescriptor** \*tdp)  
*Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.*
- **msg\_t** **mac\_lld\_get\_receive\_descriptor** (**MACDriver** \*macp, **MACReceiveDescriptor** \*rdp)  
*Returns a receive descriptor.*
- void **mac\_lld\_release\_receive\_descriptor** (**MACReceiveDescriptor** \*rdp)  
*Releases a receive descriptor.*
- bool **mac\_lld\_poll\_link\_status** (**MACDriver** \*macp)  
*Updates and returns the link status.*
- **size\_t** **mac\_lld\_write\_transmit\_descriptor** (**MACTransmitDescriptor** \*tdp, **uint8\_t** \*buf, **size\_t** size)  
*Writes to a transmit descriptor's stream.*
- **size\_t** **mac\_lld\_read\_receive\_descriptor** (**MACReceiveDescriptor** \*rdp, **uint8\_t** \*buf, **size\_t** size)  
*Reads from a receive descriptor's stream.*

- `uint8_t * mac_lld_get_next_transmit_buffer (MACTransmitDescriptor *tdp, size_t size, size_t *sizep)`  
*Returns a pointer to the next transmit buffer in the descriptor chain.*
- `const uint8_t * mac_lld_get_next_receive_buffer (MACReceiveDescriptor *rdp, size_t *sizep)`  
*Returns a pointer to the next receive buffer in the descriptor chain.*

## Enumerations

- `enum macstate_t { MAC_UNINIT = 0, MAC_STOP = 1, MAC_ACTIVE = 2 }`  
*Driver state machine possible states.*

## Variables

- `MACDriver ETHD1`  
*MAC1 driver identifier.*

### 7.16.2 Macro Definition Documentation

#### 7.16.2.1 `#define MAC_USE_ZERO_COPY FALSE`

Enables an event sources for incoming packets.

#### 7.16.2.2 `#define MAC_USE_EVENTS TRUE`

Enables an event sources for incoming packets.

#### 7.16.2.3 `#define macGetReceiveEventSource( macp ) (&(macp)->rdevent)`

Returns the received frames event source.

##### Parameters

in	<code>macp</code>	pointer to the <code>MACDriver</code> object
----	-------------------	--

##### Returns

The pointer to the `EventSource` structure.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.16.2.4 `#define macWriteTransmitDescriptor( tdp, buf, size ) mac_lld_write_transmit_descriptor(tdp, buf, size)`

Writes to a transmit descriptor's stream.

##### Parameters

in	<code>tdp</code>	pointer to a <code>MACTransmitDescriptor</code> structure
in	<code>buf</code>	pointer to the buffer containing the data to be written

in	size	number of bytes to be written
----	------	-------------------------------

**Returns**

The number of bytes written into the descriptor's stream, this value can be less than the amount specified in the parameter `size` if the maximum frame size is reached.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.2.5 #define macReadReceiveDescriptor( *rdp*, *buf*, *size* ) mac\_lld\_read\_receive\_descriptor(*rdp*, *buf*, *size*)

Reads from a receive descriptor's stream.

**Parameters**

in	<i>rdp</i>	pointer to a <a href="#">MACReceiveDescriptor</a> structure
in	<i>buf</i>	pointer to the buffer that will receive the read data
in	<i>size</i>	number of bytes to be read

**Returns**

The number of bytes read from the descriptor's stream, this value can be less than the amount specified in the parameter `size` if there are no more bytes to read.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.2.6 #define macGetNextTransmitBuffer( *tdp*, *size*, *sizep* ) mac\_lld\_get\_next\_transmit\_buffer(*tdp*, *size*, *sizep*)

Returns a pointer to the next transmit buffer in the descriptor chain.

**Note**

The API guarantees that enough buffers can be requested to fill a whole frame.

**Parameters**

in	<i>tdp</i>	pointer to a <a href="#">MACTransmitDescriptor</a> structure
in	<i>size</i>	size of the requested buffer. Specify the frame size on the first call then scale the value down subtracting the amount of data already copied into the previous buffers.
out	<i>sizep</i>	pointer to variable receiving the real buffer size. The returned value can be less than the amount requested, this means that more buffers must be requested in order to fill the frame data entirely.

**Returns**

Pointer to the returned buffer.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.16.2.7 #define macGetNextReceiveBuffer( *rdp*, *sizep* ) mac\_lld\_get\_next\_receive\_buffer(*rdp*,*sizep*)

Returns a pointer to the next receive buffer in the descriptor chain.

#### Note

The API guarantees that the descriptor chain contains a whole frame.

#### Parameters

<i>in</i>	<i>rdp</i>	pointer to a <a href="#">MACReceiveDescriptor</a> structure
<i>out</i>	<i>sizep</i>	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.

#### Returns

Pointer to the returned buffer.

#### Return values

<i>NULL</i>	if the buffer chain has been entirely scanned.
-------------	--

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.16.2.8 #define MAC\_SUPPORTS\_ZERO\_COPY TRUE

This implementation supports the zero-copy mode API.

### 7.16.2.9 #define PLATFORM\_MAC\_USE\_MAC1 FALSE

MAC driver enable switch.

If set to TRUE the support for MAC1 is included.

#### Note

The default is FALSE.

## 7.16.3 Typedef Documentation

### 7.16.3.1 typedef struct MACDriver MACDriver

Type of a structure representing a MAC driver.

## 7.16.4 Enumeration Type Documentation

### 7.16.4.1 enum macstate\_t

Driver state machine possible states.

#### Enumerator

**MAC\_UNINIT** Not initialized.

**MAC\_STOP** Stopped.

**MAC\_ACTIVE** Active.

## 7.16.5 Function Documentation

### 7.16.5.1 void macInit( void )

MAC Driver initialization.

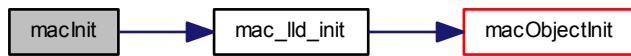
#### Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.16.5.2 void macObjectInit( MACDriver \* macp )

Initialize the standard part of a [MACDriver](#) structure.

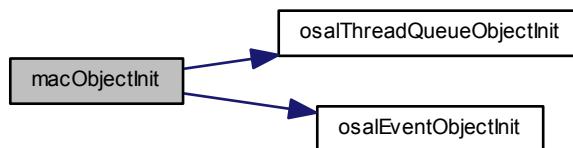
#### Parameters

out	<i>macp</i>	pointer to the <a href="#">MACDriver</a> object
-----	-------------	---

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.16.5.3 void macStart( MACDriver \* macp, const MACConfig \* config )

Configures and activates the MAC peripheral.

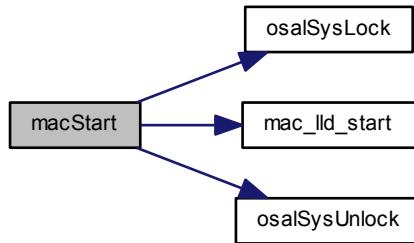
**Parameters**

in	<i>macp</i>	pointer to the <a href="#">MACDriver</a> object
in	<i>config</i>	pointer to the <a href="#">MACConfig</a> object

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.16.5.4 void macStop ( MACDriver \* *macp* )**

Deactivates the MAC peripheral.

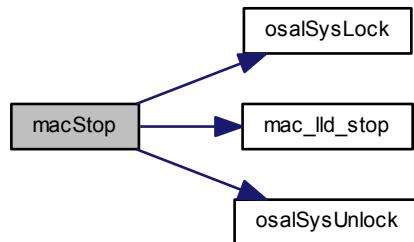
**Parameters**

in	<i>macp</i>	pointer to the <a href="#">MACDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.16.5.5 `msg_t macWaitTransmitDescriptor( MACDriver * macp, MACTransmitDescriptor * tdp, systime_t timeout )`

Allocates a transmission descriptor.

One of the available transmission descriptors is locked and returned. If a descriptor is not currently available then the invoking thread is queued until one is freed.

#### Parameters

in	<i>macp</i>	pointer to the <code>MACDriver</code> object
out	<i>tdp</i>	pointer to a <code>MACTransmitDescriptor</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <code>TIME_IMMEDIATE</code> immediate timeout.</li> <li>• <code>TIME_INFINITE</code> no timeout.</li> </ul>

#### Returns

The operation status.

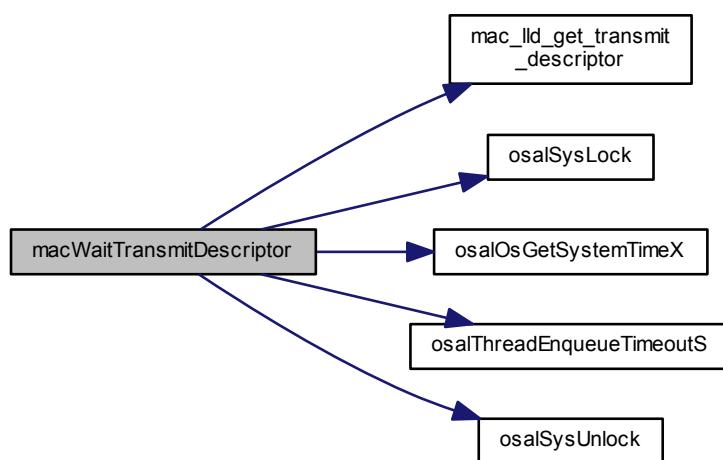
#### Return values

<code>MSG_OK</code>	the descriptor was obtained.
<code>MSG_TIMEOUT</code>	the operation timed out, descriptor not initialized.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



**7.16.5.6 void macReleaseTransmitDescriptor ( MACTransmitDescriptor \* *tdp* )**

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

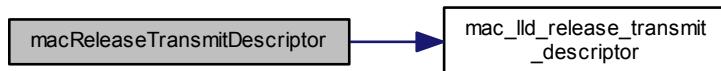
**Parameters**

in	<i>tdp</i>	the pointer to the <a href="#">MACTransmitDescriptor</a> structure
----	------------	--

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.16.5.7 `msg_t macWaitReceiveDescriptor ( MACDriver * macp, MACReceiveDescriptor * rdp, systime_t timeout )`

Waits for a received frame.

Stops until a frame is received and buffered. If a frame is not immediately available then the invoking thread is queued until one is received.

**Parameters**

in	<i>macp</i>	pointer to the <a href="#">MACDriver</a> object
out	<i>rdp</i>	pointer to a <a href="#">MACReceiveDescriptor</a> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <li>• <i>TIME_IMMEDIATE</i> immediate timeout.</li> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

**Returns**

The operation status.

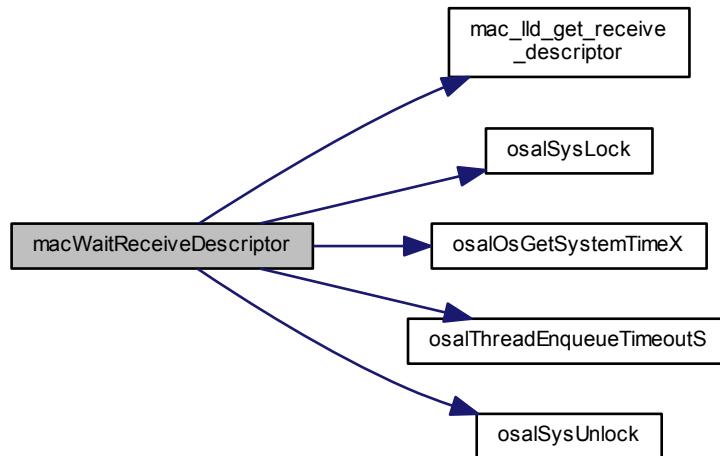
**Return values**

<i>MSG_OK</i>	the descriptor was obtained.
<i>MSG_TIMEOUT</i>	the operation timed out, descriptor not initialized.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.16.5.8 void macReleaseReceiveDescriptor ( **MACReceiveDescriptor** \* *rdp* )**

Releases a receive descriptor.

The descriptor and its buffer are made available for more incoming frames.

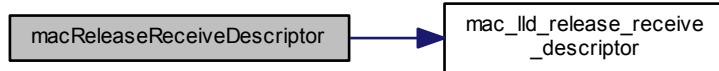
**Parameters**

in	<i>rdp</i>	the pointer to the <code>MACReceiveDescriptor</code> structure
----	------------	--

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.16.5.9 bool macPollLinkStatus ( MACDriver \* macp )

Updates and returns the link status.

##### Parameters

in	<i>macp</i>	pointer to the <a href="#">MACDriver</a> object
----	-------------	---

##### Returns

The link status.

##### Return values

<i>true</i>	if the link is active.
<i>false</i>	if the link is down.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



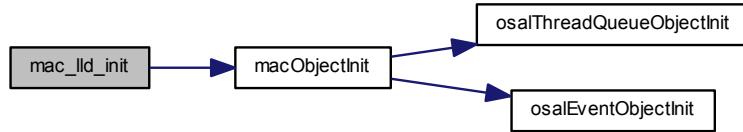
#### 7.16.5.10 void mac\_lld\_init ( void )

Low level MAC initialization.

##### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.16.5.11 void mac\_lld\_start ( **MACDriver** \* *macp* )

Configures and activates the MAC peripheral.

##### Parameters

in	<i>macp</i>	pointer to the <b>MACDriver</b> object
----	-------------	--

##### Function Class:

Not an API, this function is for internal use only.

#### 7.16.5.12 void mac\_lld\_stop ( **MACDriver** \* *macp* )

Deactivates the MAC peripheral.

##### Parameters

in	<i>macp</i>	pointer to the <b>MACDriver</b> object
----	-------------	--

##### Function Class:

Not an API, this function is for internal use only.

#### 7.16.5.13 **msg\_t** mac\_lld\_get\_transmit\_descriptor ( **MACDriver** \* *macp*, **MACTransmitDescriptor** \* *tdp* )

Returns a transmission descriptor.

One of the available transmission descriptors is locked and returned.

##### Parameters

in	<i>macp</i>	pointer to the <b>MACDriver</b> object
out	<i>tdp</i>	pointer to a <b>MACTransmitDescriptor</b> structure

##### Returns

The operation status.

**Return values**

<i>MSG_OK</i>	the descriptor has been obtained.
<i>MSG_TIMEOUT</i>	descriptor not available.

**Function Class:**

Not an API, this function is for internal use only.

**7.16.5.14 void mac\_lld\_release\_transmit\_descriptor ( **MACTransmitDescriptor** \* *tdp* )**

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

**Parameters**

<i>in</i>	<i>tdp</i>	the pointer to the <b>MACTransmitDescriptor</b> structure
-----------	------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.16.5.15 msg\_t mac\_lld\_get\_receive\_descriptor ( **MACDriver** \* *macp*, **MACReceiveDescriptor** \* *rdp* )**

Returns a receive descriptor.

**Parameters**

<i>in</i>	<i>macp</i>	pointer to the <b>MACDriver</b> object
<i>out</i>	<i>rdp</i>	pointer to a <b>MACReceiveDescriptor</b> structure

**Returns**

The operation status.

**Return values**

<i>MSG_OK</i>	the descriptor has been obtained.
<i>MSG_TIMEOUT</i>	descriptor not available.

**Function Class:**

Not an API, this function is for internal use only.

**7.16.5.16 void mac\_lld\_release\_receive\_descriptor ( **MACReceiveDescriptor** \* *rdp* )**

Releases a receive descriptor.

The descriptor and its buffer are made available for more incoming frames.

**Parameters**

<i>in</i>	<i>rdp</i>	the pointer to the <b>MACReceiveDescriptor</b> structure
-----------	------------	--

**Function Class:**

Not an API, this function is for internal use only.

**7.16.5.17 bool mac\_lld\_poll\_link\_status ( **MACDriver** \* *macp* )**

Updates and returns the link status.

**Parameters**

in	<i>macp</i>	pointer to the <a href="#">MACDriver</a> object
----	-------------	---

**Returns**

The link status.

**Return values**

<i>true</i>	if the link is active.
<i>false</i>	if the link is down.

**Function Class:**

Not an API, this function is for internal use only.

**7.16.5.18 size\_t mac\_lld\_write\_transmit\_descriptor ( [MACTransmitDescriptor](#) \* *tdp*, uint8\_t \* *buf*, size\_t *size* )**

Writes to a transmit descriptor's stream.

**Parameters**

in	<i>tdp</i>	pointer to a <a href="#">MACTransmitDescriptor</a> structure
in	<i>buf</i>	pointer to the buffer containing the data to be written
in	<i>size</i>	number of bytes to be written

**Returns**

The number of bytes written into the descriptor's stream, this value can be less than the amount specified in the parameter *size* if the maximum frame size is reached.

**Function Class:**

Not an API, this function is for internal use only.

**7.16.5.19 size\_t mac\_lld\_read\_receive\_descriptor ( [MACReceiveDescriptor](#) \* *rdp*, uint8\_t \* *buf*, size\_t *size* )**

Reads from a receive descriptor's stream.

**Parameters**

in	<i>rdp</i>	pointer to a <a href="#">MACReceiveDescriptor</a> structure
in	<i>buf</i>	pointer to the buffer that will receive the read data
in	<i>size</i>	number of bytes to be read

**Returns**

The number of bytes read from the descriptor's stream, this value can be less than the amount specified in the parameter *size* if there are no more bytes to read.

**Function Class:**

Not an API, this function is for internal use only.

### 7.16.5.20 `uint8_t * mac_lld_get_next_transmit_buffer ( MACTransmitDescriptor * tdp, size_t size, size_t * sizep )`

Returns a pointer to the next transmit buffer in the descriptor chain.

#### Note

The API guarantees that enough buffers can be requested to fill a whole frame.

#### Parameters

in	<i>tdp</i>	pointer to a <code>MACTransmitDescriptor</code> structure
in	<i>size</i>	size of the requested buffer. Specify the frame size on the first call then scale the value down subtracting the amount of data already copied into the previous buffers.
out	<i>sizep</i>	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned. Note that a returned size lower than the amount requested means that more buffers must be requested in order to fill the frame data entirely.

#### Returns

Pointer to the returned buffer.

#### Return values

<code>NULL</code>	if the buffer chain has been entirely scanned.
-------------------	--

#### Function Class:

Not an API, this function is for internal use only.

### 7.16.5.21 `const uint8_t * mac_lld_get_next_receive_buffer ( MACReceiveDescriptor * rdp, size_t * sizep )`

Returns a pointer to the next receive buffer in the descriptor chain.

#### Note

The API guarantees that the descriptor chain contains a whole frame.

#### Parameters

in	<i>rdp</i>	pointer to a <code>MACReceiveDescriptor</code> structure
out	<i>sizep</i>	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.

#### Returns

Pointer to the returned buffer.

#### Return values

<code>NULL</code>	if the buffer chain has been entirely scanned.
-------------------	--

#### Function Class:

Not an API, this function is for internal use only.

## 7.16.6 Variable Documentation

### 7.16.6.1 MACDriver ETHD1

MAC1 driver identifier.

## 7.17 HAL

Hardware Abstraction Layer.

### 7.17.1 Detailed Description

Hardware Abstraction Layer.

Under ChibiOS the set of the various device driver interfaces is called the HAL subsystem: Hardware Abstraction Layer. The HAL is the abstract interface between ChibiOS applications and hardware.

### 7.17.2 HAL Device Drivers Architecture

The HAL contains several kind of modules:

- Normal Device Drivers
- Complex Device Drivers
- Interfaces
- Inner Code

### 7.17.3 HAL Normal Device Drivers

Normal device are meant to interface the application to the underlying hardware through an high level API. Normal Device Drivers are split in two layers:

- High Level Device Driver (**HLD**). This layer contains the definitions of the driver's APIs and the platform independent part of the driver.

An HLD is composed by two files:

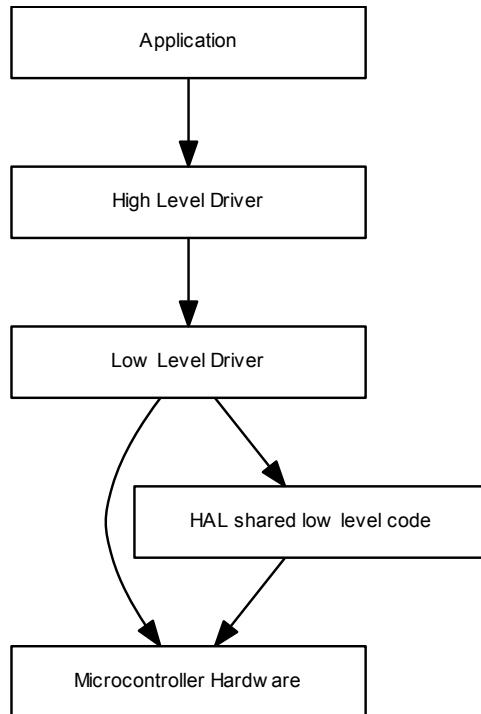
- <driver>.c, the HLD implementation file. This file must be included in the Makefile in order to use the driver.
- <driver>.h, the HLD header file. This file is implicitly included by the HAL header file `hal.h`.

- Low Level Device Driver (**LLD**). This layer contains the platform dependent part of the driver.

A LLD is composed by two files:

- <driver>\_lld.c, the LLD implementation file. This file must be included in the Makefile in order to use the driver.
- <driver>\_lld.h, the LLD header file. This file is implicitly included by the HLD header file.

### 7.17.3.1 Diagram



### 7.17.4 HAL Complex Device Drivers

It is a class of device drivers that offer an high level API but do not use the hardware directly. Complex device drivers use other drivers for accessing the machine resources.

### 7.17.5 HAL Interfaces

An interface is a binary structure allowing the access to a service using virtual functions. This allows to create drivers that can be accessed using a common interface. The concept of interface is commonly found in object-oriented languages like Java or C++, their meaning in ChibiOS/HAL is exactly the same.

### 7.17.6 HAL Inner Code

Some modules are shared among multiple device drivers and are not necessarily meant to be used by the application layer.

## Modules

- [Configuration](#)  
*HAL Configuration.*
- [Normal Drivers](#)

*HAL Normal Drivers.*

- [Complex Drivers](#)

*HAL Complex Drivers.*

- [Interfaces](#)

*HAL Interfaces.*

- [Inner Code](#)

*HAL Inner Code.*

- [Support Code](#)

*HAL Support Code.*

- [OSAL](#)

*Operating System Abstraction Layer.*

## 7.18 Configuration

HAL Configuration.

### 7.18.1 Detailed Description

HAL Configuration.

The file `halconf.h` contains the high level settings for all the drivers supported by the HAL. The low level, platform dependent, settings are contained in the `mcuconf.h` file instead and are described in the various platforms reference manuals.

#### Drivers enable switches

- `#define HAL_USE_PAL TRUE`  
*Enables the PAL subsystem.*
- `#define HAL_USE_ADC TRUE`  
*Enables the ADC subsystem.*
- `#define HAL_USE_CAN TRUE`  
*Enables the CAN subsystem.*
- `#define HAL_USE_DAC FALSE`  
*Enables the DAC subsystem.*
- `#define HAL_USE_EXT TRUE`  
*Enables the EXT subsystem.*
- `#define HAL_USE_GPT TRUE`  
*Enables the GPT subsystem.*
- `#define HAL_USE_I2C TRUE`  
*Enables the I2C subsystem.*
- `#define HAL_USE_I2S TRUE`  
*Enables the I2S subsystem.*
- `#define HAL_USE_ICU TRUE`  
*Enables the ICU subsystem.*
- `#define HAL_USE_MAC TRUE`  
*Enables the MAC subsystem.*
- `#define HAL_USE_MMC_SPI TRUE`  
*Enables the MMC\_SPI subsystem.*
- `#define HAL_USE_PWM TRUE`  
*Enables the PWM subsystem.*
- `#define HAL_USE_RTC TRUE`  
*Enables the RTC subsystem.*
- `#define HAL_USE_SDC TRUE`  
*Enables the SDC subsystem.*
- `#define HAL_USE_SERIAL TRUE`  
*Enables the SERIAL subsystem.*
- `#define HAL_USE_SERIAL_USB TRUE`  
*Enables the SERIAL over USB subsystem.*
- `#define HAL_USE_SPI TRUE`  
*Enables the SPI subsystem.*
- `#define HAL_USE_UART TRUE`  
*Enables the UART subsystem.*
- `#define HAL_USE_USB TRUE`

- `#define HAL_USE_WDG TRUE`  
*Enables the WDG subsystem.*

### ADC driver related setting

- `#define ADC_USE_WAIT TRUE`  
*Enables synchronous APIs.*
- `#define ADC_USE_MUTUAL_EXCLUSION TRUE`  
*Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.*

### CAN driver related setting

- `#define CAN_USE_SLEEP_MODE TRUE`  
*Sleep mode related APIs inclusion switch.*

### I2C driver related setting

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`  
*Enables the mutual exclusion APIs on the I2C bus.*

### MAC driver related setting

- `#define MAC_USE_ZERO_COPY TRUE`  
*Enables an event sources for incoming packets.*
- `#define MAC_USE_EVENTS TRUE`  
*Enables an event sources for incoming packets.*

### MMC\_SPI driver related setting

- `#define MMC_NICE_WAITING TRUE`  
*Delays insertions.*

### SDC driver related setting

- `#define SDC_INIT_RETRY 100`  
*Number of initialization attempts before rejecting the card.*
- `#define SDC_MMC_SUPPORT TRUE`  
*Include support for MMC cards.*
- `#define SDC_NICE_WAITING TRUE`  
*Delays insertions.*

### SERIAL driver related setting

- `#define SERIAL_DEFAULT_BITRATE 38400`  
*Default bit rate.*
- `#define SERIAL_BUFFERS_SIZE 16`  
*Serial buffers size.*

## SERIAL\_USB driver related setting

- `#define SERIAL_USB_BUFFERS_SIZE 256`  
*Serial over USB buffers size.*
- `#define SERIAL_USB_BUFFERS_NUMBER 2`  
*Serial over USB number of buffers.*

## SPI driver related setting

- `#define SPI_USE_WAIT TRUE`  
*Enables synchronous APIs.*
- `#define SPI_USE_MUTUAL_EXCLUSION TRUE`  
*Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.*

## UART driver related setting

- `#define UART_USE_WAIT TRUE`  
*Enables synchronous APIs.*
- `#define UART_USE_MUTUAL_EXCLUSION TRUE`  
*Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.*

## USB driver related setting

- `#define USB_USE_WAIT TRUE`  
*Enables synchronous APIs.*

### 7.18.2 Macro Definition Documentation

#### 7.18.2.1 `#define HAL_USE_PAL TRUE`

Enables the PAL subsystem.

#### 7.18.2.2 `#define HAL_USE_ADC TRUE`

Enables the ADC subsystem.

#### 7.18.2.3 `#define HAL_USE_CAN TRUE`

Enables the CAN subsystem.

#### 7.18.2.4 `#define HAL_USE_DAC FALSE`

Enables the DAC subsystem.

#### 7.18.2.5 `#define HAL_USE_EXT TRUE`

Enables the EXT subsystem.

7.18.2.6 #define HAL\_USE\_GPT TRUE

Enables the GPT subsystem.

7.18.2.7 #define HAL\_USE\_I2C TRUE

Enables the I2C subsystem.

7.18.2.8 #define HAL\_USE\_I2S TRUE

Enables the I2S subsystem.

7.18.2.9 #define HAL\_USE\_ICU TRUE

Enables the ICU subsystem.

7.18.2.10 #define HAL\_USE\_MAC TRUE

Enables the MAC subsystem.

7.18.2.11 #define HAL\_USE\_MMC\_SPI TRUE

Enables the MMC\_SPI subsystem.

7.18.2.12 #define HAL\_USE\_PWM TRUE

Enables the PWM subsystem.

7.18.2.13 #define HAL\_USE\_RTC TRUE

Enables the RTC subsystem.

7.18.2.14 #define HAL\_USE\_SDC TRUE

Enables the SDC subsystem.

7.18.2.15 #define HAL\_USE\_SERIAL TRUE

Enables the SERIAL subsystem.

7.18.2.16 #define HAL\_USE\_SERIAL\_USB TRUE

Enables the SERIAL over USB subsystem.

7.18.2.17 #define HAL\_USE\_SPI TRUE

Enables the SPI subsystem.

7.18.2.18 #define HAL\_USE\_UART TRUE

Enables the UART subsystem.

7.18.2.19 #define HAL\_USE\_USB TRUE

Enables the USB subsystem.

7.18.2.20 #define HAL\_USE\_WDG TRUE

Enables the WDG subsystem.

7.18.2.21 #define ADC\_USE\_WAIT TRUE

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

7.18.2.22 #define ADC\_USE\_MUTUAL\_EXCLUSION TRUE

Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

**Note**

Disabling this option saves both code and data space.

7.18.2.23 #define CAN\_USE\_SLEEP\_MODE TRUE

Sleep mode related APIs inclusion switch.

7.18.2.24 #define I2C\_USE\_MUTUAL\_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

7.18.2.25 #define MAC\_USE\_ZERO\_COPY TRUE

Enables an event sources for incoming packets.

7.18.2.26 #define MAC\_USE\_EVENTS TRUE

Enables an event sources for incoming packets.

7.18.2.27 #define MMC\_NICE\_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however. This option is recommended also if the SPI driver does not use a DMA channel and heavily loads the CPU.

**7.18.2.28 #define SDC\_INIT\_RETRY 100**

Number of initialization attempts before rejecting the card.

**Note**

Attempts are performed at 10mS intervals.

**7.18.2.29 #define SDC\_MMC\_SUPPORT TRUE**

Include support for MMC cards.

**Note**

MMC support is not yet implemented so this option must be kept at FALSE.

**7.18.2.30 #define SDC\_NICE\_WAITING TRUE**

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however.

**7.18.2.31 #define SERIAL\_DEFAULT\_BITRATE 38400**

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

**7.18.2.32 #define SERIAL\_BUFFERS\_SIZE 16**

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

**Note**

The default is 16 bytes for both the transmission and receive buffers.

**7.18.2.33 #define SERIAL\_USB\_BUFFERS\_SIZE 256**

Serial over USB buffers size.

Configuration parameter, the buffer size must be a multiple of the USB data endpoint maximum packet size.

**Note**

The default is 256 bytes for both the transmission and receive buffers.

**7.18.2.34 #define SERIAL\_USB\_BUFFERS\_NUMBER 2**

Serial over USB number of buffers.

**Note**

The default is 2 buffers.

**7.18.2.35 #define SPI\_USE\_WAIT TRUE**

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

**7.18.2.36 #define SPI\_USE\_MUTUAL\_EXCLUSION TRUE**

Enables the [spiAcquireBus\(\)](#) and [spiReleaseBus\(\)](#) APIs.

**Note**

Disabling this option saves both code and data space.

**7.18.2.37 #define UART\_USE\_WAIT TRUE**

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

**7.18.2.38 #define UART\_USE\_MUTUAL\_EXCLUSION TRUE**

Enables the [uartAcquireBus\(\)](#) and [uartReleaseBus\(\)](#) APIs.

**Note**

Disabling this option saves both code and data space.

**7.18.2.39 #define USB\_USE\_WAIT TRUE**

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

## 7.19 Normal Drivers

HAL Normal Drivers.

### 7.19.1 Detailed Description

HAL Normal Drivers.

#### Modules

- [ADC Driver](#)  
*Generic ADC Driver.*
- [CAN Driver](#)  
*Generic CAN Driver.*
- [DAC Driver](#)  
*Generic DAC Driver.*
- [EXT Driver](#)  
*Generic EXT Driver.*
- [GPT Driver](#)  
*Generic GPT Driver.*
- [HAL Driver](#)  
*Hardware Abstraction Layer.*
- [I2C Driver](#)  
*Generic I2C Driver.*
- [I2S Driver](#)  
*Generic I2S Driver.*
- [ICU Driver](#)  
*Generic ICU Driver.*
- [MAC Driver](#)  
*Generic MAC Driver.*
- [PAL Driver](#)  
*I/O Ports Abstraction Layer.*
- [PWM Driver](#)  
*Generic PWM Driver.*
- [RTC Driver](#)  
*Generic RTC Driver.*
- [SDC Driver](#)  
*Generic SD Card Driver.*
- [Serial Driver](#)  
*Generic Serial Driver.*
- [SPI Driver](#)  
*Generic SPI Driver.*
- [ST Driver](#)  
*Generic System Tick Driver.*
- [UART Driver](#)  
*Generic UART Driver.*
- [USB Driver](#)  
*Generic USB Driver.*
- [WDG Driver](#)  
*Generic WDG Driver.*

## 7.20 Complex Drivers

HAL Complex Drivers.

### 7.20.1 Detailed Description

HAL Complex Drivers.

#### Modules

- [MMC over SPI Driver](#)  
*Generic MMC driver.*
- [Serial over USB Driver](#)  
*Serial over USB Driver.*

## 7.21 Interfaces

HAL Interfaces.

### 7.21.1 Detailed Description

HAL Interfaces.

#### Modules

- [Abstract I/O Channel](#)
- [Abstract Files](#)
- [Abstract I/O Block Device](#)
- [Abstract Streams](#)

## 7.22 Inner Code

HAL Inner Code.

### 7.22.1 Detailed Description

HAL Inner Code.

#### Modules

- [I/O Buffers Queues](#)
- [I/O Bytes Queues](#)
- [MMC/SD Block Device](#)

## 7.23 Support Code

HAL Support Code.

### 7.23.1 Detailed Description

HAL Support Code.

#### Modules

- [MII/RMII Header](#)  
*MII/RMII Support Header.*
- [USB CDC Header](#)  
*USB CDC Support Header.*

## 7.24 OSAL

Operating System Abstraction Layer.

### 7.24.1 Detailed Description

Operating System Abstraction Layer.

#### The OSAL

The OSAL is the link between ChibiOS/HAL and services provided by operating systems like:

- Critical Zones handling.
- Interrupts handling.
- Runtime Errors management.
- Inter-task synchronization.
- Task-ISR synchronization.
- Time management.
- Events.

ChibiOS/HAL is designed to tightly integrate with the underlying RTOS in order to provide the best experience to developers and minimize integration issues.

This section describes the API that OSALs are expected to expose to the HAL.

#### RTOS Requirements

The OSAL API closely resembles the ChibiOS/RT API, for obvious reasons, however an OSAL module can be implemented for any reasonably complete RTOS or even a RTOS-less bare metal machine, if required.

In order to be able to support an HAL an RTOS should support the following minimal set of features:

- Task-level critical zones API.
- ISR-level critical zones API, only required on those CPU architectures supporting preemptable ISRs like Cortex-Mx cores.
- Ability to invoke API functions from inside a task critical zone. Functions that are required to support this feature are marked with an "I" or "S" letter at the end of the name.
- Ability to invoke API functions from inside an ISR critical zone. Functions that are required to support this feature are marked with an "I" letter at the end of the name.
- Tasks Queues or Counting Semaphores with Timeout capability.
- Ability to suspend a task and wakeup it from ISR with Timeout capability.
- Event flags, the mechanism can be simulated using callbacks in case the RTOS does not support it.
- Mutual Exclusion mechanism like Semaphores or Mutexes.

All the above requirements can be satisfied even on naked HW with a very think SW layer. In case that the HAL is required to work without an RTOS.

### Supported RTOSes

The RTOSes supported out of the box are:

- ChibiOS/RT
- ChibiOS/NIL

Implementations have also been successfully created on RTOSes not belonging to the ChibiOS products family but are not supported as a core feature of ChibiOS/HAL.

### Macros

- `#define OSAL_DBG_ENABLE_ASSERTS FALSE`  
*Enables OSAL assertions.*
- `#define OSAL_DBG_ENABLE_CHECKS FALSE`  
*Enables OSAL functions parameters checks.*

### Common constants

- `#define FALSE 0`
- `#define TRUE 1`
- `#define OSAL_SUCCESS false`
- `#define OSAL_FAILED true`

### Messages

- `#define MSG_OK (msg_t)0`
- `#define MSG_RESET (msg_t)-1`
- `#define MSG_TIMEOUT (msg_t)-2`

### Special time constants

- `#define TIME_IMMEDIATE ((systime_t)0)`
- `#define TIME_INFINITE ((systime_t)-1)`

### Systick modes.

- `#define OSAL_ST_MODE_NONE 0`
- `#define OSAL_ST_MODE_PERIODIC 1`
- `#define OSAL_ST_MODE_FREERUNNING 2`

### Systick parameters.

- `#define OSAL_ST_RESOLUTION 32`  
*Size in bits of the systick\_t type.*
- `#define OSAL_ST_FREQUENCY 1000`  
*Required systick frequency or resolution.*
- `#define OSAL_ST_MODE OSAL_ST_MODE_PERIODIC`  
*Systick mode required by the underlying OS.*

## IRQ-related constants

- `#define OSAL_IRQ_PRIORITY_LEVELS 16U`  
*Total priority levels.*
- `#define OSAL_IRQ_MAXIMUM_PRIORITY 0U`  
*Highest IRQ priority for HAL drivers.*

## Debug related macros

- `#define osalDbgAssert(c, remark)`  
*Condition assertion.*
- `#define osalDbgCheck(c)`  
*Function parameters check.*
- `#define osalDbgCheckClassI()`  
*I-Class state check.*
- `#define osalDbgCheckClassS()`  
*S-Class state check.*

## IRQ service routines wrappers

- `#define OSAL_IRQ_IS_VALID_PRIORITY(n) (((n) >= OSAL_IRQ_MAXIMUM_PRIORITY) && ((n) < OSAL_IRQ_PRIORITY_LEVELS))`  
*Priority level verification macro.*
- `#define OSAL_IRQ_PROLOGUE()`  
*IRQ prologue code.*
- `#define OSAL_IRQ_EPILOGUE()`  
*IRQ epilogue code.*
- `#define OSAL_IRQ_HANDLER(id) void id(void)`  
*IRQ handler function declaration.*

## Time conversion utilities

- `#define OSAL_S2ST(sec) ((systime_t)((uint32_t)(sec) * (uint32_t)OSAL_ST_FREQUENCY))`  
*Seconds to system ticks.*
- `#define OSAL_MS2ST(msec)`  
*Milliseconds to system ticks.*
- `#define OSAL_US2ST(usec)`  
*Microseconds to system ticks.*

## Time conversion utilities for the realtime counter

- `#define OSAL_S2RTC(freq, sec) ((freq) * (sec))`  
*Seconds to realtime counter.*
- `#define OSAL_MS2RTC(freq, msec) (rtcnt_t)((((freq) + 999UL) / 1000UL) * (msec))`  
*Milliseconds to realtime counter.*
- `#define OSAL_US2RTC(freq, usec) (rtcnt_t)((((freq) + 999999UL) / 1000000UL) * (usec))`  
*Microseconds to realtime counter.*

## Sleep macros using absolute time

- `#define osalThreadSleepSeconds(sec) osalThreadSleep(OSAL_S2ST(sec))`  
*Delays the invoking thread for the specified number of seconds.*
- `#define osalThreadSleepMilliseconds(msec) osalThreadSleep(OSAL_MS2ST(msec))`  
*Delays the invoking thread for the specified number of milliseconds.*
- `#define osalThreadSleepMicroseconds(usec) osalThreadSleep(OSAL_US2ST(usec))`  
*Delays the invoking thread for the specified number of microseconds.*

## Typedefs

- `typedef uint32_t syssts_t`  
*Type of a system status word.*
- `typedef int32_t msg_t`  
*Type of a message.*
- `typedef uint32_t systime_t`  
*Type of system time counter.*
- `typedef uint32_t rtcnt_t`  
*Type of realtime counter.*
- `typedef void *thread_reference_t`  
*Type of a thread reference.*
- `typedef struct event_source event_source_t`  
*Type of an event flags object.*
- `typedef void(* eventcallback_t) (event_source_t *esp)`  
*Type of an event source callback.*
- `typedef uint32_t eventflags_t`  
*Type of an event flags mask.*
- `typedef uint32_t mutex_t`  
*Type of a mutex.*

## Data Structures

- `struct event_source`  
*Events source object.*
- `struct threads_queue_t`  
*Type of a thread queue.*

## Functions

- `void osallinit (void)`  
*OSAL module initialization.*
- `void osalSysHalt (const char *reason)`  
*System halt with error message.*
- `void osalSysPolledDelayX (rtcnt_t cycles)`  
*Polled delay.*
- `void osalOsTimerHandlerI (void)`  
*System timer handler.*
- `void osalOsRescheduleS (void)`  
*Checks if a reschedule is required and performs it.*
- `systime_t osalOsGetSystemTimeX (void)`

- **Current system time.**
- void **osalThreadSleepS (systime\_t time)**  
*Suspends the invoking thread for the specified time.*
- void **osalThreadSleep (systime\_t time)**  
*Suspends the invoking thread for the specified time.*
- msg\_t **osalThreadSuspendS (thread\_reference\_t \*trp)**  
*Sends the current thread sleeping and sets a reference variable.*
- msg\_t **osalThreadSuspendTimeoutS (thread\_reference\_t \*trp, systime\_t timeout)**  
*Sends the current thread sleeping and sets a reference variable.*
- void **osalThreadResumeI (thread\_reference\_t \*trp, msg\_t msg)**  
*Wakes up a thread waiting on a thread reference object.*
- void **osalThreadResumeS (thread\_reference\_t \*trp, msg\_t msg)**  
*Wakes up a thread waiting on a thread reference object.*
- msg\_t **osalThreadEnqueueTimeoutS (threads\_queue\_t \*tqp, systime\_t timeout)**  
*Enqueues the caller thread.*
- void **osalThreadDequeueNextI (threads\_queue\_t \*tqp, msg\_t msg)**  
*Dequeues and wakes up one thread from the queue, if any.*
- void **osalThreadDequeueAllI (threads\_queue\_t \*tqp, msg\_t msg)**  
*Dequeues and wakes up all threads from the queue.*
- void **osalEventBroadcastFlagsI (event\_source\_t \*esp, eventflags\_t flags)**  
*Add flags to an event source object.*
- void **osalEventBroadcastFlags (event\_source\_t \*esp, eventflags\_t flags)**  
*Add flags to an event source object.*
- void **osalEventSetCallback (event\_source\_t \*esp, eventcallback\_t cb, void \*param)**  
*Event callback setup.*
- void **osalMutexLock (mutex\_t \*mp)**  
*Locks the specified mutex.*
- void **osalMutexUnlock (mutex\_t \*mp)**  
*Unlocks the specified mutex.*
- static void **osalSysDisable (void)**  
*Disables interrupts globally.*
- static void **osalSysEnable (void)**  
*Enables interrupts globally.*
- static void **osalSysLock (void)**  
*Enters a critical zone from thread context.*
- static void **osalSysUnlock (void)**  
*Leaves a critical zone from thread context.*
- static void **osalSysLockFromISR (void)**  
*Enters a critical zone from ISR context.*
- static void **osalSysUnlockFromISR (void)**  
*Leaves a critical zone from ISR context.*
- static syssts\_t **osalSysGetStatusAndLockX (void)**  
*Returns the execution status and enters a critical zone.*
- static void **osalSysRestoreStatusX (syssts\_t sts)**  
*Restores the specified execution status and leaves a critical zone.*
- static bool **osalOslsTimeWithinX (systime\_t time, systime\_t start, systime\_t end)**  
*Checks if the specified time is within the specified time window.*
- static void **osalThreadQueueObjectInit (threads\_queue\_t \*tqp)**  
*Initializes a threads queue object.*
- static void **osalEventObjectInit (event\_source\_t \*esp)**  
*Initializes an event flags object.*
- static void **osalMutexObjectInit (mutex\_t \*mp)**  
*Initializes a mutex\_t object.*

## Variables

- const char \* **osal\_halt\_msg**  
*Pointer to a halt error message.*
- const char \* **osal\_halt\_msg**  
*Pointer to a halt error message.*

### 7.24.2 Macro Definition Documentation

#### 7.24.2.1 #define OSAL\_ST\_RESOLUTION 32

Size in bits of the `systick_t` type.

#### 7.24.2.2 #define OSAL\_ST\_FREQUENCY 1000

Required systick frequency or resolution.

#### 7.24.2.3 #define OSAL\_ST\_MODE OSAL\_ST\_MODE\_PERIODIC

Systick mode required by the underlying OS.

#### 7.24.2.4 #define OSAL\_IRQ\_PRIORITY\_LEVELS 16U

Total priority levels.

Implementation not mandatory.

#### 7.24.2.5 #define OSAL\_IRQ\_MAXIMUM\_PRIORITY 0U

Highest IRQ priority for HAL drivers.

Implementation not mandatory.

#### 7.24.2.6 #define OSAL\_DBG\_ENABLE\_ASSERTS FALSE

Enables OSAL assertions.

#### 7.24.2.7 #define OSAL\_DBG\_ENABLE\_CHECKS FALSE

Enables OSAL functions parameters checks.

#### 7.24.2.8 #define osalDbgAssert( c, remark )

##### Value:

```
do {
    /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
    if (OSAL_DBG_ENABLE_ASSERTS != FALSE) {
        if (!(c)) {
            /*lint -restore*/
            osalSysHalt(__func__);
        }
    }
} while (false)
```

Condition assertion.

If the condition check fails then the OSAL panics with a message and halts.

#### Note

The condition is tested only if the `OSAL_ENABLE_ASSERTIONS` switch is enabled.

The remark string is not currently used except for putting a comment in the code about the assertion.

#### Parameters

in	<i>c</i>	the condition to be verified to be true
in	<i>remark</i>	a remark string

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.24.2.9 #define osalDbgCheck( *c* )

#### Value:

```
do {
    /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
    if (OSAL_DBG_ENABLE_CHECKS != FALSE) {
        if (!(c)) {
            /*lint -restore*/
            osalSysHalt (__func__);
        }
    }
} while (false)
```

Function parameters check.

If the condition check fails then the OSAL panics and halts.

#### Note

The condition is tested only if the `OSAL_ENABLE_CHECKS` switch is enabled.

#### Parameters

in	<i>c</i>	the condition to be verified to be true
----	----------	---

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.24.2.10 #define osalDbgCheckClassI( )

I-Class state check.

#### Note

Implementation is optional.

#### 7.24.2.11 #define osalDbgCheckClassS( )

S-Class state check.

#### Note

Implementation is optional.

7.24.2.12 #define OSAL\_IRQ\_IS\_VALID\_PRIORITY( *n* ) (((*n*) >= OSAL\_IRQ\_MAXIMUM\_PRIORITY) && ((*n*) < OSAL\_IRQ\_PRIORITY\_LEVELS))

Priority level verification macro.

7.24.2.13 #define OSAL\_IRQ\_PROLOGUE( )

IRQ prologue code.

This macro must be inserted at the start of all IRQ handlers.

7.24.2.14 #define OSAL\_IRQ\_EPILOGUE( )

IRQ epilogue code.

This macro must be inserted at the end of all IRQ handlers.

7.24.2.15 #define OSAL\_IRQ\_HANDLER( *id* ) void id(void)

IRQ handler function declaration.

This macro hides the details of an ISR function declaration.

#### Parameters

in	<i>id</i>	a vector name as defined in vectors.s
----	-----------	---------------------------------------

7.24.2.16 #define OSAL\_S2ST( *sec* ) ((*systime\_t*)((*uint32\_t*)(*sec*) \* (*uint32\_t*)OSAL\_ST\_FREQUENCY))

Seconds to system ticks.

Converts from seconds to system ticks number.

#### Note

The result is rounded upward to the next tick boundary.

#### Parameters

in	<i>sec</i>	number of seconds
----	------------	-------------------

#### Returns

The number of ticks.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.24.2.17 #define OSAL\_MS2ST( *msec* )

#### Value:

```
((systime_t)((((uint32_t)(msec)) * \
((uint32_t)OSAL_ST_FREQUENCY)) - 1UL) / 1000UL) + 1UL)) \
```

Milliseconds to system ticks.

Converts from milliseconds to system ticks number.

**Note**

The result is rounded upward to the next tick boundary.

**Parameters**

in	msec	number of milliseconds
----	------	------------------------

**Returns**

The number of ticks.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.24.2.18 #define OSAL\_US2ST( *usec* )****Value:**

```
((systime_t) (((((uint32_t)(usec)) *
    ((uint32_t)OSAL_ST_FREQUENCY)) - 1UL) / 1000000UL) + 1UL)
```

Microseconds to system ticks.

Converts from microseconds to system ticks number.

**Note**

The result is rounded upward to the next tick boundary.

**Parameters**

in	usec	number of microseconds
----	------	------------------------

**Returns**

The number of ticks.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.24.2.19 #define OSAL\_S2RTC( *freq*, *sec* ) ((freq) \* (sec))**

Seconds to realtime counter.

Converts from seconds to realtime counter cycles.

**Note**

The macro assumes that freq >= 1.

**Parameters**

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>sec</i>	number of seconds

**Returns**

The number of cycles.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.24.2.20 #define OSAL\_MS2RTC( freq, msec ) (rtcnt\_t)((((freq) + 999UL) / 1000UL) \* (msec))**

Milliseconds to realtime counter.

Converts from milliseconds to realtime counter cycles.

**Note**

The result is rounded upward to the next millisecond boundary.

The macro assumes that *freq*  $\geq 1000$ .

**Parameters**

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>msec</i>	number of milliseconds

**Returns**

The number of cycles.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.24.2.21 #define OSAL\_US2RTC( freq, usec ) (rtcnt\_t)((((freq) + 999999UL) / 1000000UL) \* (usec))**

Microseconds to realtime counter.

Converts from microseconds to realtime counter cycles.

**Note**

The result is rounded upward to the next microsecond boundary.

The macro assumes that *freq*  $\geq 1000000$ .

**Parameters**

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>usec</i>	number of microseconds

**Returns**

The number of cycles.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.24.2.22 #define osalThreadSleepSeconds( sec ) osalThreadSleep(OSAL\_S2ST(sec))

Delays the invoking thread for the specified number of seconds.

##### Note

The specified time is rounded up to a value allowed by the real system tick clock.  
The maximum specifiable value is implementation dependent.

##### Parameters

in	sec	time in seconds, must be different from zero
----	-----	--

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.24.2.23 #define osalThreadSleepMilliseconds( msec ) osalThreadSleep(OSAL\_MS2ST(msec))

Delays the invoking thread for the specified number of milliseconds.

##### Note

The specified time is rounded up to a value allowed by the real system tick clock.  
The maximum specifiable value is implementation dependent.

##### Parameters

in	msec	time in milliseconds, must be different from zero
----	------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.24.2.24 #define osalThreadSleepMicroseconds( usec ) osalThreadSleep(OSAL\_US2ST(usec))

Delays the invoking thread for the specified number of microseconds.

##### Note

The specified time is rounded up to a value allowed by the real system tick clock.  
The maximum specifiable value is implementation dependent.

##### Parameters

in	usec	time in microseconds, must be different from zero
----	------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.24.3 Typedef Documentation

#### 7.24.3.1 typedef uint32\_t syssts\_t

Type of a system status word.

**7.24.3.2 `typedef int32_t msg_t`**

Type of a message.

**7.24.3.3 `typedef uint32_t systime_t`**

Type of system time counter.

**7.24.3.4 `typedef uint32_t rtcnt_t`**

Type of realtime counter.

**7.24.3.5 `typedef void* thread_reference_t`**

Type of a thread reference.

**7.24.3.6 `typedef struct event_source event_source_t`**

Type of an event flags object.

**Note**

The content of this structure is not part of the API and should not be relied upon. Implementers may define this structure in an entirely different way.

Retrieval and clearing of the flags are not defined in this API and are implementation-dependent.

**7.24.3.7 `typedef void(* eventcallback_t)(event_source_t *esp)`**

Type of an event source callback.

**Note**

This type is not part of the OSAL API and is provided exclusively as an example and for convenience.

**7.24.3.8 `typedef uint32_t eventflags_t`**

Type of an event flags mask.

**7.24.3.9 `typedef uint32_t mutex_t`**

Type of a mutex.

**Note**

If the OS does not support mutexes or there is no OS then them mechanism can be simulated.

## 7.24.4 Function Documentation

**7.24.4.1 `void osallinit( void )`**

OSAL module initialization.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.24.4.2 void osalSysHalt ( const char \* *reason* )**

System halt with error message.

**Parameters**

in	<i>reason</i>	the halt message pointer
----	---------------	--------------------------

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.24.4.3 void osalSysPolledDelayX ( rtcnt\_t *cycles* )**

Polled delay.

**Note**

The real delay is always few cycles in excess of the specified value.

**Parameters**

in	<i>cycles</i>	number of cycles
----	---------------	------------------

**Function Class:**

This is an **X-Class** API, this function can be invoked from any context.

**7.24.4.4 void osalOsTimerHandlerI ( void )**

System timer handler.

The handler is used for scheduling and Virtual Timers management.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

#### 7.24.4.5 void osalOsRescheduleS ( void )

Checks if a reschedule is required and performs it.

##### Note

I-Class functions invoked from thread context must not reschedule by themselves, an explicit reschedule using this function is required in this scenario.

Not implemented in this simplified OSAL.

##### Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

#### 7.24.4.6 systime\_t osalOsGetSystemTimeX ( void )

Current system time.

Returns the number of system ticks since the `osalInit()` invocation.

##### Note

The counter can reach its maximum and then restart from zero.

This function can be called from any context but its atomicity is not guaranteed on architectures whose word size is less than `systime_t` size.

##### Returns

The system time in ticks.

##### Function Class:

This is an **X-Class** API, this function can be invoked from any context.

#### 7.24.4.7 void osalThreadSleepS ( systime\_t time )

Suspends the invoking thread for the specified time.

##### Parameters

in	time	the delay in system ticks, the special values are handled as follow: <ul style="list-style-type: none"><li>• <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification.</li><li>• <code>TIME_IMMEDIATE</code> this value is not allowed.</li></ul>
----	------	---

##### Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

#### 7.24.4.8 void osalThreadSleep ( systime\_t time )

Suspends the invoking thread for the specified time.

**Parameters**

in	<i>time</i>	the delay in system ticks, the special values are handled as follow: <ul style="list-style-type: none"><li>• <i>TIME_INFINITE</i> is allowed but interpreted as a normal time specification.</li><li>• <i>TIME_IMMEDIATE</i> this value is not allowed.</li></ul>
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.24.4.9 msg\_t osalThreadSuspendS ( thread\_reference\_t \* *trp* )**

Sends the current thread sleeping and sets a reference variable.

**Note**

This function must reschedule, it can only be called from thread context.

**Parameters**

in	<i>trp</i>	a pointer to a thread reference object
----	------------	--

**Returns**

The wake up message.

**Function Class:**

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

**7.24.4.10 msg\_t osalThreadSuspendTimeoutS ( thread\_reference\_t \* *trp*, systime\_t *timeout* )**

Sends the current thread sleeping and sets a reference variable.

**Note**

This function must reschedule, it can only be called from thread context.

**Parameters**

in	<i>trp</i>	a pointer to a thread reference object
in	<i>timeout</i>	the timeout in system ticks, the special values are handled as follow: <ul style="list-style-type: none"><li>• <i>TIME_INFINITE</i> the thread enters an infinite sleep state.</li><li>• <i>TIME_IMMEDIATE</i> the thread is not enqueued and the function returns <i>MSG_TIMEOUT</i> as if a timeout occurred.</li></ul>

**Returns**

The wake up message.

## Return values

<code>MSG_TIMEOUT</code>	if the operation timed out.
--------------------------	-----------------------------

## Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

7.24.4.11 `void osalThreadResumel ( thread_reference_t * trp, msg_t msg )`

Wakes up a thread waiting on a thread reference object.

## Note

This function must not reschedule because it can be called from ISR context.

## Parameters

in	<i>trp</i>	a pointer to a thread reference object
in	<i>msg</i>	the message code

## Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.24.4.12 `void osalThreadResumeS ( thread_reference_t * trp, msg_t msg )`

Wakes up a thread waiting on a thread reference object.

## Note

This function must reschedule, it can only be called from thread context.

## Parameters

in	<i>trp</i>	a pointer to a thread reference object
in	<i>msg</i>	the message code

## Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.24.4.13 `msg_t osalThreadEnqueueTimeoutS ( threads_queue_t * tqp, systime_t timeout )`

Enqueues the caller thread.

The caller thread is enqueued and put to sleep until it is dequeued or the specified timeouts expires.

## Parameters

in	<i>tqp</i>	pointer to the threads queue object
in	<i>timeout</i>	<p>the timeout in system ticks, the special values are handled as follow:</p> <ul style="list-style-type: none"> <li>• <i>TIME_INFINITE</i> the thread enters an infinite sleep state.</li> <li>• <i>TIME_IMMEDIATE</i> the thread is not enqueued and the function returns <code>MSG_TIMEOUT</code> as if a timeout occurred.</li> </ul>

**Returns**

The message from `osalQueueWakeupOneI()` or `osalQueueWakeupAllI()` functions.

**Return values**

<code>MSG_TIMEOUT</code>	if the thread has not been dequeued within the specified timeout or if the function has been invoked with <code>TIME_IMMEDIATE</code> as timeout specification.
--------------------------	---

**Function Class:**

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

**7.24.4.14 void osalThreadDequeueNextI( threads\_queue\_t \*tqp, msg\_t msg )**

Dequeues and wakes up one thread from the queue, if any.

**Parameters**

in	<i>tqp</i>	pointer to the threads queue object
in	<i>msg</i>	the message code

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.24.4.15 void osalThreadDequeueAllI( threads\_queue\_t \*tqp, msg\_t msg )**

Dequeues and wakes up all threads from the queue.

**Parameters**

in	<i>tqp</i>	pointer to the threads queue object
in	<i>msg</i>	the message code

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.24.4.16 void osalEventBroadcastFlagsI( event\_source\_t \*esp, eventflags\_t flags )**

Add flags to an event source object.

**Parameters**

in	<i>esp</i>	pointer to the event flags object
in	<i>flags</i>	flags to be ORed to the flags mask

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.24.4.17 void osalEventBroadcastFlags( event\_source\_t \*esp, eventflags\_t flags )**

Add flags to an event source object.

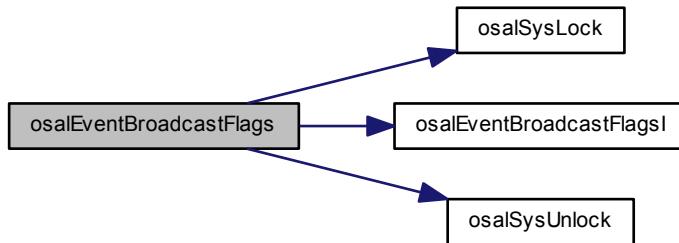
#### Parameters

in	<i>esp</i>	pointer to the event flags object
in	<i>flags</i>	flags to be ORed to the flags mask

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.24.4.18 void osalEventSetCallback ( *event\_source\_t* \* *esp*, *eventcallback\_t* *cb*, *void* \* *param* )

Event callback setup.

#### Note

The callback is invoked from ISR context and can only invoke I-Class functions. The callback is meant to wakeup the task that will handle the event by calling `osalEventGetAndClearFlagsI()`.

This function is not part of the OSAL API and is provided exclusively as an example and for convenience.

#### Parameters

in	<i>esp</i>	pointer to the event flags object
in	<i>cb</i>	pointer to the callback function
in	<i>param</i>	parameter to be passed to the callback function

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.24.4.19 void osalMutexLock ( *mutex\_t* \* *mp* )

Locks the specified mutex.

#### Postcondition

The mutex is locked and inserted in the per-thread stack of owned mutexes.

**Parameters**

in, out	<i>mp</i>	pointer to the <code>mutex_t</code> object
---------	-----------	--

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.24.4.20 void osalMutexUnlock ( mutex\_t \* mp )**

Unlocks the specified mutex.

**Note**

The HAL guarantees to release mutex in reverse lock order. The mutex being unlocked is guaranteed to be the last locked mutex by the invoking thread. The implementation can rely on this behavior and eventually ignore the *mp* parameter which is supplied in order to support those OSes not supporting a stack of the owned mutexes.

**Parameters**

in, out	<i>mp</i>	pointer to the <code>mutex_t</code> object
---------	-----------	--

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.24.4.21 static void osalSysDisable ( void ) [inline], [static]**

Disables interrupts globally.

**Function Class:**

Special function, this function has special requirements see the notes.

**7.24.4.22 static void osalSysEnable ( void ) [inline], [static]**

Enables interrupts globally.

**Function Class:**

Special function, this function has special requirements see the notes.

**7.24.4.23 static void osalSysLock ( void ) [inline], [static]**

Enters a critical zone from thread context.

**Note**

This function cannot be used for reentrant critical zones.

**Function Class:**

Special function, this function has special requirements see the notes.

**7.24.4.24 static void osalSysUnlock ( void ) [inline], [static]**

Leaves a critical zone from thread context.

**Note**

This function cannot be used for reentrant critical zones.

**Function Class:**

Special function, this function has special requirements see the notes.

**7.24.4.25 static void osalSysLockFromISR ( void ) [inline], [static]**

Enters a critical zone from ISR context.

**Note**

This function cannot be used for reentrant critical zones.

**Function Class:**

Special function, this function has special requirements see the notes.

**7.24.4.26 static void osalSysUnlockFromISR ( void ) [inline], [static]**

Leaves a critical zone from ISR context.

**Note**

This function cannot be used for reentrant critical zones.

**Function Class:**

Special function, this function has special requirements see the notes.

**7.24.4.27 static syssts\_t osalSysGetStatusAndLockX ( void ) [inline], [static]**

Returns the execution status and enters a critical zone.

This functions enters into a critical zone and can be called from any context. Because its flexibility it is less efficient than `chSysLock()` which is preferable when the calling context is known.

**Postcondition**

The system is in a critical zone.

**Returns**

The previous system status, the encoding of this status word is architecture-dependent and opaque.

**Function Class:**

This is an **X-Class API**, this function can be invoked from any context.

**7.24.4.28 static void osalSysRestoreStatusX( syssts\_t sts ) [inline], [static]**

Restores the specified execution status and leaves a critical zone.

**Note**

A call to `chSchRescheduleS()` is automatically performed if exiting the critical zone and if not in ISR context.

**Parameters**

in	<i>sts</i>	the system status to be restored.
----	------------	-----------------------------------

**Function Class:**

This is an **X-Class** API, this function can be invoked from any context.

**7.24.4.29 static bool osalOsIsTimeWithinX ( systime\_t *time*, systime\_t *start*, systime\_t *end* ) [inline], [static]**

Checks if the specified time is within the specified time window.

**Note**

When start==end then the function returns always true because the whole time range is specified.  
This function can be called from any context.

**Parameters**

in	<i>time</i>	the time to be verified
in	<i>start</i>	the start of the time window (inclusive)
in	<i>end</i>	the end of the time window (non inclusive)

**Return values**

<i>true</i>	current time within the specified time window.
<i>false</i>	current time not within the specified time window.

**Function Class:**

This is an **X-Class** API, this function can be invoked from any context.

**7.24.4.30 static void osalThreadQueueObjectInit ( threads\_queue\_t \* *tqp* ) [inline], [static]**

Initializes a threads queue object.

**Parameters**

out	<i>tqp</i>	pointer to the threads queue object
-----	------------	-------------------------------------

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

**7.24.4.31 static void osalEventObjectInit ( event\_source\_t \* *esp* ) [inline], [static]**

Initializes an event flags object.

**Parameters**

out	<i>esp</i>	pointer to the event flags object
-----	------------	-----------------------------------

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

**7.24.4.32 static void osalMutexObjectInit( mutex\_t \* *mp* ) [inline], [static]**

Initializes a mutex\_t object.

**Parameters**

out	<i>mp</i>	pointer to the <code>mutex_t</code> object
-----	-----------	--

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

## 7.24.5 Variable Documentation

### 7.24.5.1 `const char* osal_halt_msg`

Pointer to a halt error message.

**Note**

The message is meant to be retrieved by the debugger after the system halt caused by an unexpected error.

### 7.24.5.2 `const char* osal_halt_msg`

Pointer to a halt error message.

**Note**

The message is meant to be retrieved by the debugger after the system halt caused by an unexpected error.

## 7.25 MII/RMII Header

MII/RMII Support Header.

### 7.25.1 Detailed Description

MII/RMII Support Header.

This header contains definitions and types related to MII/RMII.

#### Generic MII registers

- #define MII\_BMCR 0x00
- #define MII\_BMSR 0x01
- #define MII\_PHYSID1 0x02
- #define MII\_PHYSID2 0x03
- #define MII\_ADVERTISE 0x04
- #define MII\_LPA 0x05
- #define MII\_EXPANSION 0x06
- #define MII\_ANNPTR 0x07
- #define MII\_CTRL1000 0x09
- #define MII\_STAT1000 0xa
- #define MII\_ESTATUS 0x0f
- #define MII\_PHYSTS 0x10
- #define MII\_MICR 0x11
- #define MII\_DCOUNTER 0x12
- #define MII\_FCSCOUNTER 0x13
- #define MII\_NWAYTEST 0x14
- #define MII\_RERRCOUNTER 0x15
- #define MII\_SREVISION 0x16
- #define MII\_RESV1 0x17
- #define MII\_LBRERROR 0x18
- #define MII\_PHYADDR 0x19
- #define MII\_RESV2 0x1a
- #define MII\_TPISTATUS 0x1b
- #define MII\_NCONFIG 0x1c

#### Basic mode control register

- #define BMCR\_RESV 0x007f
- #define BMCR\_CTST 0x0080
- #define BMCR\_FULLDPLX 0x0100
- #define BMCR\_ANRESTART 0x0200
- #define BMCR\_ISOLATE 0x0400
- #define BMCR\_PDOWN 0x0800
- #define BMCR\_ANENABLE 0x1000
- #define BMCR\_SPEED100 0x2000
- #define BMCR\_LOOPBACK 0x4000
- #define BMCR\_RESET 0x8000

### Basic mode status register

- #define BMSR\_ERCAP 0x0001
- #define BMSR\_JCD 0x0002
- #define BMSR\_LSTATUS 0x0004
- #define BMSR\_ANEGCAPABLE 0x0008
- #define BMSR\_RFAULT 0x0010
- #define BMSR\_ANEGCOMPLETE 0x0020
- #define BMSR\_MFPRESUPPCAP 0x0040
- #define BMSR\_RESV 0x0780
- #define BMSR\_10HALF 0x0800
- #define BMSR\_10FULL 0x1000
- #define BMSR\_100HALF 0x2000
- #define BMSR\_100FULL 0x4000
- #define BMSR\_100BASE4 0x8000

### Advertisement control register

- #define ADVERTISE\_SLCT 0x001f
- #define ADVERTISE\_CSMA 0x0001
- #define ADVERTISE\_10HALF 0x0020
- #define ADVERTISE\_10FULL 0x0040
- #define ADVERTISE\_100HALF 0x0080
- #define ADVERTISE\_100FULL 0x0100
- #define ADVERTISE\_100BASE4 0x0200
- #define ADVERTISE\_PAUSE\_CAP 0x0400
- #define ADVERTISE\_PAUSE\_ASYM 0x0800
- #define ADVERTISE\_RESV 0x1000
- #define ADVERTISE\_RFAULT 0x2000
- #define ADVERTISE\_LPACK 0x4000
- #define ADVERTISE\_NPAGE 0x8000
- #define ADVERTISE\_FULL
- #define ADVERTISE\_ALL

### Link partner ability register

- #define LPA\_SLCT 0x001f
- #define LPA\_10HALF 0x0020
- #define LPA\_10FULL 0x0040
- #define LPA\_100HALF 0x0080
- #define LPA\_100FULL 0x0100
- #define LPA\_100BASE4 0x0200
- #define LPA\_PAUSE\_CAP 0x0400
- #define LPA\_PAUSE\_ASYM 0x0800
- #define LPA\_RESV 0x1000
- #define LPA\_RFAULT 0x2000
- #define LPA\_LPACK 0x4000
- #define LPA\_NPAGE 0x8000
- #define LPA\_DUPLEX (LPA\_10FULL | LPA\_100FULL)
- #define LPA\_100 (LPA\_100FULL | LPA\_100HALF | LPA\_100BASE4)

## Expansion register for auto-negotiation

- #define EXPANSION\_NWAY 0x0001
- #define EXPANSION\_LCWP 0x0002
- #define EXPANSION\_ENABLENPAGE 0x0004
- #define EXPANSION\_NPCAPABLE 0x0008
- #define EXPANSION\_MFAULTS 0x0010
- #define EXPANSION\_RESV 0xffe0

## N-way test register

- #define NWAYTEST\_RESV1 0x00ff
- #define NWAYTEST\_LOOPBACK 0x0100
- #define NWAYTEST\_RESV2 0xfe00

## PHY identifiers

- #define MII\_DM9161\_ID 0x0181b8a0
- #define MII\_AM79C875\_ID 0x00225540
- #define MII\_KS8721\_ID 0x00221610
- #define MII\_STE101P\_ID 0x00061C50
- #define MII\_DP83848I\_ID 0x20005C90
- #define MII\_LAN8710A\_ID 0x0007C0F1
- #define MII\_LAN8720\_ID 0x0007C0F0
- #define MII\_LAN8742A\_ID 0x0007C130

## 7.25.2 Macro Definition Documentation

### 7.25.2.1 #define MII\_BMCR 0x00

Basic mode control register.

### 7.25.2.2 #define MII\_BMSR 0x01

Basic mode status register.

### 7.25.2.3 #define MII\_PHYSID1 0x02

PHYS ID 1.

### 7.25.2.4 #define MII\_PHYSID2 0x03

PHYS ID 2.

### 7.25.2.5 #define MII\_ADVERTISE 0x04

Advertisement control reg.

### 7.25.2.6 #define MII\_LPA 0x05

Link partner ability reg.

7.25.2.7 #define MII\_EXPANSION 0x06

Expansion register.

7.25.2.8 #define MII\_ANNPTR 0x07

1000BASE-T control.

7.25.2.9 #define MII\_CTRL1000 0x09

1000BASE-T control.

7.25.2.10 #define MII\_STAT1000 0x0a

1000BASE-T status.

7.25.2.11 #define MII\_ESTATUS 0x0f

Extended Status.

7.25.2.12 #define MII\_PHYSTS 0x10

PHY Status register.

7.25.2.13 #define MII\_MICR 0x11

MII Interrupt ctrl register.

7.25.2.14 #define MII\_DCOUNTER 0x12

Disconnect counter.

7.25.2.15 #define MII\_FCSCOUNTER 0x13

False carrier counter.

7.25.2.16 #define MII\_NWAYTEST 0x14

N-way auto-neg test reg.

7.25.2.17 #define MII\_RERRCOUNTER 0x15

Receive error counter.

7.25.2.18 #define MII\_SREVISION 0x16

Silicon revision.

7.25.2.19 #define MII\_RESERVED1 0x17

Reserved.

7.25.2.20 #define MII\_LBRError 0x18

Lpback, rx, bypass error.

7.25.2.21 #define MII\_PHYADDR 0x19

PHY address.

7.25.2.22 #define MII\_RESERVED2 0x1a

Reserved.

7.25.2.23 #define MII\_TPISTATUS 0x1b

TPI status for 10Mbps.

7.25.2.24 #define MII\_NCONFIG 0x1c

Network interface config.

7.25.2.25 #define BMCR\_RESERVED 0x007f

Unused.

7.25.2.26 #define BMCR\_CTST 0x0080

Collision test.

7.25.2.27 #define BMCR\_FULLDPLX 0x0100

Full duplex.

7.25.2.28 #define BMCR\_ANRESTART 0x0200

Auto negotiation restart.

7.25.2.29 #define BMCR\_ISOLATE 0x0400

Disconnect DP83840 from MII.

7.25.2.30 #define BMCR\_PDOWN 0x0800

Powerdown.

7.25.2.31 #define BMCR\_ANENABLE 0x1000

Enable auto negotiation.

7.25.2.32 #define BMCR\_SPEED100 0x2000

Select 100Mbps.

7.25.2.33 #define BMCR\_LOOPBACK 0x4000

TXD loopback bit.

7.25.2.34 #define BMCR\_RESET 0x8000

Reset.

7.25.2.35 #define BMSR\_ERCAP 0x0001

Ext-reg capability.

7.25.2.36 #define BMSR\_JCD 0x0002

Jabber detected.

7.25.2.37 #define BMSR\_LSTATUS 0x0004

Link status.

7.25.2.38 #define BMSR\_ANEGCAPABLE 0x0008

Able to do auto-negotiation.

7.25.2.39 #define BMSR\_RFAULT 0x0010

Remote fault detected.

7.25.2.40 #define BMSR\_ANEGCOMPLETE 0x0020

Auto-negotiation complete.

7.25.2.41 #define BMSR\_MFPRESUPPCAP 0x0040

Able to suppress preamble.

7.25.2.42 #define BMSR\_RESV 0x0780

Unused.

7.25.2.43 #define BMSR\_10HALF 0x0800

Can do 10mbps, half-duplex.

7.25.2.44 #define BMSR\_10FULL 0x1000

Can do 10mbps, full-duplex.

7.25.2.45 #define BMSR\_100HALF 0x2000

Can do 100mbps, half-duplex.

7.25.2.46 #define BMSR\_100FULL 0x4000

Can do 100mbps, full-duplex.

7.25.2.47 #define BMSR\_100BASE4 0x8000

Can do 100mbps, 4k packets.

7.25.2.48 #define ADVERTISE\_SLCT 0x001f

Selector bits.

7.25.2.49 #define ADVERTISE\_CSMA 0x0001

Only selector supported.

7.25.2.50 #define ADVERTISE\_10HALF 0x0020

Try for 10mbps half-duplex.

7.25.2.51 #define ADVERTISE\_10FULL 0x0040

Try for 10mbps full-duplex.

7.25.2.52 #define ADVERTISE\_100HALF 0x0080

Try for 100mbps half-duplex.

7.25.2.53 #define ADVERTISE\_100FULL 0x0100

Try for 100mbps full-duplex.

7.25.2.54 #define ADVERTISE\_100BASE4 0x0200

Try for 100mbps 4k packets.

7.25.2.55 #define ADVERTISE\_PAUSE\_CAP 0x0400

Try for pause.

7.25.2.56 #define ADVERTISE\_PAUSE\_ASYM 0x0800

Try for asymmetric pause.

7.25.2.57 #define ADVERTISE\_RESV 0x1000

Unused.

7.25.2.58 #define ADVERTISE\_RFAULT 0x2000

Say we can detect faults.

7.25.2.59 #define ADVERTISE\_LPACK 0x4000

Ack link partners response.

7.25.2.60 #define ADVERTISE\_NPAGE 0x8000

Next page bit.

7.25.2.61 #define LPA\_SLCT 0x001f

Same as advertise selector.

7.25.2.62 #define LPA\_10HALF 0x0020

Can do 10mbps half-duplex.

7.25.2.63 #define LPA\_10FULL 0x0040

Can do 10mbps full-duplex.

7.25.2.64 #define LPA\_100HALF 0x0080

Can do 100mbps half-duplex.

7.25.2.65 #define LPA\_100FULL 0x0100

Can do 100mbps full-duplex.

7.25.2.66 #define LPA\_100BASE4 0x0200

Can do 100mbps 4k packets.

7.25.2.67 #define LPA\_PAUSE\_CAP 0x0400

Can pause.

7.25.2.68 #define LPA\_PAUSE\_ASYM 0x0800

Can pause asymmetrically.

7.25.2.69 #define LPA\_RESV 0x1000

Unused.

7.25.2.70 #define LPA\_RFAULT 0x2000

Link partner faulted.

7.25.2.71 #define LPA\_LPACK 0x4000

Link partner acked us.

7.25.2.72 #define LPA\_NPAGE 0x8000

Next page bit.

7.25.2.73 #define EXPANSION\_NWAY 0x0001

Can do N-way auto-nego.

7.25.2.74 #define EXPANSION\_LCWP 0x0002

Got new RX page code word.

7.25.2.75 #define EXPANSION\_ENABLENPAGE 0x0004

This enables npage words.

7.25.2.76 #define EXPANSION\_NPCAPABLE 0x0008

Link partner supports npage.

7.25.2.77 #define EXPANSION\_MFAULTS 0x0010

Multiple faults detected.

7.25.2.78 #define EXPANSION\_RESV 0xffe0

Unused.

7.25.2.79 #define NWAYTEST\_RESERVED1 0x00ff

Unused.

7.25.2.80 #define NWAYTEST\_LOOPBACK 0x0100

Enable loopback for N-way.

7.25.2.81 #define NWAYTEST\_RESERVED2 0xfe00

Unused.

## 7.26 MMC over SPI Driver

Generic MMC driver.

### 7.26.1 Detailed Description

Generic MMC driver.

This module implements a portable MMC/SD driver that uses a SPI driver as physical layer. Hot plugging and removal are supported through kernel events.

#### Precondition

In order to use the MMC\_SPI driver the HAL\_USE\_MMC\_SPI and HAL\_USE\_SPI options must be enabled in `halconf.h`.

### 7.26.2 Driver State Machine

This driver implements a state machine internally, see the [Abstract I/O Block Device](#) module documentation for details.

### 7.26.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

#### Macros

- `#define _mmc_driver_methods _mmcsd_block_device_methods`  
*MMCDriver specific methods.*

#### MMC\_SPI configuration options

- `#define MMC_NICE_WAITING TRUE`  
*Delays insertions.*

#### Macro Functions

- `#define mmclsCardInserted(mmcp) mmc_lld_is_card_inserted(mmcp)`  
*Returns the card insertion status.*
- `#define mmclsWriteProtected(mmcp) mmc_lld_is_write_protected(mmcp)`  
*Returns the write protect status.*

#### Data Structures

- struct `MMCConfig`  
*MMC/SD over SPI driver configuration structure.*
- struct `MMCDriverVMT`  
*MMCDriver virtual methods table.*
- struct `MMCDriver`  
*Structure representing a MMC/SD over SPI driver.*

## Functions

- static uint8\_t **crc7** (uint8\_t crc, const uint8\_t \*buffer, size\_t len)  
*Calculate the MMC standard CRC-7 based on a lookup table.*
- static void **wait** (MMCDriver \*mmcp)  
*Waits an idle condition.*
- static void **send\_hdr** (MMCDriver \*mmcp, uint8\_t cmd, uint32\_t arg)  
*Sends a command header.*
- static uint8\_t **recv1** (MMCDriver \*mmcp)  
*Receives a single byte response.*
- static uint8\_t **recv3** (MMCDriver \*mmcp, uint8\_t \*buffer)  
*Receives a three byte response.*
- static uint8\_t **send\_command\_R1** (MMCDriver \*mmcp, uint8\_t cmd, uint32\_t arg)  
*Sends a command and returns a single byte response.*
- static uint8\_t **send\_command\_R3** (MMCDriver \*mmcp, uint8\_t cmd, uint32\_t arg, uint8\_t \*response)  
*Sends a command which returns a five bytes response (R3).*
- static bool **read\_CxD** (MMCDriver \*mmcp, uint8\_t cmd, uint32\_t cxd[4])  
*Reads the CSD.*
- static void **sync** (MMCDriver \*mmcp)  
*Waits that the card reaches an idle state.*
- void **mmcInit** (void)  
*MMC over SPI driver initialization.*
- void **mmcObjectInit** (MMCDriver \*mmcp)  
*Initializes an instance.*
- void **mmcStart** (MMCDriver \*mmcp, const MMCCConfig \*config)  
*Configures and activates the MMC peripheral.*
- void **mmcStop** (MMCDriver \*mmcp)  
*Disables the MMC peripheral.*
- bool **mmcConnect** (MMCDriver \*mmcp)  
*Performs the initialization procedure on the inserted card.*
- bool **mmcDisconnect** (MMCDriver \*mmcp)  
*Brings the driver in a state safe for card removal.*
- bool **mmcStartSequentialRead** (MMCDriver \*mmcp, uint32\_t startblk)  
*Starts a sequential read.*
- bool **mmcSequentialRead** (MMCDriver \*mmcp, uint8\_t \*buffer)  
*Reads a block within a sequential read operation.*
- bool **mmcStopSequentialRead** (MMCDriver \*mmcp)  
*Stops a sequential read gracefully.*
- bool **mmcStartSequentialWrite** (MMCDriver \*mmcp, uint32\_t startblk)  
*Starts a sequential write.*
- bool **mmcSequentialWrite** (MMCDriver \*mmcp, const uint8\_t \*buffer)  
*Writes a block within a sequential write operation.*
- bool **mmcStopSequentialWrite** (MMCDriver \*mmcp)  
*Stops a sequential write gracefully.*
- bool **mmcSync** (MMCDriver \*mmcp)  
*Waits for card idle condition.*
- bool **mmcGetInfo** (MMCDriver \*mmcp, BlockDeviceInfo \*bdip)  
*Returns the media info.*
- bool **mmcErase** (MMCDriver \*mmcp, uint32\_t startblk, uint32\_t endblk)  
*Erases blocks.*

## Variables

- static const struct [MMCDriverVMT](#) mmc\_vmt  
*Virtual methods table.*
- static const uint8\_t [crc7\\_lookup\\_table](#) [256]  
*Lookup table for CRC-7 ( based on polynomial  $x^7 + x^3 + 1$ ).*

### 7.26.4 Macro Definition Documentation

#### 7.26.4.1 #define MMC\_NICE\_WAITING TRUE

Delays insertions.

If enabled this option inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however. This option is recommended also if the SPI driver does not use a DMA channel and heavily loads the CPU.

#### 7.26.4.2 #define \_mmc\_driver\_methods \_mmcsd\_block\_device\_methods

[MMCDriver](#) specific methods.

#### 7.26.4.3 #define mmclsCardInserted( *mmcp* ) mmc\_lld\_is\_card\_inserted(*mmcp*)

Returns the card insertion status.

##### Note

This macro wraps a low level function named `sdc_lld_is_card_inserted()`, this function must be provided by the application because it is not part of the SDC driver.

##### Parameters

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
----	-------------	---

##### Returns

The card state.

##### Return values

<i>FALSE</i>	card not inserted.
<i>TRUE</i>	card inserted.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.26.4.4 #define mmclsWriteProtected( *mmcp* ) mmc\_lld\_is\_write\_protected(*mmcp*)

Returns the write protect status.

**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
----	-------------	---

**Returns**

The card state.

**Return values**

<i>FALSE</i>	card not inserted.
<i>TRUE</i>	card inserted.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.26.5 Function Documentation****7.26.5.1 static uint8\_t crc7( uint8\_t *crc*, const uint8\_t \* *buffer*, size\_t *len* ) [static]**

Calculate the MMC standard CRC-7 based on a lookup table.

**Parameters**

in	<i>crc</i>	start value for CRC
in	<i>buffer</i>	pointer to data buffer
in	<i>len</i>	length of data

**Returns**

Calculated CRC

**7.26.5.2 static void wait( MMCDriver \* *mmcp* ) [static]**

Waits an idle condition.

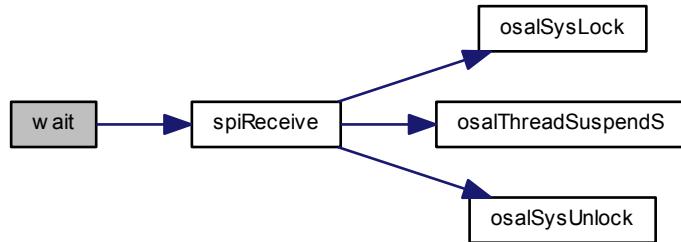
**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.26.5.3 static void send\_hdr ( **MMCDriver** \* *mmcp*, uint8\_t *cmd*, uint32\_t *arg* ) [static]

Sends a command header.

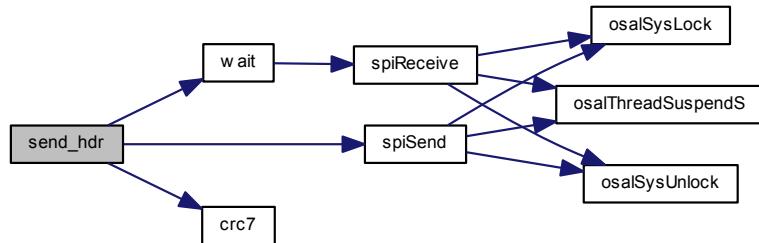
##### Parameters

in	<i>mmcp</i>	pointer to the <b>MMCDriver</b> object
in	<i>cmd</i>	the command id
in	<i>arg</i>	the command argument

##### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.26.5.4 static uint8\_t recv1 ( **MMCDriver** \* *mmcp* ) [static]

Receives a single byte response.

**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
----	-------------	---

**Returns**

The response as an `uint8_t` value.

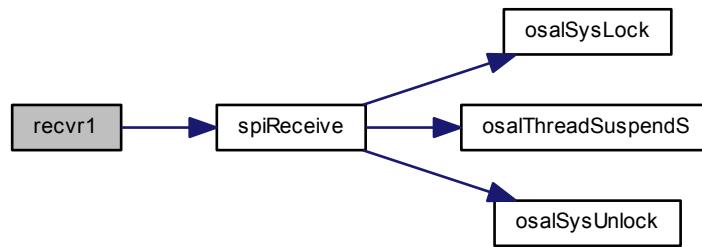
**Return values**

<i>0xFF</i>	timed out.
-------------	------------

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.26.5.5 static uint8\_t recv3 ( [MMCDriver](#) \* *mmcp*, `uint8_t` \* *buffer* ) [static]**

Receives a three byte response.

**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
out	<i>buffer</i>	pointer to four bytes wide buffer

**Returns**

First response byte as an `uint8_t` value.

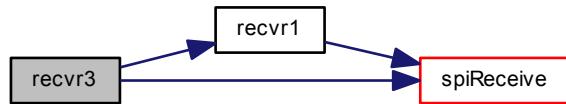
**Return values**

<i>0xFF</i>	timed out.
-------------	------------

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.26.5.6 static uint8\_t send\_command\_R1 ( **MMCDriver** \* mmcp, uint8\_t cmd, uint32\_t arg ) [static]

Sends a command and returns a single byte response.

##### Parameters

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
in	<i>cmd</i>	the command id
in	<i>arg</i>	the command argument

##### Returns

The response as an `uint8_t` value.

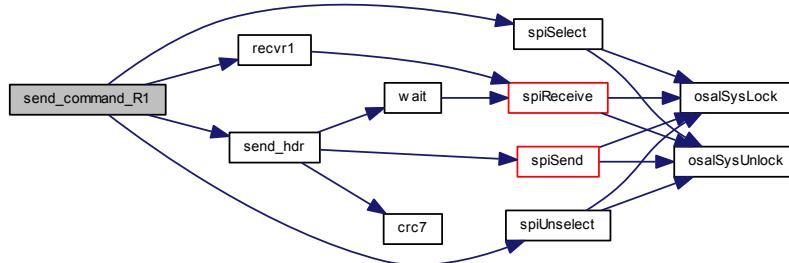
##### Return values

<code>0xFF</code>	timed out.
-------------------	------------

##### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.26.5.7 static uint8\_t send\_command\_R3 ( **MMCDriver** \* mmcp, uint8\_t cmd, uint32\_t arg, uint8\_t \* response ) [static]

Sends a command which returns a five bytes response (R3).

**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
in	<i>cmd</i>	the command id
in	<i>arg</i>	the command argument
out	<i>response</i>	pointer to four bytes wide uint8_t buffer

**Returns**

The first byte of the response (R1) as an `uint8_t` value.

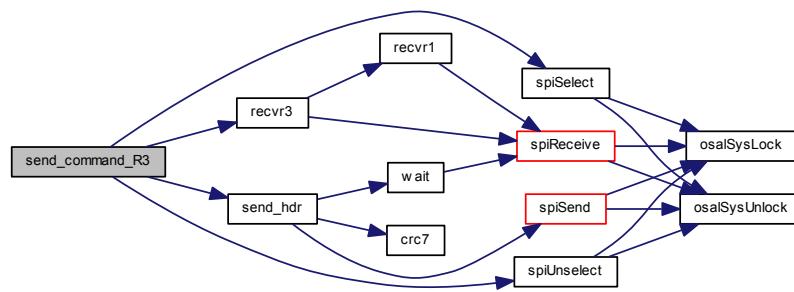
**Return values**

<code>0xFF</code>	timed out.
-------------------	------------

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.26.5.8 static bool read\_CxD ( [MMCDriver](#) \* *mmcp*, `uint8_t` *cmd*, `uint32_t` *cxd[4]* ) [static]**

Reads the CSD.

**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
out	<i>cmd</i>	command
out	<i>cxd</i>	pointer to the CSD/CID buffer

**Returns**

The operation status.

**Return values**

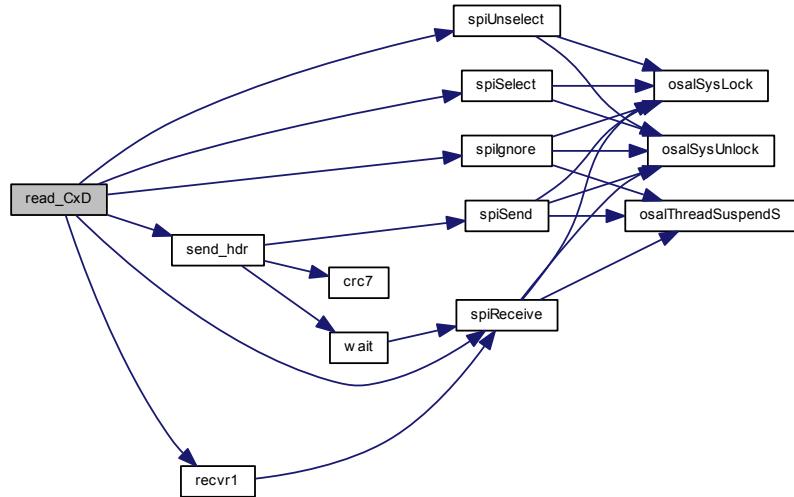
<code>HAL_SUCCESS</code>	the operation succeeded.
--------------------------	--------------------------

<code>HAL_FAILED</code>	the operation failed.
-------------------------	-----------------------

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.26.5.9 static void sync ( MMCDriver \* mmcp ) [static]**

Waits that the card reaches an idle state.

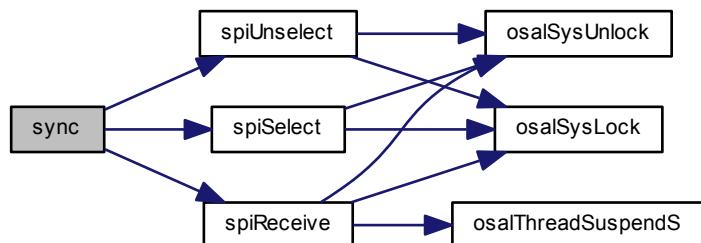
**Parameters**

in	<code>mmcp</code>	pointer to the <a href="#">MMCDriver</a> object
----	-------------------	---

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



**7.26.5.10 void mmcInit( void )**

MMC over SPI driver initialization.

**Note**

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

**7.26.5.11 void mmcObjectInit( MMCDriver \* mmcp )**

Initializes an instance.

**Parameters**

out	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
-----	-------------	---

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

**7.26.5.12 void mmcStart( MMCDriver \* mmcp, const MMCCConfig \* config )**

Configures and activates the MMC peripheral.

**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
in	<i>config</i>	pointer to the <a href="#">MMCCConfig</a> object.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.26.5.13 void mmcStop( MMCDriver \* mmcp )**

Disables the MMC peripheral.

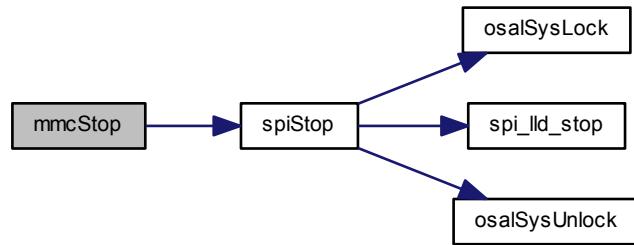
**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.26.5.14 bool mmcConnect ( [MMCDriver](#) \* *mmcp* )

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the `MMC_READY` state where it is possible to perform read and write operations.

#### Note

It is possible to invoke this function from the insertion event handler.

#### Parameters

<code>in</code>	<code>mmcp</code>	pointer to the <a href="#">MMCDriver</a> object
-----------------	-------------------	---

#### Returns

The operation status.

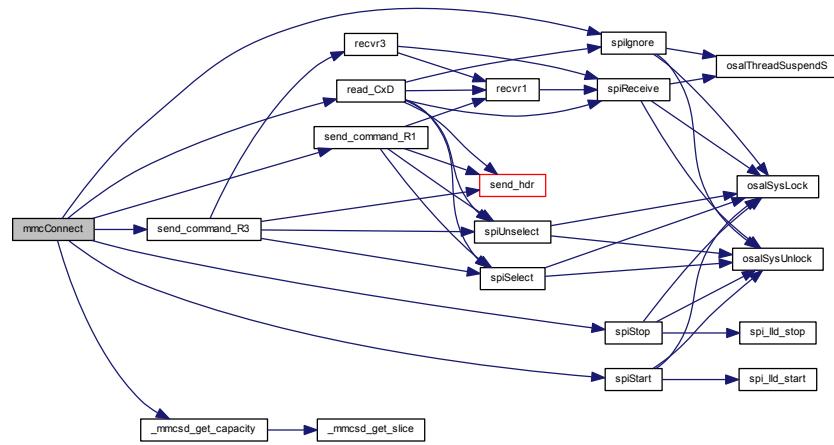
#### Return values

<code>HAL_SUCCESS</code>	the operation succeeded and the driver is now in the <code>MMC_READY</code> state.
<code>HAL_FAILED</code>	the operation failed.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.26.5.15 bool mmcDisconnect ( [MMCDriver](#) \* *mmcp* )

Brings the driver in a state safe for card removal.

##### Parameters

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
----	-------------	---

##### Returns

The operation status.

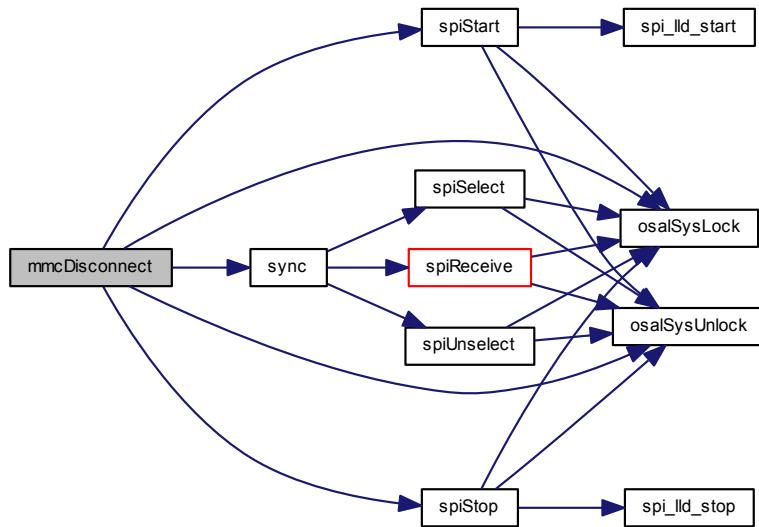
##### Return values

<i>HAL_SUCCESS</i>	the operation succeeded and the driver is now in the <a href="#">MMC_INSERTED</a> state.
<i>HAL_FAILED</i>	the operation failed.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.26.5.16 bool mmcStartSequentialRead ( `MMCDriver * mmcp`, `uint32_t startblk` )

Starts a sequential read.

##### Parameters

<code>in</code>	<code>mmcp</code>	pointer to the <code>MMCDriver</code> object
<code>in</code>	<code>startblk</code>	first block to read

##### Returns

The operation status.

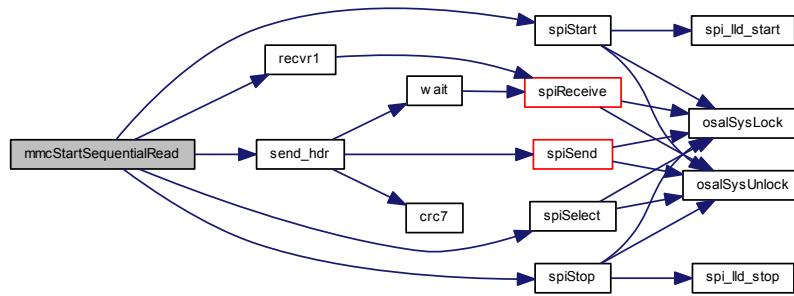
##### Return values

<code>HAL_SUCCESS</code>	the operation succeeded.
<code>HAL_FAILED</code>	the operation failed.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.26.5.17 bool mmcSequentialRead ( MMCDriver \* mmcp, uint8\_t \* buffer )

Reads a block within a sequential read operation.

##### Parameters

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
out	<i>buffer</i>	pointer to the read buffer

##### Returns

The operation status.

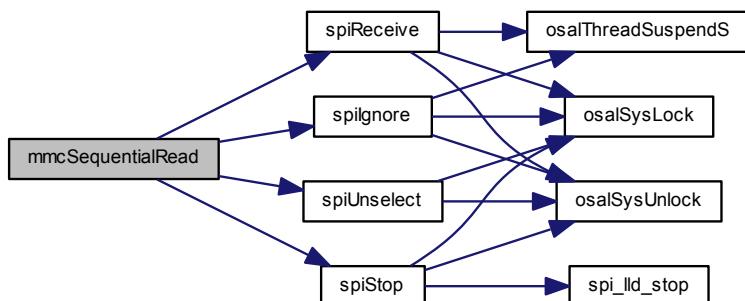
##### Return values

<a href="#">HAL_SUCCESS</a>	the operation succeeded.
<a href="#">HAL_FAILED</a>	the operation failed.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



**7.26.5.18 bool mmcStopSequentialRead ( **MMCDriver** \* *mmcp* )**

Stops a sequential read gracefully.

**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
----	-------------	---

**Returns**

The operation status.

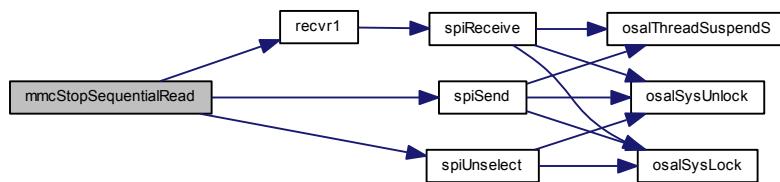
**Return values**

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	the operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.26.5.19 bool mmcStartSequentialWrite( [MMCDriver](#) \* *mmcp*, uint32\_t *startblk* )**

Starts a sequential write.

**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
in	<i>startblk</i>	first block to write

**Returns**

The operation status.

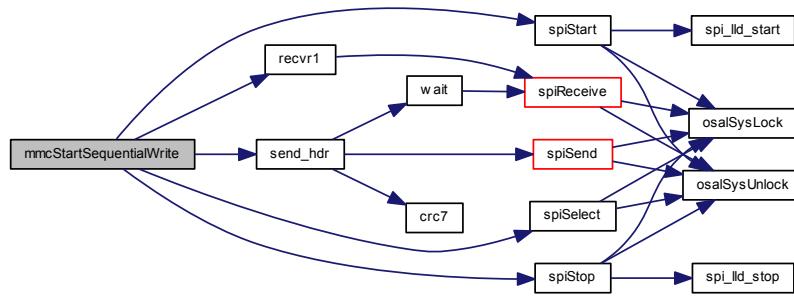
**Return values**

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	the operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.26.5.20 bool mmcSequentialWrite ( MMCDDriver \* mmcp, const uint8\_t \* buffer )

Writes a block within a sequential write operation.

##### Parameters

in	<i>mmcp</i>	pointer to the <a href="#">MMCDDriver</a> object
out	<i>buffer</i>	pointer to the write buffer

##### Returns

The operation status.

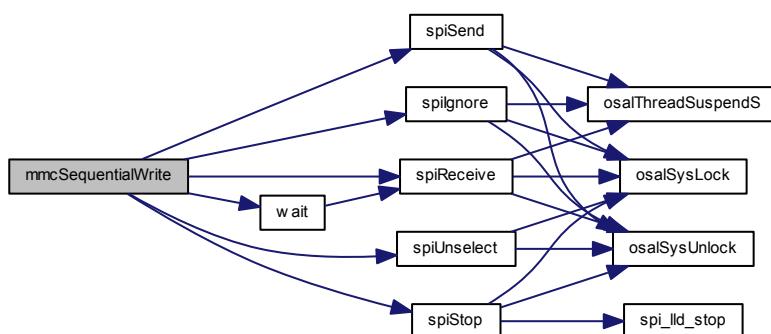
##### Return values

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	the operation failed.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.26.5.21 bool mmcStopSequentialWrite ( **MMCDriver** \* *mmcp* )

Stops a sequential write gracefully.

**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
----	-------------	---

**Returns**

The operation status.

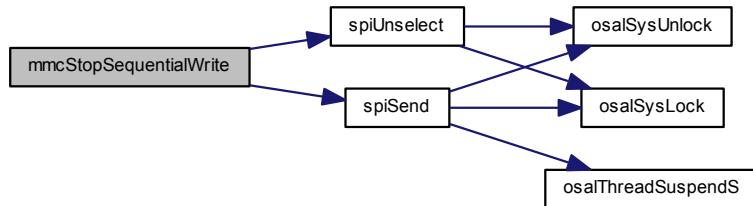
**Return values**

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	the operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.26.5.22 bool mmcSync ( MMCDriver \* *mmcp* )**

Waits for card idle condition.

**Parameters**

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
----	-------------	---

**Returns**

The operation status.

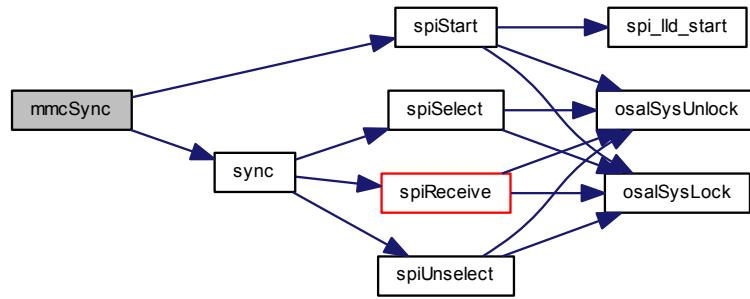
**Return values**

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	the operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.26.5.23 bool mmcGetInfo ( **MMCDriver** \* *mmcp*, **BlockDeviceInfo** \* *bdip* )

Returns the media info.

##### Parameters

in	<i>mmcp</i>	pointer to the <b>MMCDriver</b> object
out	<i>bdip</i>	pointer to a <b>BlockDeviceInfo</b> structure

##### Returns

The operation status.

##### Return values

<b>HAL_SUCCESS</b>	the operation succeeded.
<b>HAL_FAILED</b>	the operation failed.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.26.5.24 bool mmcErase ( **MMCDriver** \* *mmcp*, **uint32\_t** *startblk*, **uint32\_t** *endblk* )

Erases blocks.

##### Parameters

in	<i>mmcp</i>	pointer to the <b>MMCDriver</b> object
in	<i>startblk</i>	starting block number
in	<i>endblk</i>	ending block number

##### Returns

The operation status.

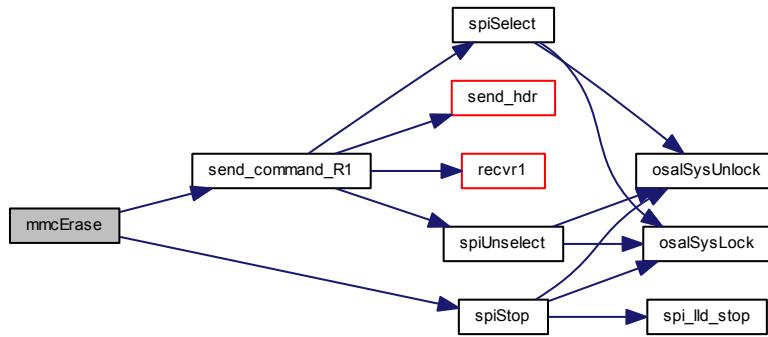
**Return values**

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	the operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



## 7.26.6 Variable Documentation

### 7.26.6.1 const struct MMCDriverVMT mmc\_vmt [static]

**Initial value:**

```
= {
    (bool (*)(void *))mmc_lld_is_card_inserted,
    (bool (*)(void *))mmc_lld_is_write_protected,
    (bool (*)(void *))mmcConnect,
    (bool (*)(void *))mmcDisconnect,
    mmc_read,
    mmc_write,
    (bool (*)(void *))mmcSync,
    (bool (*)(void *, BlockDeviceInfo *))mmcGetInfo
}
```

Virtual methods table.

### 7.26.6.2 const uint8\_t crc7\_lookup\_table[256] [static]

**Initial value:**

```
= {
    0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a, 0x53,
    0x6c, 0x65, 0x7e, 0x77, 0x19, 0x10, 0xb, 0x02, 0x3d, 0x34, 0x2f, 0x26,
    0x51, 0x58, 0x43, 0xa, 0x75, 0x7c, 0x67, 0x6e, 0x32, 0x3b, 0x20, 0x29,
    0x16, 0x1f, 0x04, 0xd, 0x7a, 0x73, 0x68, 0x61, 0x5e, 0x57, 0x4c, 0x45,
    0x2b, 0x22, 0x39, 0x30, 0xf, 0x06, 0x1d, 0x14, 0x63, 0x6a, 0x71, 0x78,
    0x47, 0x4e, 0x55, 0x5c, 0x64, 0x6d, 0x76, 0x7f, 0x40, 0x49, 0x52, 0x5b,
    0x2c, 0x25, 0x3e, 0x37, 0x08, 0x01, 0x1a, 0x13, 0x7d, 0x74, 0x6f, 0x66,
    0x59, 0x50, 0x4b, 0x42, 0x35, 0x3c, 0x27, 0x2e, 0x11, 0x18, 0x03, 0xa,
    0x56, 0x5f, 0x44, 0x4d, 0x72, 0x7b, 0x60, 0x69, 0x1e, 0x17, 0x0c, 0x05,
    0x3a, 0x33, 0x28, 0x21, 0x4f, 0x46, 0x5d, 0x54, 0x6b, 0x62, 0x79, 0x70,
    0x07, 0xe, 0x15, 0x1c, 0x23, 0x2a, 0x31, 0x38, 0x41, 0x48, 0x53, 0x5a,
```

```
0x65, 0x6c, 0x77, 0x7e, 0x09, 0x00, 0x1b, 0x12, 0x2d, 0x24, 0x3f, 0x36,
0x58, 0x51, 0x4a, 0x43, 0x7c, 0x75, 0x6e, 0x67, 0x10, 0x19, 0x02, 0x0b,
0x34, 0x3d, 0x26, 0x2f, 0x73, 0x7a, 0x61, 0x68, 0x57, 0x5e, 0x45, 0x4c,
0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x6a, 0x63, 0x78, 0x71,
0x4e, 0x47, 0x5c, 0x55, 0x22, 0x2b, 0x30, 0x39, 0x06, 0x0f, 0x14, 0x1d,
0x25, 0x2c, 0x37, 0x3e, 0x01, 0x08, 0x13, 0x1a, 0x6d, 0x64, 0x7f, 0x76,
0x49, 0x40, 0x5b, 0x52, 0x3c, 0x35, 0x2e, 0x27, 0x18, 0x11, 0x0a, 0x03,
0x74, 0x7d, 0x66, 0x6f, 0x50, 0x59, 0x42, 0x4b, 0x17, 0x1e, 0x05, 0x0c,
0x33, 0x3a, 0x21, 0x28, 0x5f, 0x56, 0x4d, 0x44, 0x7b, 0x72, 0x69, 0x60,
0x0e, 0x07, 0x1c, 0x15, 0x2a, 0x23, 0x38, 0x31, 0x46, 0x4f, 0x54, 0x5d,
0x62, 0x6b, 0x70, 0x79
}
```

Lookup table for CRC-7 ( based on polynomial  $x^7 + x^3 + 1$ ).

## 7.27 MMC/SD Block Device

### 7.27.1 Detailed Description

This module implements a common ancestor for all device drivers accessing MMC or SD cards. This interface inherits the state machine and the interface from the [Abstract I/O Block Device](#) module.

#### Macros

- `#define MMCSD_BLOCK_SIZE 512U`  
*Fixed block size for MMC/SD block devices.*
- `#define MMCSD_R1_ERROR_MASK 0xFDFE008U`  
*Mask of error bits in R1 responses.*
- `#define MMCSD_CMD8_PATTERN 0x000001AAU`  
*Fixed pattern for CMD8.*
- `#define _mmcsd_block_device_methods _base_block_device_methods`  
*MMCSDBlockDevice specific methods.*
- `#define _mmcsd_block_device_data`  
*MMCSDBlockDevice specific data.*

#### SD/MMC status conditions

- `#define MMCSD_STS_IDLE 0U`
- `#define MMCSD_STS_READY 1U`
- `#define MMCSD_STS_IDENT 2U`
- `#define MMCSD_STS_STBY 3U`
- `#define MMCSD_STS_TRAN 4U`
- `#define MMCSD_STS_DATA 5U`
- `#define MMCSD_STS_RCV 6U`
- `#define MMCSD_STS_PRG 7U`
- `#define MMCSD_STS_DIS 8U`

#### SD/MMC commands

- `#define MMCSD_CMD_GO_IDLE_STATE 0U`
- `#define MMCSD_CMD_INIT 1U`
- `#define MMCSD_CMD_ALL_SEND_CID 2U`
- `#define MMCSD_CMD_SEND_RELATIVE_ADDR 3U`
- `#define MMCSD_CMD_SET_BUS_WIDTH 6U`
- `#define MMCSD_CMD_SWITCH MMCSD_CMD_SET_BUS_WIDTH`
- `#define MMCSD_CMD_SEL_DESEL_CARD 7U`
- `#define MMCSD_CMD_SEND_IF_COND 8U`
- `#define MMCSD_CMD_SEND_EXT_CSD MMCSD_CMD_SEND_IF_COND`
- `#define MMCSD_CMD_SEND_CSD 9U`
- `#define MMCSD_CMD_SEND_CID 10U`
- `#define MMCSD_CMD_STOP_TRANSMISSION 12U`
- `#define MMCSD_CMD_SEND_STATUS 13U`
- `#define MMCSD_CMD_SET_BLOCKLEN 16U`
- `#define MMCSD_CMD_READ_SINGLE_BLOCK 17U`
- `#define MMCSD_CMD_READ_MULTIPLE_BLOCK 18U`
- `#define MMCSD_CMD_SET_BLOCK_COUNT 23U`
- `#define MMCSD_CMD_WRITE_BLOCK 24U`

- #define **MMCSD\_CMD\_WRITE\_MULTIPLE\_BLOCK** 25U
- #define **MMCSD\_CMD\_ERASE\_RW\_BLK\_START** 32U
- #define **MMCSD\_CMD\_ERASE\_RW\_BLK\_END** 33U
- #define **MMCSD\_CMD\_ERASE** 38U
- #define **MMCSD\_CMD\_APP\_OP\_COND** 41U
- #define **MMCSD\_CMD\_LOCK\_UNLOCK** 42U
- #define **MMCSD\_CMD\_APP\_CMD** 55U
- #define **MMCSD\_CMD\_READ\_OCR** 58U

### CSD record offsets

- #define **MMCSD\_CSD\_MMC\_CSD\_STRUCTURE\_SLICE** 127U,126U  
*Slice position of values in CSD register.*
- #define **MMCSD\_CSD\_MMC\_SPEC\_VERS\_SLICE** 125U,122U
- #define **MMCSD\_CSD\_MMC\_TAAC\_SLICE** 119U,112U
- #define **MMCSD\_CSD\_MMC\_NSAC\_SLICE** 111U,104U
- #define **MMCSD\_CSD\_MMC\_TRAN\_SPEED\_SLICE** 103U,96U
- #define **MMCSD\_CSD\_MMC\_CCC\_SLICE** 95U,84U
- #define **MMCSD\_CSD\_MMC\_READ\_BL\_LEN\_SLICE** 83U,80U
- #define **MMCSD\_CSD\_MMC\_READ\_BL\_PARTIAL\_SLICE** 79U,79U
- #define **MMCSD\_CSD\_MMC\_WRITE\_BLK\_MISALIGN\_SLICE** 78U,78U
- #define **MMCSD\_CSD\_MMC\_READ\_BLK\_MISALIGN\_SLICE** 77U,77U
- #define **MMCSD\_CSD\_MMC\_DSR\_IMP\_SLICE** 76U,76U
- #define **MMCSD\_CSD\_MMC\_C\_SIZE\_SLICE** 73U,62U
- #define **MMCSD\_CSD\_MMC\_VDD\_R\_CURR\_MIN\_SLICE** 61U,59U
- #define **MMCSD\_CSD\_MMC\_VDD\_R\_CURR\_MAX\_SLICE** 58U,56U
- #define **MMCSD\_CSD\_MMC\_VDD\_W\_CURR\_MIN\_SLICE** 55U,53U
- #define **MMCSD\_CSD\_MMC\_VDD\_W\_CURR\_MAX\_SLICE** 52U,50U
- #define **MMCSD\_CSD\_MMC\_C\_SIZE\_MULT\_SLICE** 49U,47U
- #define **MMCSD\_CSD\_MMC\_ERASE\_GRP\_SIZE\_SLICE** 46U,42U
- #define **MMCSD\_CSD\_MMC\_ERASE\_GRP\_MULT\_SLICE** 41U,37U
- #define **MMCSD\_CSD\_MMC\_WP\_GRP\_SIZE\_SLICE** 36U,32U
- #define **MMCSD\_CSD\_MMC\_WP\_GRP\_ENABLE\_SLICE** 31U,31U
- #define **MMCSD\_CSD\_MMC\_DEFAULT\_ECC\_SLICE** 30U,29U
- #define **MMCSD\_CSD\_MMC\_R2W\_FACTOR\_SLICE** 28U,26U
- #define **MMCSD\_CSD\_MMC\_WRITE\_BL\_LEN\_SLICE** 25U,22U
- #define **MMCSD\_CSD\_MMC\_WRITE\_BL\_PARTIAL\_SLICE** 21U,21U
- #define **MMCSD\_CSD\_MMC\_CONTENT\_PROT\_APP\_SLICE** 16U,16U
- #define **MMCSD\_CSD\_MMC\_FILE\_FORMAT\_GRP\_SLICE** 15U,15U
- #define **MMCSD\_CSD\_MMC\_COPY\_SLICE** 14U,14U
- #define **MMCSD\_CSD\_MMC\_PERM\_WRITE\_PROTECT\_SLICE** 13U,13U
- #define **MMCSD\_CSD\_MMC\_TMP\_WRITE\_PROTECT\_SLICE** 12U,12U
- #define **MMCSD\_CSD\_MMC\_FILE\_FORMAT\_SLICE** 11U,10U
- #define **MMCSD\_CSD\_MMC\_ECC\_SLICE** 9U,8U
- #define **MMCSD\_CSD\_MMC\_CRC\_SLICE** 7U,1U
- #define **MMCSD\_CSD\_20\_CRC\_SLICE** 7U,1U
- #define **MMCSD\_CSD\_20\_FILE\_FORMAT\_SLICE** 11U,10U
- #define **MMCSD\_CSD\_20\_TMP\_WRITE\_PROTECT\_SLICE** 12U,12U
- #define **MMCSD\_CSD\_20\_PERM\_WRITE\_PROTECT\_SLICE** 13U,13U
- #define **MMCSD\_CSD\_20\_COPY\_SLICE** 14U,14U
- #define **MMCSD\_CSD\_20\_FILE\_FORMAT\_GRP\_SLICE** 15U,15U
- #define **MMCSD\_CSD\_20\_WRITE\_BL\_PARTIAL\_SLICE** 21U,21U
- #define **MMCSD\_CSD\_20\_WRITE\_BL\_LEN\_SLICE** 25U,12U
- #define **MMCSD\_CSD\_20\_R2W\_FACTOR\_SLICE** 28U,26U

- #define **MMCSD\_CSD\_20\_WP\_GRP\_ENABLE\_SLICE** 31U,31U
- #define **MMCSD\_CSD\_20\_WP\_GRP\_SIZE\_SLICE** 38U,32U
- #define **MMCSD\_CSD\_20\_ERASE\_SECTOR\_SIZE\_SLICE** 45U,39U
- #define **MMCSD\_CSD\_20\_ERASE\_BLK\_EN\_SLICE** 46U,46U
- #define **MMCSD\_CSD\_20\_C\_SIZE\_SLICE** 69U,48U
- #define **MMCSD\_CSD\_20\_DSR\_IMP\_SLICE** 76U,76U
- #define **MMCSD\_CSD\_20\_READ\_BLK\_MISALIGN\_SLICE** 77U,77U
- #define **MMCSD\_CSD\_20\_WRITE\_BLK\_MISALIGN\_SLICE** 78U,78U
- #define **MMCSD\_CSD\_20\_READ\_BL\_PARTIAL\_SLICE** 79U,79U
- #define **MMCSD\_CSD\_20\_READ\_BL\_LEN\_SLICE** 83U,80U
- #define **MMCSD\_CSD\_20\_CCC\_SLICE** 95U,84U
- #define **MMCSD\_CSD\_20\_TRANS\_SPEED\_SLICE** 103U,96U
- #define **MMCSD\_CSD\_20\_NSAC\_SLICE** 111U,104U
- #define **MMCSD\_CSD\_20\_TAAC\_SLICE** 119U,112U
- #define **MMCSD\_CSD\_20\_CSD\_STRUCTURE\_SLICE** 127U,126U
- #define **MMCSD\_CSD\_10\_CRC\_SLICE** MMCSD\_CSD\_20\_CRC\_SLICE
- #define **MMCSD\_CSD\_10\_FILE\_FORMAT\_SLICE** MMCSD\_CSD\_20\_FILE\_FORMAT\_SLICE
- #define **MMCSD\_CSD\_10\_TMP\_WRITE\_PROTECT\_SLICE** MMCSD\_CSD\_20\_TMP\_WRITE\_PROTECT\_SLICE
- #define **MMCSD\_CSD\_10\_PERM\_WRITE\_PROTECT\_SLICE** MMCSD\_CSD\_20\_PERM\_WRITE\_PROTECT\_SLICE
- #define **MMCSD\_CSD\_10\_COPY\_SLICE** MMCSD\_CSD\_20\_COPY\_SLICE
- #define **MMCSD\_CSD\_10\_FILE\_FORMAT\_GRP\_SLICE** MMCSD\_CSD\_20\_FILE\_FORMAT\_GRP\_SLICE
- #define **MMCSD\_CSD\_10\_WRITE\_BL\_PARTIAL\_SLICE** MMCSD\_CSD\_20\_WRITE\_BL\_PARTIAL\_SLICE
- #define **MMCSD\_CSD\_10\_WRITE\_BL\_LEN\_SLICE** MMCSD\_CSD\_20\_WRITE\_BL\_LEN\_SLICE
- #define **MMCSD\_CSD\_10\_R2W\_FACTOR\_SLICE** MMCSD\_CSD\_20\_R2W\_FACTOR\_SLICE
- #define **MMCSD\_CSD\_10\_WP\_GRP\_ENABLE\_SLICE** MMCSD\_CSD\_20\_WP\_GRP\_ENABLE\_SLICE
- #define **MMCSD\_CSD\_10\_WP\_GRP\_SIZE\_SLICE** MMCSD\_CSD\_20\_WP\_GRP\_SIZE\_SLICE
- #define **MMCSD\_CSD\_10\_ERASE\_SECTOR\_SIZE\_SLICE** MMCSD\_CSD\_20\_ERASE\_SECTOR\_SIZE\_SLICE
- #define **MMCSD\_CSD\_10\_ERASE\_BLK\_EN\_SLICE** MMCSD\_CSD\_20\_ERASE\_BLK\_EN\_SLICE
- #define **MMCSD\_CSD\_10\_C\_SIZE\_MULT\_SLICE** 49U,47U
- #define **MMCSD\_CSD\_10\_VDD\_W\_CURR\_MAX\_SLICE** 52U,50U
- #define **MMCSD\_CSD\_10\_VDD\_W\_CURR\_MIN\_SLICE** 55U,53U
- #define **MMCSD\_CSD\_10\_VDD\_R\_CURR\_MAX\_SLICE** 58U,56U
- #define **MMCSD\_CSD\_10\_VDD\_R\_CURR\_MIX\_SLICE** 61U,59U
- #define **MMCSD\_CSD\_10\_C\_SIZE\_SLICE** 73U,62U
- #define **MMCSD\_CSD\_10\_DSR\_IMP\_SLICE** MMCSD\_CSD\_20\_DSR\_IMP\_SLICE
- #define **MMCSD\_CSD\_10\_READ\_BLK\_MISALIGN\_SLICE** MMCSD\_CSD\_20\_READ\_BLK\_MISALIGN\_SLICE
- #define **MMCSD\_CSD\_10\_WRITE\_BLK\_MISALIGN\_SLICE** MMCSD\_CSD\_20\_WRITE\_BLK\_MISALIGN\_SLICE
- #define **MMCSD\_CSD\_10\_READ\_BL\_PARTIAL\_SLICE** MMCSD\_CSD\_20\_READ\_BL\_PARTIAL\_SLICE
- #define **MMCSD\_CSD\_10\_READ\_BL\_LEN\_SLICE** 83U,80U
- #define **MMCSD\_CSD\_10\_CCC\_SLICE** MMCSD\_CSD\_20\_CCC\_SLICE
- #define **MMCSD\_CSD\_10\_TRANS\_SPEED\_SLICE** MMCSD\_CSD\_20\_TRANS\_SPEED\_SLICE
- #define **MMCSD\_CSD\_10\_NSAC\_SLICE** MMCSD\_CSD\_20\_NSAC\_SLICE
- #define **MMCSD\_CSD\_10\_TAAC\_SLICE** MMCSD\_CSD\_20\_TAAC\_SLICE
- #define **MMCSD\_CSD\_10\_CSD\_STRUCTURE\_SLICE** MMCSD\_CSD\_20\_CSD\_STRUCTURE\_SLICE

### CID record offsets

- #define **MMCSD\_CID\_SDC\_CRC\_SLICE** 7U,1U  
*Slice position of values in CID register.*
- #define **MMCSD\_CID\_SDC\_MDT\_M\_SLICE** 11U,8U
- #define **MMCSD\_CID\_SDC\_MDT\_Y\_SLICE** 19U,12U
- #define **MMCSD\_CID\_SDC\_PSN\_SLICE** 55U,24U
- #define **MMCSD\_CID\_SDC\_PRV\_M\_SLICE** 59U,56U
- #define **MMCSD\_CID\_SDC\_PRV\_N\_SLICE** 63U,60U
- #define **MMCSD\_CID\_SDC\_PNM0\_SLICE** 71U,64U
- #define **MMCSD\_CID\_SDC\_PNM1\_SLICE** 79U,72U
- #define **MMCSD\_CID\_SDC\_PNM2\_SLICE** 87U,80U
- #define **MMCSD\_CID\_SDC\_PNM3\_SLICE** 95U,88U
- #define **MMCSD\_CID\_SDC\_PNM4\_SLICE** 103U,96U
- #define **MMCSD\_CID\_SDC\_OID\_SLICE** 119U,104U
- #define **MMCSD\_CID\_SDC\_MID\_SLICE** 127U,120U
- #define **MMCSD\_CID\_MMCCRC\_SLICE** 7U,1U
- #define **MMCSD\_CID\_MMCMDTY\_SLICE** 11U,8U
- #define **MMCSD\_CID\_MMCMDTM\_SLICE** 15U,12U
- #define **MMCSD\_CID\_MMCPSON\_SLICE** 47U,16U
- #define **MMCSD\_CID\_MMCPRM\_SLICE** 51U,48U
- #define **MMCSD\_CID\_MMCPRN\_SLICE** 55U,52U
- #define **MMCSD\_CID\_MMCPNM0\_SLICE** 63U,56U
- #define **MMCSD\_CID\_MMCPNM1\_SLICE** 71U,64U
- #define **MMCSD\_CID\_MMCPNM2\_SLICE** 79U,72U
- #define **MMCSD\_CID\_MMCPNM3\_SLICE** 87U,80U
- #define **MMCSD\_CID\_MMCPNM4\_SLICE** 95U,88U
- #define **MMCSD\_CID\_MMCPNM5\_SLICE** 103U,96U
- #define **MMCSD\_CID\_MMCOID\_SLICE** 119U,104U
- #define **MMCSD\_CID\_MMCMID\_SLICE** 127U,120U

### R1 response utilities

- #define **MMCSD\_R1\_ERROR**(r1) (((r1) & **MMCSD\_R1\_ERROR\_MASK**) != 0U)  
*Evaluates to TRUE if the R1 response contains error flags.*
- #define **MMCSD\_R1\_STS**(r1) (((r1) >> 9U) & 15U)  
*Returns the status field of an R1 response.*
- #define **MMCSD\_R1\_IS\_CARD\_LOCKED**(r1) (((((r1) >> 21U) & 1U) != 0U)  
*Evaluates to TRUE if the R1 response indicates a locked card.*

### Macro Functions

- #define **mmcsdGetCardCapacity**(ip) ((ip)->capacity)  
*Returns the card capacity in blocks.*

### Data Structures

- struct **MMCSDBlockDeviceVMT**  
*MMCSDBlockDevice virtual methods table.*
- struct **MMCSDBlockDevice**  
*MCC/SD block device class.*
- struct **unpacked\_sdc\_cid\_t**

- struct [unpacked\\_mmc\\_cid\\_t](#)  
*Unpacked CID register from SDC.*
- struct [unpacked\\_sdc\\_csd\\_10\\_t](#)  
*Unpacked CSD v1.0 register from SDC.*
- struct [unpacked\\_sdc\\_csd\\_20\\_t](#)  
*Unpacked CSD v2.0 register from SDC.*
- struct [unpacked\\_mmc\\_csd\\_t](#)  
*Unpacked CSD register from MMC.*

## Functions

- uint32\_t [\\_mmcsd\\_get\\_slice](#) (const uint32\_t \*data, uint32\_t end, uint32\_t start)  
*Gets a bit field from a words array.*
- uint32\_t [\\_mmcsd\\_get\\_capacity](#) (const uint32\_t \*csd)  
*Extract card capacity from a CSD.*
- uint32\_t [\\_mmcsd\\_get\\_capacity\\_ext](#) (const uint8\_t \*ext\_csd)  
*Extract MMC card capacity from EXT\_CSD.*
- void [\\_mmcsd\\_unpack\\_sdc\\_cid](#) (const MMCSDBlockDevice \*sdcp, [unpacked\\_sdc\\_cid\\_t](#) \*cidsdc)  
*Unpacks SDC CID array in structure.*
- void [\\_mmcsd\\_unpack\\_mmc\\_cid](#) (const MMCSDBlockDevice \*sdcp, [unpacked\\_mmc\\_cid\\_t](#) \*cidmmc)  
*Unpacks MMC CID array in structure.*
- void [\\_mmcsd\\_unpack\\_csd\\_mmc](#) (const MMCSDBlockDevice \*sdcp, [unpacked\\_mmc\\_csd\\_t](#) \*csdmmc)  
*Unpacks MMC CSD array in structure.*
- void [\\_mmcsd\\_unpack\\_csd\\_v10](#) (const MMCSDBlockDevice \*sdcp, [unpacked\\_sdc\\_csd\\_10\\_t](#) \*csd10)  
*Unpacks SDC CSD v1.0 array in structure.*
- void [\\_mmcsd\\_unpack\\_csd\\_v20](#) (const MMCSDBlockDevice \*sdcp, [unpacked\\_sdc\\_csd\\_20\\_t](#) \*csd20)  
*Unpacks SDC CSD v2.0 array in structure.*

### 7.27.2 Macro Definition Documentation

#### 7.27.2.1 #define MMCSD\_BLOCK\_SIZE 512U

Fixed block size for MMC/SD block devices.

#### 7.27.2.2 #define MMCSD\_R1\_ERROR\_MASK 0xFDFFFE008U

Mask of error bits in R1 responses.

#### 7.27.2.3 #define MMCSD\_CMD8\_PATTERN 0x0000001AAU

Fixed pattern for CMD8.

#### 7.27.2.4 #define MMCSD\_CSD\_MMC\_CSD\_STRUCTURE\_SLICE 127U,126U

Slice position of values in CSD register.

#### 7.27.2.5 #define MMCSD\_CID\_SDC\_CRC\_SLICE 7U,1U

Slice position of values in CID register.

## 7.27.2.6 #define \_mmcsd\_block\_device\_methods \_base\_block\_device\_methods

`MMCSDBlockDevice` specific methods.

## 7.27.2.7 #define \_mmcsd\_block\_device\_data

**Value:**

```
_base_block_device_data
/* Card CID.*/
uint32_t cid[4];
/* Card CSD.*/
uint32_t csd[4];
/* Total number of blocks in card.*/
uint32_t capacity;
```



`MMCSDBlockDevice` specific data.

**Note**

It is empty because `MMCSDBlockDevice` is only an interface without implementation.

## 7.27.2.8 #define MMCSD\_R1\_ERROR( r1 ) (((r1) &amp; MMCSD\_R1\_ERROR\_MASK) != 0U)

Evaluates to TRUE if the R1 response contains error flags.

**Parameters**

in	r1	the r1 response
----	----	-----------------

## 7.27.2.9 #define MMCSD\_R1\_STS( r1 ) (((r1) &gt;&gt; 9U) &amp; 15U)

Returns the status field of an R1 response.

**Parameters**

in	r1	the r1 response
----	----	-----------------

## 7.27.2.10 #define MMCSD\_R1\_IS\_CARD\_LOCKED( r1 ) (((r1) &gt;&gt; 21U) &amp; 1U) != 0U)

Evaluates to TRUE if the R1 response indicates a locked card.

**Parameters**

in	r1	the r1 response
----	----	-----------------

## 7.27.2.11 #define mmcsdGetCardCapacity( ip ) ((ip)-&gt;capacity)

Returns the card capacity in blocks.

**Parameters**

in	ip	pointer to a <code>MMCSDBlockDevice</code> or derived class
----	----	---

**Returns**

The card capacity.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.27.3 Function Documentation****7.27.3.1 uint32\_t \_mmcsd\_get\_slice ( const uint32\_t \* *data*, uint32\_t *end*, uint32\_t *start* )**

Gets a bit field from a words array.

**Note**

The bit zero is the LSb of the first word.

**Parameters**

in	<i>data</i>	pointer to the words array
in	<i>end</i>	bit offset of the last bit of the field, inclusive
in	<i>start</i>	bit offset of the first bit of the field, inclusive

**Returns**

The bits field value, left aligned.

**Function Class:**

Not an API, this function is for internal use only.

**7.27.3.2 uint32\_t \_mmcsd\_get\_capacity ( const uint32\_t \* *csd* )**

Extract card capacity from a CSD.

The capacity is returned as number of available blocks.

**Parameters**

in	<i>csd</i>	the CSD record
----	------------	----------------

**Returns**

The card capacity.

**Return values**

0	CSD format error
---	------------------

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.27.3.3 uint32\_t \_mmcsd\_get\_capacity\_ext ( const uint8\_t \* ext\_csd )

Extract MMC card capacity from EXT\_CSD.

The capacity is returned as number of available blocks.

##### Parameters

in	<i>ext_csd</i>	the extended CSD record
----	----------------	-------------------------

##### Returns

The card capacity.

##### Function Class:

Not an API, this function is for internal use only.

#### 7.27.3.4 void \_mmcsd\_unpack\_sdc\_cid ( const MMCSDBlockDevice \* sdc, unpacked\_sdc\_cid\_t \* cidsdc )

Unpacks SDC CID array in structure.

##### Parameters

in	<i>sdc</i>	pointer to the <a href="#">MMCSDBlockDevice</a> object
out	<i>cidsdc</i>	pointer to the <a href="#">unpacked_sdc_cid_t</a> object

##### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.27.3.5 void \_mmcsd\_unpack\_mmc\_cid ( const MMCSDBlockDevice \* sdc, unpacked\_mmc\_cid\_t \* cidmmc )

Unpacks MMC CID array in structure.

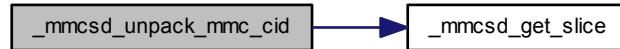
**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">MMCSDBlockDevice</a> object
out	<i>cidmmc</i>	pointer to the <a href="#">unpacked_mmc_cid_t</a> object

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.27.3.6 void \_mmcsd\_unpack\_csd\_mmc ( const MMCSDBlockDevice \* *sdcp*, unpacked\_mmc\_csd\_t \* *csdmmc* )**

Unpacks MMC CSD array in structure.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">MMCSDBlockDevice</a> object
out	<i>csdmmc</i>	pointer to the <a href="#">unpacked_mmc_csd_t</a> object

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.27.3.7 void \_mmcsd\_unpack\_csd\_v10 ( const MMCSDBlockDevice \* *sdcp*, unpacked\_sdc\_csd\_10\_t \* *csd10* )**

Unpacks SDC CSD v1.0 array in structure.

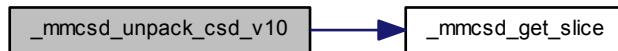
**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">MMCSDBlockDevice</a> object
out	<i>csd10</i>	pointer to the <a href="#">unpacked_sdc_csd_10_t</a> object

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



### 7.27.3.8 void \_mmcsd\_unpack\_csd\_v20( const MMCSDBlockDevice \* sdcp, unpacked\_sdc\_csd\_20\_t \* csd20 )

Unpacks SDC CSD v2.0 array in structure.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">MMCSDBlockDevice</a> object
out	<i>csd20</i>	pointer to the <a href="#">unpacked_sdc_csd_20_t</a> object

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



## 7.28 PAL Driver

I/O Ports Abstraction Layer.

### 7.28.1 Detailed Description

I/O Ports Abstraction Layer.

This module defines an abstract interface for digital I/O ports. Note that most I/O ports functions are just macros. The macros have default software implementations that can be redefined in a PAL Low Level Driver if the target hardware supports special features like, for example, atomic bit set/reset/masking. Please refer to the ports specific documentation for details.

The [PAL Driver](#) driver has the advantage to make the access to the I/O ports platform independent and still be optimized for the specific architectures.

Note that the PAL Low Level Driver may also offer non standard macro and functions in order to support specific features but, of course, the use of such interfaces would not be portable. Such interfaces shall be marked with the architecture name inside the function names.

#### Precondition

In order to use the PAL driver the `HAL_USE_PAL` option must be enabled in [`halconf.h`](#).

### 7.28.2 Implementation Rules

In implementing a PAL Low Level Driver there are some rules/behaviors that should be respected.

#### 7.28.2.1 Writing on input pads

The behavior is not specified but there are implementations better than others, this is the list of possible implementations, preferred options are on top:

1. The written value is not actually output but latched, should the pads be reprogrammed as outputs the value would be in effect.
2. The write operation is ignored.
3. The write operation has side effects, as example disabling/enabling pull up/down resistors or changing the pad direction. This scenario is discouraged, please try to avoid this scenario.

#### 7.28.2.2 Reading from output pads

The behavior is not specified but there are implementations better than others, this is the list of possible implementations, preferred options are on top:

1. The actual pads states are read (not the output latch).
2. The output latch value is read (regardless of the actual pads states).
3. Unspecified, please try to avoid this scenario.

#### 7.28.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

#### 7.28.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

#### 7.28.2.5 Reading or writing on pins associated to other functionalities

The behavior is not specified.

### Macros

- `#define PAL_PORT_BIT(n) ((ioportmask_t)(1U << (n)))`  
*Port bit helper macro.*
- `#define PAL_GROUP_MASK(width) ((ioportmask_t)(1U << (width)) - 1U)`  
*Bits group mask helper.*
- `#define _IOBUS_DATA(name, port, width, offset) {port, PAL_GROUP_MASK(width), offset}`  
*Data part of a static I/O bus initializer.*
- `#define IOBUS_DECL(name, port, width, offset) IOBus name = _IOBUS_DATA(name, port, width, offset)`  
*Static I/O bus initializer.*
- `#define IOPORT1 0`  
*First I/O port identifier.*
- `#define pal_lld_init(config) _pal_lld_init(config)`  
*Low level PAL subsystem initialization.*
- `#define pal_lld_readport(port) 0U`  
*Reads the physical I/O port states.*
- `#define pal_lld_readlatch(port) 0U`  
*Reads the output latch.*
- `#define pal_lld_writeport(port, bits)`  
*Writes a bits mask on a I/O port.*
- `#define pal_lld_setport(port, bits)`  
*Sets a bits mask on a I/O port.*
- `#define pal_lld_clearport(port, bits)`  
*Clears a bits mask on a I/O port.*
- `#define pal_lld_toggleport(port, bits)`  
*Toggles a bits mask on a I/O port.*
- `#define pal_lld_readgroup(port, mask, offset) 0U`  
*Reads a group of bits.*
- `#define pal_lld_writegroup(port, mask, offset, bits)`  
*Writes a group of bits.*
- `#define pal_lld_setgroupmode(port, mask, offset, mode) _pal_lld_setgroupmode(port, mask << offset, mode)`  
*Pads group mode setup.*
- `#define pal_lld_readpad(port, pad) PAL_LOW`  
*Reads a logical state from an I/O pad.*
- `#define pal_lld_writepad(port, pad, bit)`  
*Writes a logical state on an output pad.*
- `#define pal_lld_setpad(port, pad)`  
*Sets a pad logical state to PAL\_HIGH.*
- `#define pal_lld_clearpad(port, pad)`  
*Clears a pad logical state to PAL\_LOW.*
- `#define pal_lld_togglepad(port, pad)`  
*Toggles a pad logical state.*
- `#define pal_lld_setpadmode(port, pad, mode)`  
*Pad mode setup.*

## Pads mode constants

- `#define PAL_MODE_RESET 0U`  
*After reset state.*
- `#define PAL_MODE_UNCONNECTED 1U`  
*Safe state for **unconnected** pads.*
- `#define PAL_MODE_INPUT 2U`  
*Regular input high-Z pad.*
- `#define PAL_MODE_INPUT_PULLUP 3U`  
*Input pad with weak pull up resistor.*
- `#define PAL_MODE_INPUT_PULLDOWN 4U`  
*Input pad with weak pull down resistor.*
- `#define PAL_MODE_INPUT_ANALOG 5U`  
*Analog input mode.*
- `#define PAL_MODE_OUTPUT_PUSH_PULL 6U`  
*Push-pull output pad.*
- `#define PAL_MODE_OUTPUT_OPENDRAIN 7U`  
*Open-drain output pad.*

## Logic level constants

- `#define PAL_LOW 0U`  
*Logical low state.*
- `#define PAL_HIGH 1U`  
*Logical high state.*

## Macro Functions

- `#define palInit(config) pal_lld_init(config)`  
*PAL subsystem initialization.*
- `#define palReadPort(port) ((void)(port), 0U)`  
*Reads the physical I/O port states.*
- `#define palReadLatch(port) ((void)(port), 0U)`  
*Reads the output latch.*
- `#define palWritePort(port, bits) ((void)(port), (void)(bits))`  
*Writes a bits mask on a I/O port.*
- `#define palSetPort(port, bits) palWritePort(port, palReadLatch(port) | (bits))`  
*Sets a bits mask on a I/O port.*
- `#define palClearPort(port, bits) palWritePort(port, palReadLatch(port) & ~(bits))`  
*Clears a bits mask on a I/O port.*
- `#define palTogglePort(port, bits) palWritePort(port, palReadLatch(port) ^ (bits))`  
*Toggles a bits mask on a I/O port.*
- `#define palReadGroup(port, mask, offset) ((palReadPort(port) >> (offset)) & (mask))`  
*Reads a group of bits.*
- `#define palWriteGroup(port, mask, offset, bits)`  
*Writes a group of bits.*
- `#define palSetGroupMode(port, mask, offset, mode)`  
*Pads group mode setup.*
- `#define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1U)`  
*Reads an input pad logic state.*

- `#define palWritePad(port, pad, bit)`  
*Writes a logic state on an output pad.*
- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`  
*Sets a pad logic state to PAL\_HIGH.*
- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`  
*Clears a pad logic state to PAL\_LOW.*
- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`  
*Toggles a pad logic state.*
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)`  
*Pad mode setup.*
- `#define palReadLine(line) palReadPad(PAL_PORT(line), PAL_PAD(line))`  
*Reads an input line logic state.*
- `#define palWriteLine(line, bit) palWritePad(PAL_PORT(line), PAL_PAD(line), bit)`  
*Writes a logic state on an output line.*
- `#define palSetLine(line) palSetPad(PAL_PORT(line), PAL_PAD(line))`  
*Sets a line logic state to PAL\_HIGH.*
- `#define palClearLine(line) palClearPad(PAL_PORT(line), PAL_PAD(line))`  
*Clears a line logic state to PAL\_LOW.*
- `#define palToggleLine(line) palTogglePad(PAL_PORT(line), PAL_PAD(line))`  
*Toggles a line logic state.*
- `#define palSetLineMode(line, mode) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)`  
*Line mode setup.*

## Port related definitions

- `#define PAL_IOPORTS_WIDTH 16U`  
*Width, in bits, of an I/O port.*
- `#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFU)`  
*Whole port mask.*

## Line handling macros

- `#define PAL_LINE(port, pad) ((ioline_t)((uint32_t)(port) | ((uint32_t)(pad)))`  
*Forms a line identifier.*
- `#define PAL_PORT(line) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFFFFF0U))`  
*Decodes a port identifier from a line identifier.*
- `#define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))`  
*Decodes a pad identifier from a line identifier.*
- `#define PAL_NOLINE 0U`  
*Value identifying an invalid line.*

## Typedefs

- `typedef uint32_t ioportmask_t`  
*Digital I/O port sized unsigned type.*
- `typedef uint32_t iomode_t`  
*Digital I/O modes.*
- `typedef uint32_t ioline_t`  
*Type of an I/O line.*
- `typedef uint32_t ioportid_t`  
*Port Identifier.*

## Data Structures

- struct [IOBus](#)  
*I/O bus descriptor.*
- struct [PALConfig](#)  
*Generic I/O ports static initializer.*

## Functions

- [ioportmask\\_t palReadBus \(IOBus \\*bus\)](#)  
*Read from an I/O bus.*
- [void palWriteBus \(IOBus \\*bus, ioportmask\\_t bits\)](#)  
*Write to an I/O bus.*
- [void palSetBusMode \(IOBus \\*bus, iomode\\_t mode\)](#)  
*Programs a bus with the specified mode.*
- [void \\_pal\\_lld\\_init \(const PALConfig \\*config\)](#)  
*STM32 I/O ports configuration.*
- [void \\_pal\\_lld\\_setgroupmode \(ioportid\\_t port, ioportmask\\_t mask, iomode\\_t mode\)](#)  
*Pads mode setup.*

### 7.28.3 Macro Definition Documentation

#### 7.28.3.1 #define PAL\_MODE\_RESET 0U

After reset state.

The state itself is not specified and is architecture dependent, it is guaranteed to be equal to the after-reset state. It is usually an input state.

#### 7.28.3.2 #define PAL\_MODE\_UNCONNECTED 1U

Safe state for **unconnected** pads.

The state itself is not specified and is architecture dependent, it may be mapped on `PAL_MODE_INPUT_PULLUP`, `PAL_MODE_INPUT_PULLDOWN` or `PAL_MODE_OUTPUT_PUSH_PULL` for example.

#### 7.28.3.3 #define PAL\_MODE\_INPUT 2U

Regular input high-Z pad.

#### 7.28.3.4 #define PAL\_MODE\_INPUT\_PULLUP 3U

Input pad with weak pull up resistor.

#### 7.28.3.5 #define PAL\_MODE\_INPUT\_PULLDOWN 4U

Input pad with weak pull down resistor.

#### 7.28.3.6 #define PAL\_MODE\_INPUT\_ANALOG 5U

Analog input mode.

**7.28.3.7 #define PAL\_MODE\_OUTPUT\_PUSH\_PULL 6U**

Push-pull output pad.

**7.28.3.8 #define PAL\_MODE\_OUTPUT\_OPENDRAIN 7U**

Open-drain output pad.

**7.28.3.9 #define PAL\_LOW 0U**

Logical low state.

**7.28.3.10 #define PAL\_HIGH 1U**

Logical high state.

**7.28.3.11 #define PAL\_PORT\_BIT( n ) ((ioportmask\_t)(1U << (n)))**

Port bit helper macro.

This macro calculates the mask of a bit within a port.

**Parameters**

in	n	bit position within the port
----	---	------------------------------

**Returns**

The bit mask.

**7.28.3.12 #define PAL\_GROUP\_MASK( width ) ((ioportmask\_t)(1U << (width)) - 1U)**

Bits group mask helper.

This macro calculates the mask of a bits group.

**Parameters**

in	width	group width
----	-------	-------------

**Returns**

The group mask.

**7.28.3.13 #define \_IOBUS\_DATA( name, port, width, offset ) {port, PAL\_GROUP\_MASK(width), offset}**

Data part of a static I/O bus initializer.

This macro should be used when statically initializing an I/O bus that is part of a bigger structure.

**Parameters**

in	<i>name</i>	name of the <code>IOBus</code> variable
in	<i>port</i>	I/O port descriptor
in	<i>width</i>	bus width in bits
in	<i>offset</i>	bus bit offset within the port

7.28.3.14 `#define IOBUS_DECL( name, port, width, offset ) IOBus name = _IOBUS_DATA(name, port, width, offset)`

Static I/O bus initializer.

#### Parameters

in	<i>name</i>	name of the <code>IOBus</code> variable
in	<i>port</i>	I/O port descriptor
in	<i>width</i>	bus width in bits
in	<i>offset</i>	bus bit offset within the port

7.28.3.15 `#define pallInit( config ) pal_lld_init(config)`

PAL subsystem initialization.

#### Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

#### Parameters

in	<i>config</i>	pointer to an architecture specific configuration structure. This structure is defined in the low level driver header.
----	---------------	--

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.28.3.16 `#define palReadPort( port ) ((void)(port), 0U)`

Reads the physical I/O port states.

#### Note

The default implementation always return zero and computes the parameter eventual side effects.  
The function can be called from any context.

#### Parameters

in	<i>port</i>	port identifier
----	-------------	-----------------

#### Returns

The port logic states.

#### Function Class:

Special function, this function has special requirements see the notes.

**7.28.3.17 #define palReadLatch( *port* ) ((void)(port), 0U)**

Reads the output latch.

The purpose of this function is to read back the latched output value.

**Note**

The default implementation always return zero and computes the parameter eventual side effects.  
The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
----	-------------	-----------------

**Returns**

The latched logic states.

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.18 #define palWritePort( *port*, *bits* ) ((void)(port), (void)(bits))**

Writes a bits mask on a I/O port.

**Note**

The default implementation does nothing except computing the parameters eventual side effects.  
The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be written on the specified port

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.19 #define palSetPort( *port*, *bits* ) palWritePort(*port*, palReadLatch(*port*) | (*bits*))**

Sets a bits mask on a I/O port.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.  
The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.  
The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be ORed on the specified port

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.20 #define palClearPort( *port*, *bits* ) palWritePort(*port*, palReadLatch(*port*) & ~(*bits*))**

Clears a bits mask on a I/O port.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be cleared on the specified port

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.21 #define palTogglePort( *port*, *bits* ) palWritePort(*port*, palReadLatch(*port*) ^ (*bits*))**

Toggles a bits mask on a I/O port.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be XORed on the specified port

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.22 #define palReadGroup( *port*, *mask*, *offset* ) ((palReadPort(*port*) >> (*offset*)) & (*mask*))**

Reads a group of bits.

**Note**

The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask, a logic AND is performed on the input data
in	<i>offset</i>	group bit offset within the port

**Returns**

The group logic states.

**Function Class:**

Special function, this function has special requirements see the notes.

7.28.3.23 #define palWriteGroup( *port*, *mask*, *offset*, *bits* )

**Value:**

```
palWritePort(port, (palReadLatch(port) & ~((mask) << (offset))) |           \
          ((bits) & (mask)) << (offset)))
```

Writes a group of bits.

**Note**

The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask, a logic AND is performed on the output data
in	<i>offset</i>	group bit offset within the port
in	<i>bits</i>	bits to be written. Values exceeding the group width are masked.

**Function Class:**

Special function, this function has special requirements see the notes.

7.28.3.24 #define palSetGroupMode( *port*, *mask*, *offset*, *mode* )

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

**Note**

Programming an unknown or unsupported mode is silently ignored.

The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask

in	<i>offset</i>	group bit offset within the port
in	<i>mode</i>	group mode

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.25 #define palReadPad( *port*, *pad* ) ((palReadPort(*port*) >> (*pad*)) & 1U)**

Reads an input pad logic state.

**Note**

The default implementation not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palReadPort\(\)](#).

The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

**Returns**

The logic state.

**Return values**

<i>PAL_LOW</i>	low logic state.
<i>PAL_HIGH</i>	high logic state.

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.26 #define palWritePad( *port*, *pad*, *bit* )****Value:**

```
palWritePort(port, (palReadLatch(port) & ~PAL\_PORT\_BIT(pad)) |  
\\  
((bit) & 1U) << pad))
```

Writes a logic state on an output pad.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between [osalSysLock\(\)](#) and [osalSysUnlock\(\)](#).

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palReadLatch\(\)](#) and [palWritePort\(\)](#).

The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>bit</i>	logic value, the value must be PAL_LOW or PAL_HIGH

**Function Class:**

Special function, this function has special requirements see the notes.

7.28.3.27 #define palSetPad( *port*, *pad* ) palSetPort(*port*, PAL\_PORT\_BIT(*pad*))

Sets a pad logic state to PAL\_HIGH.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between [osalSysLock\(\)](#) and [osalSysUnlock\(\)](#).

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palSetPort\(\)](#).

The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

**Function Class:**

Special function, this function has special requirements see the notes.

7.28.3.28 #define palClearPad( *port*, *pad* ) palClearPort(*port*, PAL\_PORT\_BIT(*pad*))

Clears a pad logic state to PAL\_LOW.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between [osalSysLock\(\)](#) and [osalSysUnlock\(\)](#).

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palClearPort\(\)](#).

The function can be called from any context.

**Parameters**

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

**Function Class:**

Special function, this function has special requirements see the notes.

---

7.28.3.29 #define palTogglePad( *port*, *pad* ) palTogglePort(*port*, PAL\_PORT\_BIT(*pad*))

Toggles a pad logic state.

#### Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the `palTogglePort()`.

The function can be called from any context.

#### Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

#### Function Class:

Special function, this function has special requirements see the notes.

7.28.3.30 #define palSetPadMode( *port*, *pad*, *mode* ) palSetGroupMode(*port*, PAL\_PORT\_BIT(*pad*), 0U, *mode*)

Pad mode setup.

This function programs a pad with the specified mode.

#### Note

The default implementation not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

Programming an unknown or unsupported mode is silently ignored.

The function can be called from any context.

#### Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>mode</i>	pad mode

#### Function Class:

Special function, this function has special requirements see the notes.

7.28.3.31 #define palReadLine( *line* ) palReadPad(PAL\_PORT(*line*), PAL\_PAD(*line*))

Reads an input line logic state.

#### Note

The function can be called from any context.

**Parameters**

in	<i>line</i>	line identifier
----	-------------	-----------------

**Returns**

The logic state.

**Return values**

<i>PAL_LOW</i>	low logic state.
<i>PAL_HIGH</i>	high logic state.

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.32 #define palWriteLine( *line*, *bit* ) palWritePad(PAL\_PORT(*line*), PAL\_PAD(*line*), *bit*)**

Writes a logic state on an output line.

**Note**

The function can be called from any context.

**Parameters**

in	<i>line</i>	line identifier
in	<i>bit</i>	logic value, the value must be <i>PAL_LOW</i> or <i>PAL_HIGH</i>

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.33 #define palSetLine( *line* ) palSetPad(PAL\_PORT(*line*), PAL\_PAD(*line*))**

Sets a line logic state to *PAL\_HIGH*.

**Note**

The function can be called from any context.

**Parameters**

in	<i>line</i>	line identifier
----	-------------	-----------------

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.34 #define palClearLine( *line* ) palClearPad(PAL\_PORT(*line*), PAL\_PAD(*line*))**

Clears a line logic state to *PAL\_LOW*.

**Note**

The function can be called from any context.

**Parameters**

in	<i>line</i>	line identifier
----	-------------	-----------------

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.35 #define palToggleLine( *line* ) palTogglePad(PAL\_PORT(*line*), PAL\_PAD(*line*))**

Toggles a line logic state.

**Note**

The function can be called from any context.

**Parameters**

in	<i>line</i>	line identifier
----	-------------	-----------------

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.36 #define palSetLineMode( *line*, *mode* ) palSetPadMode(PAL\_PORT(*line*), PAL\_PAD(*line*), *mode*)**

Line mode setup.

**Note**

The function can be called from any context.

**Parameters**

in	<i>line</i>	line identifier
in	<i>mode</i>	pad mode

**Function Class:**

Special function, this function has special requirements see the notes.

**7.28.3.37 #define PAL\_IOPORTS\_WIDTH 16U**

Width, in bits, of an I/O port.

**7.28.3.38 #define PAL\_WHOLE\_PORT ((ioportmask\_t)0xFFFFU)**

Whole port mask.

This macro specifies all the valid bits into a port.

**7.28.3.39 #define PAL\_LINE( *port*, *pad* ) ((ioline\_t)((uint32\_t)(*port*) | ((uint32\_t)(*pad*)))**

Forms a line identifier.

A port/pad pair are encoded into an *ioline\_t* type. The encoding of this type is platform-dependent.

---

```
7.28.3.40 #define PAL_PORT( line ) ((stm32_gpio_t *)(((uint32_t)(line) & 0xFFFFFFFF0U))
```

Decodes a port identifier from a line identifier.

```
7.28.3.41 #define PAL_PAD( line ) ((uint32_t)((uint32_t)(line) & 0x0000000FU))
```

Decodes a pad identifier from a line identifier.

```
7.28.3.42 #define PAL_NOLINE 0U
```

Value identifying an invalid line.

```
7.28.3.43 #define IOPORT1 0
```

First I/O port identifier.

Low level drivers can define multiple ports, it is suggested to use this naming convention.

```
7.28.3.44 #define pal_lld_init( config ) _pal_lld_init(config)
```

Low level PAL subsystem initialization.

**Parameters**

in	<i>config</i>	architecture-dependent ports configuration
----	---------------	--

**Function Class:**

Not an API, this function is for internal use only.

```
7.28.3.45 #define pal_lld_readport( port ) 0U
```

Reads the physical I/O port states.

**Parameters**

in	<i>port</i>	port identifier
----	-------------	-----------------

**Returns**

The port bits.

**Function Class:**

Not an API, this function is for internal use only.

```
7.28.3.46 #define pal_lld_readlatch( port ) 0U
```

Reads the output latch.

The purpose of this function is to read back the latched output value.

**Parameters**

in	<i>port</i>	port identifier
----	-------------	-----------------

**Returns**

The latched logical states.

**Function Class:**

Not an API, this function is for internal use only.

**7.28.3.47 #define pal\_lld\_writeport( *port*, *bits* )****Value:**

```
do {
    (void)port;
    (void)bits;
} while (false)
```

Writes a bits mask on a I/O port.

**Parameters**

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be written on the specified port

**Function Class:**

Not an API, this function is for internal use only.

**7.28.3.48 #define pal\_lld\_setport( *port*, *bits* )****Value:**

```
do {
    (void)port;
    (void)bits;
} while (false)
```

Sets a bits mask on a I/O port.

**Note**

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be ORed on the specified port

**Function Class:**

Not an API, this function is for internal use only.

7.28.3.49 #define pal\_lld\_clearport( *port*, *bits* )

**Value:**

```
do {
    (void)port;
    (void)bits;
} while (false)
```

Clears a bits mask on a I/O port.

**Note**

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be cleared on the specified port

**Function Class:**

Not an API, this function is for internal use only.

7.28.3.50 #define pal\_lld\_toggleport( *port*, *bits* )

**Value:**

```
do {
    (void)port;
    (void)bits;
} while (false)
```

Toggles a bits mask on a I/O port.

**Note**

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be XORed on the specified port

**Function Class:**

Not an API, this function is for internal use only.

7.28.3.51 #define pal\_lld\_readgroup( *port*, *mask*, *offset* ) 0U

Reads a group of bits.

**Note**

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port

**Returns**

The group logical states.

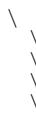
**Function Class:**

Not an API, this function is for internal use only.

7.28.3.52 #define pal\_lld\_writegroup( *port*, *mask*, *offset*, *bits* )

**Value:**

```
do {
    (void)port;
    (void)mask;
    (void)offset;
    (void)bits;
} while (false)
```



Writes a group of bits.

**Note**

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port
in	<i>bits</i>	bits to be written. Values exceeding the group width are masked.

**Function Class:**

Not an API, this function is for internal use only.

7.28.3.53 #define pal\_lld\_setgroupmode( *port*, *mask*, *offset*, *mode* ) \_pal\_lld\_setgroupmode(*port*, *mask* << *offset*,  
  *mode*)

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

**Note**

Programming an unknown or unsupported mode is silently ignored.

**Parameters**

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port
in	<i>mode</i>	group mode

**Function Class:**

Not an API, this function is for internal use only.

**7.28.3.54 #define pal\_lld\_readpad( *port*, *pad* ) PAL\_LOW**

Reads a logical state from an I/O pad.

**Note**

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

**Returns**

The logical state.

**Return values**

<i>PAL_LOW</i>	low logical state.
<i>PAL_HIGH</i>	high logical state.

**Function Class:**

Not an API, this function is for internal use only.

**7.28.3.55 #define pal\_lld\_writepad( *port*, *pad*, *bit* )****Value:**

```
do {
    (void)port;
    (void)pad;
    (void)bit;
} while (false)
```

Writes a logical state on an output pad.

**Note**

This function is not meant to be invoked directly by the application code.

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>bit</i>	logical value, the value must be PAL_LOW or PAL_HIGH

**Function Class:**

Not an API, this function is for internal use only.

7.28.3.56 #define pal\_lld\_setpad( *port*, *pad* )

**Value:**

```
do {
    (void)port;
    (void)pad;
} while (false)
```

Sets a pad logical state to PAL\_HIGH.

**Note**

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

**Function Class:**

Not an API, this function is for internal use only.

7.28.3.57 #define pal\_lld\_clearpad( *port*, *pad* )

**Value:**

```
do {
    (void)port;
    (void)pad;
} while (false)
```

Clears a pad logical state to PAL\_LOW.

**Note**

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

in	<i>port</i>	port identifier
----	-------------	-----------------

in	<i>pad</i>	pad number within the port
----	------------	----------------------------

**Function Class:**

Not an API, this function is for internal use only.

7.28.3.58 #define pal\_lld\_togglepad( *port*, *pad* )

**Value:**

```
do {
    (void)port;
    (void)pad;
} while (false)
```

Toggles a pad logical state.

**Note**

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

**Function Class:**

Not an API, this function is for internal use only.

7.28.3.59 #define pal\_lld\_setpadmode( *port*, *pad*, *mode* )

**Value:**

```
do {
    (void)port;
    (void)pad;
    (void)mode;
} while (false)
```

Pad mode setup.

This function programs a pad with the specified mode.

**Note**

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Programming an unknown or unsupported mode is silently ignored.

**Parameters**

in	<i>port</i>	port identifier
----	-------------	-----------------

in	<i>pad</i>	pad number within the port
in	<i>mode</i>	pad mode

**Function Class:**

Not an API, this function is for internal use only.

**7.28.4 TYPEDOC Documentation****7.28.4.1 `typedef uint32_t ioportmask_t`**

Digital I/O port sized unsigned type.

**7.28.4.2 `typedef uint32_t iomode_t`**

Digital I/O modes.

**7.28.4.3 `typedef uint32_t ioline_t`**

Type of an I/O line.

**7.28.4.4 `typedef uint32_t ioportid_t`**

Port Identifier.

This type can be a scalar or some kind of pointer, do not make any assumption about it, use the provided macros when populating variables of this type.

**7.28.5 Function Documentation****7.28.5.1 `ioportmask_t palReadBus( IOBus * bus )`**

Read from an I/O bus.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The function internally uses the `palReadGroup()` macro. The use of this function is preferred when you value code size, readability and error checking over speed.

The function can be called from any context.

**Parameters**

in	<i>bus</i>	the I/O bus, pointer to a <code>IOBus</code> structure
----	------------	--

**Returns**

The bus logical states.

**Function Class:**

Special function, this function has special requirements see the notes.

### 7.28.5.2 void palWriteBus ( *IOBus \* bus, ioportmask\_t bits* )

Write to an I/O bus.

#### Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The function can be called from any context.

#### Parameters

in	<i>bus</i>	the I/O bus, pointer to a <code>IOBus</code> structure
in	<i>bits</i>	the bits to be written on the I/O bus. Values exceeding the bus width are masked so most significant bits are lost.

#### Function Class:

Special function, this function has special requirements see the notes.

### 7.28.5.3 void palSetBusMode ( *IOBus \* bus, iomode\_t mode* )

Programs a bus with the specified mode.

#### Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The function can be called from any context.

#### Parameters

in	<i>bus</i>	the I/O bus, pointer to a <code>IOBus</code> structure
in	<i>mode</i>	the mode

#### Function Class:

Special function, this function has special requirements see the notes.

### 7.28.5.4 void \_pal\_lld\_init ( *const PALConfig \* config* )

STM32 I/O ports configuration.

Ports A-D(E, F, G, H) clocks enabled.

#### Parameters

in	<i>config</i>	the STM32 ports configuration
----	---------------	-------------------------------

#### Function Class:

Not an API, this function is for internal use only.

#### 7.28.5.5 void \_pal\_lld\_setgroupmode ( *ioportid\_t port*, *ioportmask\_t mask*, *iomode\_t mode* )

Pads mode setup.

This function programs a pads group belonging to the same port with the specified mode.

**Parameters**

in	<i>port</i>	the port identifier
in	<i>mask</i>	the group mask
in	<i>mode</i>	the mode

**Function Class:**

Not an API, this function is for internal use only.

## 7.29 PWM Driver

Generic PWM Driver.

### 7.29.1 Detailed Description

Generic PWM Driver.

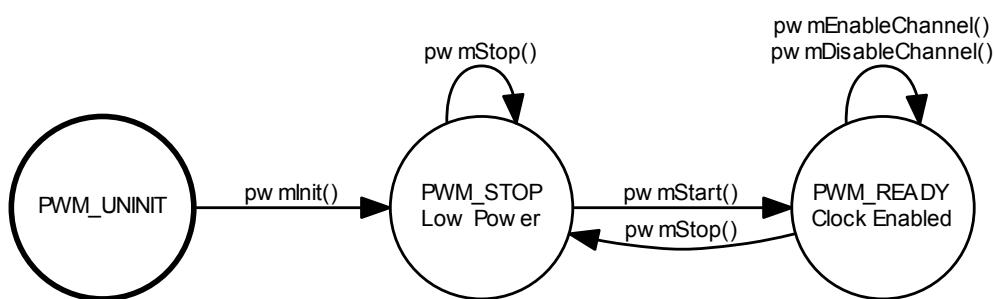
This module implements a generic PWM (Pulse Width Modulation) driver.

#### Precondition

In order to use the PWM driver the `HAL_USE_PWM` option must be enabled in `halconf.h`.

### 7.29.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



### 7.29.3 PWM Operations.

This driver abstracts a generic PWM timer composed of:

- A clock prescaler.
- A main up counter.
- A comparator register that resets the main counter to zero when the limit is reached. An optional callback can be generated when this happens.
- An array of `PWM_CHANNELS` PWM channels, each channel has an output, a comparator and is able to invoke an optional callback when a comparator match with the main counter happens.

A PWM channel output can be in two different states:

- **IDLE**, when the channel is disabled or after a match occurred.
- **ACTIVE**, when the channel is enabled and a match didn't occur yet in the current PWM cycle.

Note that the two states can be associated to both logical zero or one in the `PWMChannelConfig` structure.

## Macros

- `#define PWM_CHANNELS 4`  
*Number of PWM channels per PWM driver.*
- `#define pwm_lld_change_period(pwmp, period)`  
*Changes the period the PWM peripheral.*

## PWM output mode macros

- `#define PWM_OUTPUT_MASK 0x0FU`  
*Standard output modes mask.*
- `#define PWM_OUTPUT_DISABLED 0x00U`  
*Output not driven, callback only.*
- `#define PWM_OUTPUT_ACTIVE_HIGH 0x01U`  
*Positive PWM logic, active is logic level one.*
- `#define PWM_OUTPUT_ACTIVE_LOW 0x02U`  
*Inverse PWM logic, active is logic level zero.*

## PWM duty cycle conversion

- `#define PWM_FRACTION_TO_WIDTH(pwmp, denominator, numerator)`  
*Converts from fraction to pulse width.*
- `#define PWM_DEGREES_TO_WIDTH(pwmp, degrees) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)`  
*Converts from degrees to pulse width.*
- `#define PWM_PERCENTAGE_TO_WIDTH(pwmp, percentage) PWM_FRACTION_TO_WIDTH(pwmp, 10000, percentage)`  
*Converts from percentage to pulse width.*

## Macro Functions

- `#define pwmChangePeriodI(pwmp, value)`  
*Changes the period the PWM peripheral.*
- `#define pwmEnableChannelI(pwmp, channel, width)`  
*Enables a PWM channel.*
- `#define pwmDisableChannelI(pwmp, channel)`  
*Disables a PWM channel.*
- `#define pwmlsChannelEnabledI(pwmp, channel) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)`  
*Returns a PWM channel status.*
- `#define pwmEnablePeriodicNotificationI(pwmp) pwm_lld_enable_periodic_notification(pwmp)`  
*Enables the periodic activation edge notification.*
- `#define pwmDisablePeriodicNotificationI(pwmp) pwm_lld_disable_periodic_notification(pwmp)`  
*Disables the periodic activation edge notification.*
- `#define pwmEnableChannelNotificationI(pwmp, channel) pwm_lld_enable_channel_notification(pwmp, channel)`  
*Enables a channel de-activation edge notification.*
- `#define pwmDisableChannelNotificationI(pwmp, channel) pwm_lld_disable_channel_notification(pwmp, channel)`  
*Disables a channel de-activation edge notification.*

## PLATFORM configuration options

- `#define PLATFORM_PWM_USE_PWM1 FALSE`  
*PWMD1 driver enable switch.*

## Typedefs

- `typedef struct PWMDriver PWMDriver`  
*Type of a structure representing a PWM driver.*
- `typedef void(* pwmcallback_t) (PWMDriver *pwmp)`  
*Type of a PWM notification callback.*
- `typedef uint32_t pwmmodemode_t`  
*Type of a PWM mode.*
- `typedef uint8_t pwmchannel_t`  
*Type of a PWM channel.*
- `typedef uint32_t pwmchnmsk_t`  
*Type of a channels mask.*
- `typedef uint32_t pwcnt_t`  
*Type of a PWM counter.*

## Data Structures

- `struct PWMChannelConfig`  
*Type of a PWM driver channel configuration structure.*
- `struct PWMConfig`  
*Type of a PWM driver configuration structure.*
- `struct PWMDriver`  
*Structure representing a PWM driver.*

## Functions

- `void pwmInit (void)`  
*PWM Driver initialization.*
- `void pwmObjectInit (PWMDriver *pwmp)`  
*Initializes the standard part of a `PWMDriver` structure.*
- `void pwmStart (PWMDriver *pwmp, const PWMConfig *config)`  
*Configures and activates the PWM peripheral.*
- `void pwmStop (PWMDriver *pwmp)`  
*Deactivates the PWM peripheral.*
- `void pwmChangePeriod (PWMDriver *pwmp, pwcnt_t period)`  
*Changes the period the PWM peripheral.*
- `void pwmEnableChannel (PWMDriver *pwmp, pwmchannel_t channel, pwcnt_t width)`  
*Enables a PWM channel.*
- `void pwmDisableChannel (PWMDriver *pwmp, pwmchannel_t channel)`  
*Disables a PWM channel and its notification.*
- `void pwmEnablePeriodicNotification (PWMDriver *pwmp)`  
*Enables the periodic activation edge notification.*
- `void pwmDisablePeriodicNotification (PWMDriver *pwmp)`  
*Disables the periodic activation edge notification.*
- `void pwmEnableChannelNotification (PWMDriver *pwmp, pwmchannel_t channel)`

- `void pwmDisableChannelNotification (PWMDriver *pwmp, pwmchannel_t channel)`  
*Disables a channel de-activation edge notification.*
- `void pwm_lld_init (void)`  
*Low level PWM driver initialization.*
- `void pwm_lld_start (PWMDriver *pwmp)`  
*Configures and activates the PWM peripheral.*
- `void pwm_lld_stop (PWMDriver *pwmp)`  
*Deactivates the PWM peripheral.*
- `void pwm_lld_enable_channel (PWMDriver *pwmp, pwmchannel_t channel, pwcmt_t width)`  
*Enables a PWM channel.*
- `void pwm_lld_disable_channel (PWMDriver *pwmp, pwmchannel_t channel)`  
*Disables a PWM channel and its notification.*
- `void pwm_lld_enable_periodic_notification (PWMDriver *pwmp)`  
*Enables the periodic activation edge notification.*
- `void pwm_lld_disable_periodic_notification (PWMDriver *pwmp)`  
*Disables the periodic activation edge notification.*
- `void pwm_lld_enable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)`  
*Enables a channel de-activation edge notification.*
- `void pwm_lld_disable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)`  
*Disables a channel de-activation edge notification.*

## Enumerations

- enum `pwmstate_t { PWM_UNINIT = 0, PWM_STOP = 1, PWM_READY = 2 }`  
*Driver state machine possible states.*

## Variables

- `PWMDriver PWMD1`  
*PWMD1 driver identifier.*

### 7.29.4 Macro Definition Documentation

#### 7.29.4.1 #define PWM\_OUTPUT\_MASK 0x0FU

Standard output modes mask.

#### 7.29.4.2 #define PWM\_OUTPUT\_DISABLED 0x00U

Output not driven, callback only.

#### 7.29.4.3 #define PWM\_OUTPUT\_ACTIVE\_HIGH 0x01U

Positive PWM logic, active is logic level one.

#### 7.29.4.4 #define PWM\_OUTPUT\_ACTIVE\_LOW 0x02U

Inverse PWM logic, active is logic level zero.

#### 7.29.4.5 #define PWM\_FRACTION\_TO\_WIDTH( *pwmp*, *denominator*, *numerator* )

**Value:**

```
( (pwmcnt_t) (((pwmcnt_t) (pwmp)->period) *
    (pwmcnt_t) (numerator)) / (pwmcnt_t) (denominator))) \
```

Converts from fraction to pulse width.

**Note**

Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

**Parameters**

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>denominator</i>	denominator of the fraction
in	<i>numerator</i>	numerator of the fraction

**Returns**

The pulse width to be passed to `pwmEnableChannel()`.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.29.4.6 #define PWM\_DEGREES\_TO\_WIDTH( *pwmp*, *degrees* ) PWM\_FRACTION\_TO\_WIDTH(*pwmp*, 36000, *degrees*)

Converts from degrees to pulse width.

**Note**

Be careful with rounding errors, this is integer math not magic. You can specify hundredths of degrees but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

**Parameters**

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>degrees</i>	degrees as an integer between 0 and 36000

**Returns**

The pulse width to be passed to `pwmEnableChannel()`.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.29.4.7 #define PWM\_PERCENTAGE\_TO\_WIDTH( *pwmp*, *percentage* ) PWM\_FRACTION\_TO\_WIDTH(*pwmp*, 10000, *percentage*)

Converts from percentage to pulse width.

**Note**

Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>percentage</i>	percentage as an integer between 0 and 10000

**Returns**

The pulse width to be passed to [pwmEnableChannel\(\)](#).

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.29.4.8 #define pwmChangePeriod( *pwmp*, *value* )****Value:**

```
{
    (pwmp) ->period = (value);
    pwm\_lld\_change\_period(pwmp, value);
}
```

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using [pwmStart\(\)](#).

**Precondition**

The PWM unit must have been activated using [pwmStart\(\)](#).

**Postcondition**

The PWM unit period is changed to the new value.

**Note**

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>value</i>	new cycle time in ticks

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.29.4.9 #define pwmEnableChannel( *pwmp*, *channel*, *width* )****Value:**

```
do {
    (pwmp) ->enabled |= ((pwmchnmsk\_t)1U << (pwmchnmsk\_t) (channel));
    pwm\_lld\_enable\_channel(pwmp, channel, width);
} while (false)
```

Enables a PWM channel.

**Precondition**

The PWM unit must have been activated using `pwmStart ()`.

**Postcondition**

The channel is active using the specified configuration.

**Note**

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

**Parameters**

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)
in	<i>width</i>	PWM pulse width as clock pulses number

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.29.4.10 `#define pwmDisableChannel( pwmp, channel )`

**Value:**

```
do {
    (pwmp) ->enabled &= ~((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel));
    \_pwm_lld_disable_channel(pwmp, channel);
} while (false) \_
```

Disables a PWM channel.

**Precondition**

The PWM unit must have been activated using `pwmStart ()`.

**Postcondition**

The channel is disabled and its output line returned to the idle state.

**Note**

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

**Parameters**

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

---

7.29.4.11 #define `pwmIsChannelEnabled( pwmp, channel ) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)`

Returns a PWM channel status.

#### Precondition

The PWM unit must have been activated using `pwmStart()`.

#### Parameters

in	<i>pwmp</i>	pointer to a <code>PWM<a href="#">Driver</a></code> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.29.4.12 #define `pwmEnablePeriodicNotification( pwmp ) pwm_lld_enable_periodic_notification(pwmp)`

Enables the periodic activation edge notification.

#### Precondition

The PWM unit must have been activated using `pwmStart()`.

#### Note

If the notification is already enabled then the call has no effect.

#### Parameters

in	<i>pwmp</i>	pointer to a <code>PWM<a href="#">Driver</a></code> object
----	-------------	--

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.29.4.13 #define `pwmDisablePeriodicNotification( pwmp ) pwm_lld_disable_periodic_notification(pwmp)`

Disables the periodic activation edge notification.

#### Precondition

The PWM unit must have been activated using `pwmStart()`.

#### Note

If the notification is already disabled then the call has no effect.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
----	-------------	---

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.29.4.14 #define pwmEnableChannelNotification( *pwmp*, *channel* ) pwm\_lld\_enable\_channel\_notification(*pwmp*, *channel*)

Enables a channel de-activation edge notification.

**Precondition**

The PWM unit must have been activated using [pwmStart \(\)](#).

The channel must have been activated using [pwmEnableChannel \(\)](#).

**Note**

If the notification is already enabled then the call has no effect.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.29.4.15 #define pwmDisableChannelNotification( *pwmp*, *channel* ) pwm\_lld\_disable\_channel\_notification(*pwmp*, *channel*)

Disables a channel de-activation edge notification.

**Precondition**

The PWM unit must have been activated using [pwmStart \(\)](#).

The channel must have been activated using [pwmEnableChannel \(\)](#).

**Note**

If the notification is already disabled then the call has no effect.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.29.4.16 #define PWM\_CHANNELS 4**

Number of PWM channels per PWM driver.

**7.29.4.17 #define PLATFORM\_PWM\_USE\_PWM1 FALSE**

PWMD1 driver enable switch.

If set to TRUE the support for PWM1 is included.

**Note**

The default is FALSE.

**7.29.4.18 #define pwm\_lld\_change\_period( pwmp, period )**

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using [pwmStart \(\)](#).

**Precondition**

The PWM unit must have been activated using [pwmStart \(\)](#).

**Postcondition**

The PWM unit period is changed to the new value.

**Note**

The function has effect at the next cycle start.

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>period</i>	new cycle time in ticks

**Function Class:**

Not an API, this function is for internal use only.

**7.29.5 Typedef Documentation****7.29.5.1 typedef struct PWMDriver PWMDriver**

Type of a structure representing a PWM driver.

**7.29.5.2 typedef void(\* pwmcallback\_t)(PWMDriver \*pwmp)**

Type of a PWM notification callback.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
----	-------------	---

**7.29.5.3 `typedef uint32_t pwmmode_t`**

Type of a PWM mode.

**7.29.5.4 `typedef uint8_t pwmchannel_t`**

Type of a PWM channel.

**7.29.5.5 `typedef uint32_t pwmchnmsk_t`**

Type of a channels mask.

**7.29.5.6 `typedef uint32_t pwcnt_t`**

Type of a PWM counter.

**7.29.6 Enumeration Type Documentation****7.29.6.1 `enum pwmstate_t`**

Driver state machine possible states.

**Enumerator**

**PWM\_UNINIT** Not initialized.

**PWM\_STOP** Stopped.

**PWM\_READY** Ready.

**7.29.7 Function Documentation****7.29.7.1 `void pwmlInit( void )`**

PWM Driver initialization.

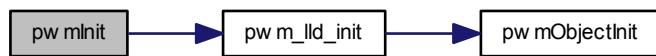
**Note**

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

**7.29.7.2 void pwmObjectInit ( PWMDriver \* pwmp )**

Initializes the standard part of a `PWMDriver` structure.

**Parameters**

<code>out</code>	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
------------------	-------------------	--

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

**7.29.7.3 void pwmStart ( PWMDriver \* pwmp, const PWMConfig \* config )**

Configures and activates the PWM peripheral.

**Note**

Starting a driver that is already in the `PWM_READY` state disables all the active channels.

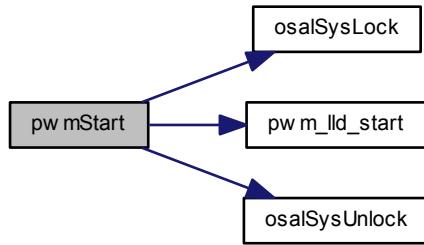
**Parameters**

<code>in</code>	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
<code>in</code>	<code>config</code>	pointer to a <code>PWMConfig</code> object

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.29.7.4 void pwmStop ( PWMDriver \* pwmp )

Deactivates the PWM peripheral.

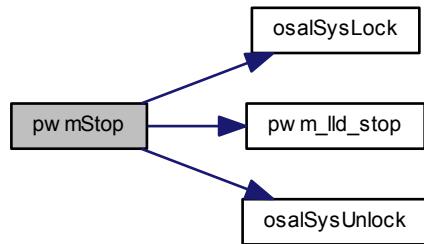
##### Parameters

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
----	-------------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.29.7.5 void pwmChangePeriod ( PWMDriver \* pwmp, pwcnt\_t period )

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using [pwmStart \(\)](#).

**Precondition**

The PWM unit must have been activated using `pwmStart()`.

**Postcondition**

The PWM unit period is changed to the new value.

**Note**

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

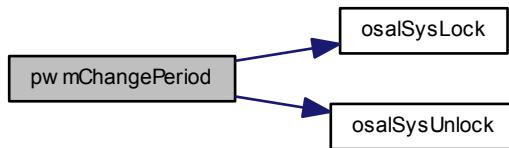
**Parameters**

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>period</i>	new cycle time in ticks

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.29.7.6 void pwmEnableChannel ( PWMDriver \* *pwmp*, pwmchannel\_t *channel*, pwcnt\_t *width* )**

Enables a PWM channel.

**Precondition**

The PWM unit must have been activated using `pwmStart()`.

**Postcondition**

The channel is active using the specified configuration.

**Note**

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

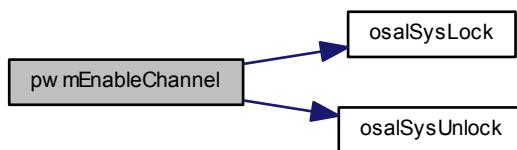
### Parameters

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)
in	<i>width</i>	PWM pulse width as clock pulses number

### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.29.7.7 void pwmDisableChannel ( PWMDriver \* *pwmp*, pwmchannel\_t *channel* )

Disables a PWM channel and its notification.

#### Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

#### Postcondition

The channel is disabled and its output line returned to the idle state.

#### Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

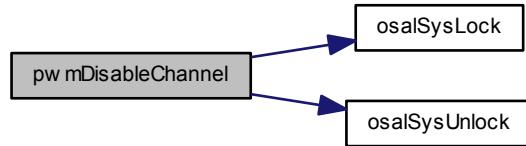
### Parameters

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.29.7.8 void pwmEnablePeriodicNotification ( PWMDriver \* pwmp )

Enables the periodic activation edge notification.

##### Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

##### Note

If the notification is already enabled then the call has no effect.

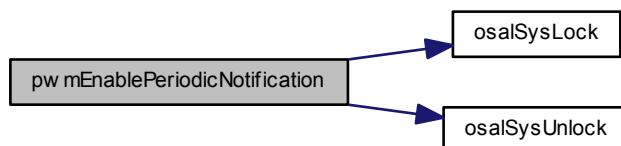
##### Parameters

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
----	-------------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.29.7.9 void pwmDisablePeriodicNotification ( PWMDriver \* pwmp )

Disables the periodic activation edge notification.

**Precondition**

The PWM unit must have been activated using [pwmStart \(\)](#).

**Note**

If the notification is already disabled then the call has no effect.

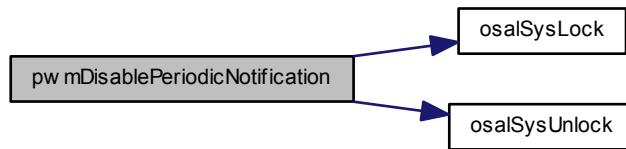
**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.29.7.10 void pwmEnableChannelNotification ( PWMDriver \* *pwmp*, pwmchannel\_t *channel* )**

Enables a channel de-activation edge notification.

**Precondition**

The PWM unit must have been activated using [pwmStart \(\)](#).

The channel must have been activated using [pwmEnableChannel \(\)](#).

**Note**

If the notification is already enabled then the call has no effect.

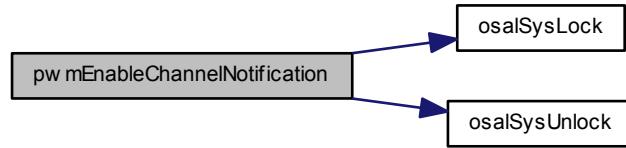
**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.29.7.11 void pwmDisableChannelNotification ( `PWMDriver * pwmp`, `pwmchannel_t channel` )

Disables a channel de-activation edge notification.

##### Precondition

- The PWM unit must have been activated using `pwmStart()`.
- The channel must have been activated using `pwmEnableChannel()`.

##### Note

If the notification is already disabled then the call has no effect.

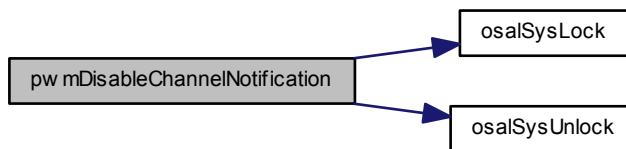
##### Parameters

<code>in</code>	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
<code>in</code>	<code>channel</code>	PWM channel identifier (0...channels-1)

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.29.7.12 void pwm\_lld\_init ( `void` )

Low level PWM driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.29.7.13 void pwm\_lld\_start ( PWMDriver \* pwmp )**

Configures and activates the PWM peripheral.

**Note**

Starting a driver that is already in the `PWM_READY` state disables all the active channels.

**Parameters**

in	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
----	-------------------	--

**Function Class:**

Not an API, this function is for internal use only.

**7.29.7.14 void pwm\_lld\_stop ( PWMDriver \* pwmp )**

Deactivates the PWM peripheral.

**Parameters**

in	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
----	-------------------	--

**Function Class:**

Not an API, this function is for internal use only.

**7.29.7.15 void pwm\_lld\_enable\_channel ( PWMDriver \* pwmp, pwmchannel\_t channel, pwcnt\_t width )**

Enables a PWM channel.

**Precondition**

The PWM unit must have been activated using `pwmStart()`.

**Postcondition**

The channel is active using the specified configuration.

**Note**

The function has effect at the next cycle start.  
Channel notification is not enabled.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)
in	<i>width</i>	PWM pulse width as clock pulses number

**Function Class:**

Not an API, this function is for internal use only.

**7.29.7.16 void pwm\_ll\_disable\_channel ( PWMDriver \* *pwmp*, pwmchannel\_t *channel* )**

Disables a PWM channel and its notification.

**Precondition**

The PWM unit must have been activated using [pwmStart \(\)](#).

**Postcondition**

The channel is disabled and its output line returned to the idle state.

**Note**

The function has effect at the next cycle start.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

**Function Class:**

Not an API, this function is for internal use only.

**7.29.7.17 void pwm\_ll\_enable\_periodic\_notification ( PWMDriver \* *pwmp* )**

Enables the periodic activation edge notification.

**Precondition**

The PWM unit must have been activated using [pwmStart \(\)](#).

**Note**

If the notification is already enabled then the call has no effect.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

### 7.29.7.18 void pwm\_lld\_disable\_periodic\_notification ( **PWMDriver** \* *pwmp* )

Disables the periodic activation edge notification.

#### Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

#### Note

If the notification is already disabled then the call has no effect.

#### Parameters

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
----	-------------	---

#### Function Class:

Not an API, this function is for internal use only.

### 7.29.7.19 void pwm\_lld\_enable\_channel\_notification ( **PWMDriver** \* *pwmp*, **pwmchannel\_t** *channel* )

Enables a channel de-activation edge notification.

#### Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

The channel must have been activated using [pwmEnableChannel \(\)](#).

#### Note

If the notification is already enabled then the call has no effect.

#### Parameters

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

#### Function Class:

Not an API, this function is for internal use only.

### 7.29.7.20 void pwm\_lld\_disable\_channel\_notification ( **PWMDriver** \* *pwmp*, **pwmchannel\_t** *channel* )

Disables a channel de-activation edge notification.

#### Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

The channel must have been activated using [pwmEnableChannel \(\)](#).

#### Note

If the notification is already disabled then the call has no effect.

**Parameters**

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

**Function Class:**

Not an API, this function is for internal use only.

## 7.29.8 Variable Documentation

### 7.29.8.1 PWMDriver PWMD1

PWMD1 driver identifier.

**Note**

The driver PWMD1 allocates the complex timer TIM1 when enabled.

## 7.30 RTC Driver

Generic RTC Driver.

### 7.30.1 Detailed Description

Generic RTC Driver.

This module defines an abstract interface for a Real Time Clock Peripheral.

#### Precondition

In order to use the RTC driver the `HAL_USE_RTC` option must be enabled in `halconf.h`.

#### Macros

- `#define RTC_BASE_YEAR 1980U`  
*Base year of the calendar.*
- `#define _rtc_driver_methods _file_stream_methods`  
*FileStream specific methods.*

#### Date/Time bit masks for FAT format

- `#define RTC_FAT_TIME_SECONDS_MASK 0x0000001FU`
- `#define RTC_FAT_TIME_MINUTES_MASK 0x000007E0U`
- `#define RTC_FAT_TIME_HOURS_MASK 0x0000F800U`
- `#define RTC_FAT_DATE_DAYS_MASK 0x001F0000U`
- `#define RTC_FAT_DATE_MONTHS_MASK 0x01E00000U`
- `#define RTC_FAT_DATE_YEARS_MASK 0xFE000000U`

#### Day of week encoding

- `#define RTC_DAY_CATURDAY 0U`
- `#define RTC_DAY_MONDAY 1U`
- `#define RTC_DAY_TUESDAY 2U`
- `#define RTC_DAY_WEDNESDAY 3U`
- `#define RTC_DAY_THURSDAY 4U`
- `#define RTC_DAY_FRIDAY 5U`
- `#define RTC_DAY_SATURDAY 6U`
- `#define RTC_DAY_SUNDAY 7U`

#### Implementation capabilities

- `#define RTC_SUPPORTS_CALLBACKS TRUE`  
*Callback support int the driver.*
- `#define RTC_ALARMS 2`  
*Number of alarms available.*
- `#define RTC_HAS_STORAGE FALSE`  
*Presence of a local persistent storage.*

## PLATFORM configuration options

- `#define PLATFORM_RTC_USE_RTC1 FALSE`  
*RTCD1 driver enable switch.*

## Typedefs

- `typedef struct RTCDriver RTCDriver`  
*Type of a structure representing an RTC driver.*
- `typedef uint32_t rtcalarm_t`  
*Type of an RTC alarm number.*
- `typedef void(* rtccb_t) (RTCDriver *rtcp, rtcevent_t event)`  
*Type of a generic RTC callback.*

## Data Structures

- `struct RTCDateTime`  
*Type of a structure representing an RTC date/time stamp.*
- `struct RTCAlarm`  
*Type of a structure representing an RTC alarm time stamp.*
- `struct RTCDriverVMT`  
*RTCDriver virtual methods table.*
- `struct RTCDriver`  
*Structure representing an RTC driver.*

## Functions

- `void rtcInit (void)`  
*RTC Driver initialization.*
- `void rtcObjectInit (RTCDriver *rtcp)`  
*Initializes a generic RTC driver object.*
- `void rtcSetTime (RTCDriver *rtcp, const RTCDateTime *timespec)`  
*Set current time.*
- `void rtcGetTime (RTCDriver *rtcp, RTCDateTime *timespec)`  
*Get current time.*
- `void rtcSetAlarm (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAlarm *alarmspec)`  
*Set alarm time.*
- `void rtcGetAlarm (RTCDriver *rtcp, rtcalarm_t alarm, RTCAlarm *alarmspec)`  
*Get current alarm.*
- `void rtcSetCallback (RTCDriver *rtcp, rtccb_t callback)`  
*Enables or disables RTC callbacks.*
- `void rtcConvertDateTimeToStructTm (const RTCDateTime *timespec, struct tm *timp, uint32_t *tv_msec)`  
*Convert RTCDateTime to broken-down time structure.*
- `void rtcConvertStructTmToDateTm (const struct tm *timp, uint32_t tv_msec, RTCDateTime *timespec)`  
*Convert broken-down time structure to RTCDateTime.*
- `uint32_t rtcConvertDateTimeToFAT (const RTCDateTime *timespec)`  
*Get current time in format suitable for usage in FAT file system.*
- `void rtc_lld_init (void)`  
*Enable access to registers.*
- `void rtc_lld_set_time (RTCDriver *rtcp, const RTCDateTime *timespec)`

- Set current time.
  - void `rtc_ll_get_time (RTCDriver *rtcp, RTCDateTime *timespec)`
- Get current time.
  - void `rtc_ll_set_alarm (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAlarm *alarmspec)`
- Set alarm time.
  - void `rtc_ll_get_alarm (RTCDriver *rtcp, rtcalarm_t alarm, RTCAlarm *alarmspec)`
- Get alarm time.

## Enumerations

- enum `rtcevent_t`

*Type of an RTC event.*

## Variables

- RTCDriver `RTCD1`

*RTC driver identifier.*

### 7.30.2 Macro Definition Documentation

#### 7.30.2.1 `#define RTC_BASE_YEAR 1980U`

Base year of the calendar.

#### 7.30.2.2 `#define RTC_SUPPORTS_CALLBACKS TRUE`

Callback support int the driver.

#### 7.30.2.3 `#define RTC_ALARMS 2`

Number of alarms available.

#### 7.30.2.4 `#define RTC_HAS_STORAGE FALSE`

Presence of a local persistent storage.

#### 7.30.2.5 `#define PLATFORM_RTC_USE_RTC1 FALSE`

RTCD1 driver enable switch.

If set to TRUE the support for RTC1 is included.

#### Note

The default is FALSE.

#### 7.30.2.6 `#define _rtc_driver_methods _file_stream_methods`

`FileStream` specific methods.

### 7.30.3 Typedef Documentation

#### 7.30.3.1 `typedef struct RTCDriver RTCDriver`

Type of a structure representing an RTC driver.

#### 7.30.3.2 `typedef uint32_t rtcalarm_t`

Type of an RTC alarm number.

#### 7.30.3.3 `typedef void(* rtccb_t) (RTCDriver *rtcp, rtcevent_t event)`

Type of a generic RTC callback.

### 7.30.4 Enumeration Type Documentation

#### 7.30.4.1 `enum rtcevent_t`

Type of an RTC event.

### 7.30.5 Function Documentation

#### 7.30.5.1 `void rtcInit( void )`

RTC Driver initialization.

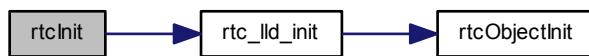
##### Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.30.5.2 `void rtcObjectInit( RTCDriver * rtcp )`

Initializes a generic RTC driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.

**Parameters**

out	<i>rtcp</i>	pointer to RTC driver structure
-----	-------------	---------------------------------

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

**7.30.5.3 void rtcSetTime ( RTCDriver \* *rtcp*, const RTCDateTime \* *timespec* )**

Set current time.

**Note**

This function can be called from any context but limitations could be imposed by the low level implementation.

It is guaranteed that the function can be called from thread context.

The function can be reentrant or not reentrant depending on the low level implementation.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>timespec</i>	pointer to a <a href="#">RTCDateTime</a> structure

**Function Class:**

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

**7.30.5.4 void rtcGetTime ( RTCDriver \* *rtcp*, RTCDateTime \* *timespec* )**

Get current time.

**Note**

This function can be called from any context but limitations could be imposed by the low level implementation.

It is guaranteed that the function can be called from thread context.

The function can be reentrant or not reentrant depending on the low level implementation.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
----	-------------	---------------------------------

out	<i>timespec</i>	pointer to a <a href="#">RTCDateTime</a> structure
-----	-----------------	--

**Function Class:**

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

**7.30.5.5 void rtcSetAlarm ( RTCDriver \* *rtcp*, rtcalarm\_t *alarm*, const RTCAlarm \* *alarmspec* )**

Set alarm time.

**Note**

This function can be called from any context but limitations could be imposed by the low level implementation.

It is guaranteed that the function can be called from thread context.

The function can be reentrant or not reentrant depending on the low level implementation.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
in	<i>alarmspec</i>	pointer to a <a href="#">RTCAlarm</a> structure or NULL

**Function Class:**

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

**7.30.5.6 void rtcGetAlarm ( RTCDriver \* *rtcp*, rtcalarm\_t *alarm*, RTCAlarm \* *alarmspec* )**

Get current alarm.

**Note**

If an alarm has not been set then the returned alarm specification is not meaningful.  
 This function can be called from any context but limitations could be imposed by the low level implementation.  
 It is guaranteed that the function can be called from thread context.  
 The function can be reentrant or not reentrant depending on the low level implementation.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
out	<i>alarmspec</i>	pointer to a <a href="#">RTCAlarm</a> structure

**Function Class:**

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

**7.30.5.7 void rtcSetCallback ( RTCDriver \* *rtcp*, rtccb\_t *callback* )**

Enables or disables RTC callbacks.

This function enables or disables the callback, use a `NULL` pointer in order to disable it.

**Note**

This function can be called from any context but limitations could be imposed by the low level implementation.  
 It is guaranteed that the function can be called from thread context.  
 The function can be reentrant or not reentrant depending on the low level implementation.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>callback</i>	callback function pointer or <code>NULL</code>

**Function Class:**

Special function, this function has special requirements see the notes.

**7.30.5.8 void rtcConvertDateTimeToStructTm ( const RTCDateTime \* *timespec*, struct tm \* *tmp*, uint32\_t \* *tv\_msec* )**

Convert [RTCDateTime](#) to broken-down time structure.

**Parameters**

in	<i>timespec</i>	pointer to a <a href="#">RTCDateTime</a> structure
out	<i>tmp</i>	pointer to a broken-down time structure
out	<i>tv_msec</i>	pointer to milliseconds value or NULL

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.30.5.9 void rtcConvertStructTmToDateTIme ( const struct tm \* *tmp*, uint32\_t *tv\_msec*, RTCDateTime \* *timespec* )**

Convert broken-down time structure to [RTCDateTime](#).

**Parameters**

in	<i>tmp</i>	pointer to a broken-down time structure
in	<i>tv_msec</i>	milliseconds value
out	<i>timespec</i>	pointer to a <a href="#">RTCDateTime</a> structure

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.30.5.10 uint32\_t rtcConvertDateTimeToFAT ( const RTCDateTime \* *timespec* )**

Get current time in format suitable for usage in FAT file system.

**Note**

The information about day of week and DST is lost in DOS format, the second field loses its least significant bit.

**Parameters**

out	<i>timespec</i>	pointer to a <a href="#">RTCDateTime</a> structure
-----	-----------------	--

**Returns**

FAT date/time value.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.30.5.11 void rtc\_lld\_init ( void )**

Enable access to registers.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



### 7.30.5.12 void rtc\_lld\_set\_time ( RTCDriver \* *rtcp*, const RTCDateTime \* *timespec* )

Set current time.

#### Note

Fractional part will be silently ignored. There is no possibility to set it on PLATFORM platform.  
The function can be called from any context.

#### Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>timespec</i>	pointer to a <a href="#">RTCDateTime</a> structure

#### Function Class:

Not an API, this function is for internal use only.

### 7.30.5.13 void rtc\_lld\_get\_time ( RTCDriver \* *rtcp*, RTCDateTime \* *timespec* )

Get current time.

#### Note

The function can be called from any context.

#### Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
out	<i>timespec</i>	pointer to a <a href="#">RTCDateTime</a> structure

#### Function Class:

Not an API, this function is for internal use only.

### 7.30.5.14 void rtc\_lld\_set\_alarm ( RTCDriver \* *rtcp*, rtcalarm\_t *alarm*, const RTCAlarm \* *alarmspec* )

Set alarm time.

#### Note

Default value after BKP domain reset for both comparators is 0.  
Function does not performs any checks of alarm time validity.  
The function can be called from any context.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure.
in	<i>alarm</i>	alarm identifier. Can be 1 or 2.
in	<i>alarmspec</i>	pointer to a <a href="#">RTCAalarm</a> structure.

**Function Class:**

Not an API, this function is for internal use only.

**7.30.5.15 void rtc\_lld\_get\_alarm ( RTCDriver \* *rtcp*, rtcalarm\_t *alarm*, RTCAalarm \* *alarmspec* )**

Get alarm time.

**Note**

The function can be called from any context.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
out	<i>alarmspec</i>	pointer to a <a href="#">RTCAalarm</a> structure

**Function Class:**

Not an API, this function is for internal use only.

**7.30.6 Variable Documentation****7.30.6.1 RTCDriver RTCD1**

RTC driver identifier.

## 7.31 SDC Driver

Generic SD Card Driver.

### 7.31.1 Detailed Description

Generic SD Card Driver.

This module implements a generic SDC (Secure Digital Card) driver.

#### Precondition

In order to use the SDC driver the `HAL_USE_SDC` option must be enabled in `halconf.h`.

### 7.31.2 Driver State Machine

This driver implements a state machine internally, see the [Abstract I/O Block Device](#) module documentation for details.

### 7.31.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

#### Macros

- `#define _sdc_driver_methods _mmcsd_block_device_methods`  
*SDC driver specific methods.*

#### SD card types

- `#define SDC_MODE_CARDTYPE_MASK 0xFU`
- `#define SDC_MODE_CARDTYPE_SDV11 0U`
- `#define SDC_MODE_CARDTYPE_SDV20 1U`
- `#define SDC_MODE_CARDTYPE_MMC 2U`
- `#define SDC_MODE_HIGH_CAPACITY 0x10U`

#### SDC bus error conditions

- `#define SDC_NO_ERROR 0U`
- `#define SDC_CMD_CRC_ERROR 1U`
- `#define SDC_DATA_CRC_ERROR 2U`
- `#define SDC_DATA_TIMEOUT 4U`
- `#define SDC_COMMAND_TIMEOUT 8U`
- `#define SDC_TX_UNDERRUN 16U`
- `#define SDC_RX_OVERRUN 32U`
- `#define SDC_STARTBIT_ERROR 64U`
- `#define SDC_OVERFLOW_ERROR 128U`
- `#define SDC_UNHANDLED_ERROR 0xFFFFFFFFU`

## SDC configuration options

- `#define SDC_INIT_RETRY 100`  
*Number of initialization attempts before rejecting the card.*
- `#define SDC_MMC_SUPPORT FALSE`  
*Include support for MMC cards.*
- `#define SDC_NICE_WAITING TRUE`  
*Delays insertions.*

## Macro Functions

- `#define sdclsCardInserted(sdc) (sdc_lld_is_card_inserted(sdc))`  
*Returns the card insertion status.*
- `#define sdclsWriteProtected(sdc) (sdc_lld_is_write_protected(sdc))`  
*Returns the write protect status.*

## PLATFORM configuration options

- `#define PLATFORM_SDC_USE_SDC1 FALSE`  
*PWMD1 driver enable switch.*

## Typedefs

- `typedef uint32_t sdcmode_t`  
*Type of card flags.*
- `typedef uint32_t sdcflags_t`  
*SDC Driver condition flags type.*
- `typedef struct SDCDriver SDCDriver`  
*Type of a structure representing an SDC driver.*

## Data Structures

- `struct SDCCConfig`  
*Driver configuration structure.*
- `struct SDCDriverVMT`  
*SDCDriver virtual methods table.*
- `struct SDCDriver`  
*Structure representing an SDC driver.*

## Functions

- `static bool mode_detect (SDCDriver *sdc)`  
*Detects card mode.*
- `static bool mmc_init (SDCDriver *sdc)`  
*Init procedure for MMC.*
- `static bool sdc_init (SDCDriver *sdc)`  
*Init procedure for SDC.*
- `static uint32_t mmc_cmd6_construct (mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set)`  
*Constructs CMD6 argument for MMC.*
- `static uint32_t sdc_cmd6_construct (sd_switch_t mode, sd_switch_function_t function, uint32_t value)`

- static uint16\_t **sdc\_cmd6\_extract\_info** (sd\_switch\_function\_t function, const uint8\_t \*buf)
 

*Extracts information from CMD6 answer.*
- static bool **sdc\_cmd6\_check\_status** (sd\_switch\_function\_t function, const uint8\_t \*buf)
 

*Checks status after switching using CMD6.*
- static bool **sdc\_detect\_bus\_clk** (SDCDriver \*sdcp, sdcbusclk\_t \*clk)
 

*Reads supported bus clock and switch SDC to appropriate mode.*
- static bool **mmc\_detect\_bus\_clk** (SDCDriver \*sdcp, sdcbusclk\_t \*clk)
 

*Reads supported bus clock and switch MMC to appropriate mode.*
- static bool **detect\_bus\_clk** (SDCDriver \*sdcp, sdcbusclk\_t \*clk)
 

*Reads supported bus clock and switch card to appropriate mode.*
- static bool **sdc\_set\_bus\_width** (SDCDriver \*sdcp)
 

*Sets bus width for SDC.*
- static bool **mmc\_set\_bus\_width** (SDCDriver \*sdcp)
 

*Sets bus width for MMC.*
- bool **\_sdc\_wait\_for\_transfer\_state** (SDCDriver \*sdcp)
 

*Wait for the card to complete pending operations.*
- void **sdcInit** (void)
 

*SDC Driver initialization.*
- void **sdcObjectInit** (SDCDriver \*sdcp)
 

*Initializes the standard part of a *SDCDriver* structure.*
- void **sdcStart** (SDCDriver \*sdcp, const SDCCConfig \*config)
 

*Configures and activates the SDC peripheral.*
- void **sdcStop** (SDCDriver \*sdcp)
 

*Deactivates the SDC peripheral.*
- bool **sdcConnect** (SDCDriver \*sdcp)
 

*Performs the initialization procedure on the inserted card.*
- bool **sdcDisconnect** (SDCDriver \*sdcp)
 

*Brings the driver in a state safe for card removal.*
- bool **sdcRead** (SDCDriver \*sdcp, uint32\_t startblk, uint8\_t \*buf, uint32\_t n)
 

*Reads one or more blocks.*
- bool **sdcWrite** (SDCDriver \*sdcp, uint32\_t startblk, const uint8\_t \*buf, uint32\_t n)
 

*Writes one or more blocks.*
- **sdcflags\_t sdcGetAndClearErrors** (SDCDriver \*sdcp)
 

*Returns the errors mask associated to the previous operation.*
- bool **sdcSync** (SDCDriver \*sdcp)
 

*Waits for card idle condition.*
- bool **sdcGetInfo** (SDCDriver \*sdcp, BlockDeviceInfo \*bdip)
 

*Returns the media info.*
- bool **sdcErase** (SDCDriver \*sdcp, uint32\_t startblk, uint32\_t endblk)
 

*Erases the supplied blocks.*
- void **sdc\_lld\_init** (void)
 

*Low level SDC driver initialization.*
- void **sdc\_lld\_start** (SDCDriver \*sdcp)
 

*Configures and activates the SDC peripheral.*
- void **sdc\_lld\_stop** (SDCDriver \*sdcp)
 

*Deactivates the SDC peripheral.*
- void **sdc\_lld\_start\_clk** (SDCDriver \*sdcp)
 

*Starts the SDIO clock and sets it to init mode (400kHz or less).*
- void **sdc\_lld\_set\_data\_clk** (SDCDriver \*sdcp, sdcbusclk\_t clk)
 

*Sets the SDIO clock to data mode (25MHz or less).*

- void `sdc_lld_stop_clk (SDCDriver *sdcp)`  
*Stops the SDIO clock.*
- void `sdc_lld_set_bus_mode (SDCDriver *sdcp, sdcbusmode_t mode)`  
*Switches the bus to 4 bits mode.*
- void `sdc_lld_send_cmd_none (SDCDriver *sdcp, uint8_t cmd, uint32_t arg)`  
*Sends an SDIO command with no response expected.*
- bool `sdc_lld_send_cmd_short (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`  
*Sends an SDIO command with a short response expected.*
- bool `sdc_lld_send_cmd_short_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`  
*Sends an SDIO command with a short response expected and CRC.*
- bool `sdc_lld_send_cmd_long_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`  
*Sends an SDIO command with a long response expected and CRC.*
- bool `sdc_lld_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`  
*Reads one or more blocks.*
- bool `sdc_lld_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`  
*Writes one or more blocks.*
- bool `sdc_lld_sync (SDCDriver *sdcp)`  
*Waits for card idle condition.*

## Enumerations

- enum `mmc_switch_t`  
*MMC switch mode.*
- enum `sd_switch_t`  
*SDC switch mode.*
- enum `sd_switch_function_t`  
*SDC switch function.*
- enum `sdcbusmode_t`  
*Type of SDIO bus mode.*
- enum `sdcbusclk_t`  
*Max supported clock.*

## Variables

- static const struct SDCDriverVMT `sdc_vmt`  
*Virtual methods table.*
- SDCDriver `SDCD1`  
*SDCD1 driver identifier.*

### 7.31.4 Macro Definition Documentation

#### 7.31.4.1 #define SDC\_INIT\_RETRY 100

Number of initialization attempts before rejecting the card.

#### Note

Attempts are performed at 10mS intervals.

#### 7.31.4.2 #define SDC\_MMC\_SUPPORT FALSE

Include support for MMC cards.

##### Note

MMC support is not yet implemented so this option must be kept at FALSE.

#### 7.31.4.3 #define SDC\_NICE\_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however.

#### 7.31.4.4 #define sdclIsCardInserted( *sdcp* ) (sdc\_lld\_is\_card\_inserted(*sdcp*))

Returns the card insertion status.

##### Note

This macro wraps a low level function named `sdc_lld_is_card_inserted()`, this function must be provided by the application because it is not part of the SDC driver.

##### Parameters

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
----	-------------	--

##### Returns

The card state.

##### Return values

<i>FALSE</i>	card not inserted.
<i>TRUE</i>	card inserted.

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.31.4.5 #define sdclIsWriteProtected( *sdcp* ) (sdc\_lld\_is\_write\_protected(*sdcp*))

Returns the write protect status.

##### Note

This macro wraps a low level function named `sdc_lld_is_write_protected()`, this function must be provided by the application because it is not part of the SDC driver.

##### Parameters

in	<code>sdcp</code>	pointer to the <code>SDCDriver</code> object
----	-------------------	--

#### Returns

The card state.

#### Return values

<i>FALSE</i>	not write protected.
<i>TRUE</i>	write protected.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.31.4.6 `#define PLATFORM_SDC_USE_SDC1 FALSE`

PWMD1 driver enable switch.

If set to `TRUE` the support for PWM1 is included.

#### Note

The default is `FALSE`.

#### 7.31.4.7 `#define _sdc_driver_methods _mmcsd_block_device_methods`

`SDCDriver` specific methods.

### 7.31.5 Typedef Documentation

#### 7.31.5.1 `typedef uint32_t sdcmode_t`

Type of card flags.

#### 7.31.5.2 `typedef uint32_t sdcflags_t`

SDC Driver condition flags type.

#### 7.31.5.3 `typedef struct SDCDriver SDCDriver`

Type of a structure representing an SDC driver.

### 7.31.6 Enumeration Type Documentation

#### 7.31.6.1 `enum mmc_switch_t`

MMC switch mode.

#### 7.31.6.2 `enum sd_switch_t`

SDC switch mode.

### 7.31.6.3 enum sd\_switch\_function\_t

SDC switch function.

### 7.31.6.4 enum sdcbusmode\_t

Type of SDIO bus mode.

### 7.31.6.5 enum sdcbusclk\_t

Max supported clock.

## 7.31.7 Function Documentation

### 7.31.7.1 static bool mode\_detect ( SDCDriver \* *sdcp* ) [static]

Detects card mode.

#### Parameters

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
----	-------------	---

#### Returns

The operation status.

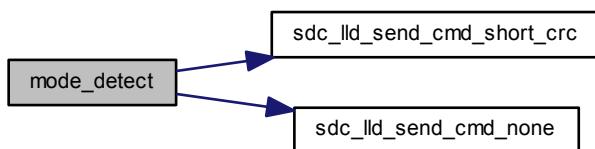
#### Return values

<a href="#">HAL_SUCCESS</a>	operation succeeded.
<a href="#">HAL_FAILED</a>	operation failed.

#### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



### 7.31.7.2 static bool mmc\_init ( SDCDriver \* *sdcp* ) [static]

Init procedure for MMC.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
----	-------------	---

**Returns**

The operation status.

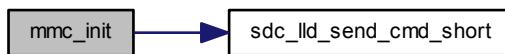
**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.31.7.3 static bool sdc\_init( [SDCDriver](#) \* *sdcp* ) [static]**

Init procedure for SDC.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
----	-------------	---

**Returns**

The operation status.

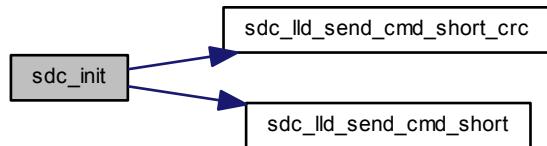
**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



**7.31.7.4 static uint32\_t mmc\_cmd6\_construct ( mmc\_switch\_t access, uint32\_t idx, uint32\_t value, uint32\_t cmd\_set ) [static]**

Constructs CMD6 argument for MMC.

**Parameters**

in	<i>access</i>	EXT_CSD access mode
in	<i>idx</i>	EXT_CSD byte number
in	<i>value</i>	value to be written in target field
in	<i>cmd_set</i>	switch current command set

**Returns**

CMD6 argument.

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.5 static uint32\_t sdc\_cmd6\_construct ( sd\_switch\_t mode, sd\_switch\_function\_t function, uint32\_t value ) [static]**

Constructs CMD6 argument for SDC.

**Parameters**

in	<i>mode</i>	switch/test mode
in	<i>function</i>	function number to be switched
in	<i>value</i>	value to be written in target function

**Returns**

CMD6 argument.

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.6 static uint16\_t sdc\_cmd6\_extract\_info ( *sd\_switch\_function\_t function*, *const uint8\_t \* buf* ) [static]**

Extracts information from CMD6 answer.

**Parameters**

in	<i>function</i>	function number to be switched
in	<i>buf</i>	buffer with answer

**Returns**

extracted answer.

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.7 static bool sdc\_cmd6\_check\_status ( *sd\_switch\_function\_t function*, *const uint8\_t \* buf* ) [static]**

Checks status after switching using CMD6.

**Parameters**

in	<i>function</i>	function number to be switched
in	<i>buf</i>	buffer with answer

**Returns**

The operation status.

**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.8 static bool sdc\_detect\_bus\_clk ( *SDCDriver \* sdcp*, *sdcbusclk\_t \* clk* ) [static]**

Reads supported bus clock and switch SDC to appropriate mode.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
out	<i>clk</i>	pointer to clock enum

**Returns**

The operation status.

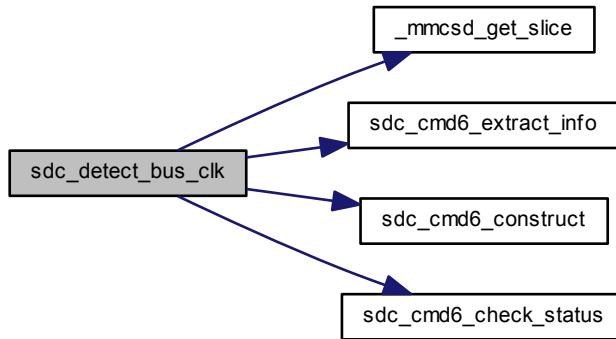
**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.31.7.9 static bool mmc\_detect\_bus\_clk( [SDCDriver](#) \* *sdcp*, [sdcbusclk\\_t](#) \* *clk* ) [static]**

Reads supported bus clock and switch MMC to appropriate mode.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
out	<i>clk</i>	pointer to clock enum

**Returns**

The operation status.

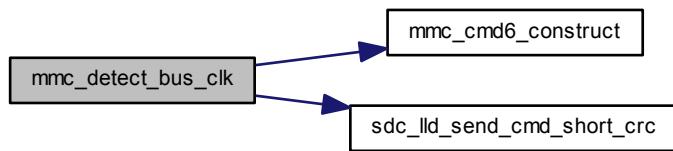
**Return values**

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



### 7.31.7.10 static bool detect\_bus\_clk ( `SDCDriver` \* `sdcpc`, `sdcbusclk_t` \* `clk` ) [static]

Reads supported bus clock and switch card to appropriate mode.

**Parameters**

in	<code>sdcpc</code>	pointer to the <code>SDCDriver</code> object
out	<code>clk</code>	pointer to clock enum

**Returns**

The operation status.

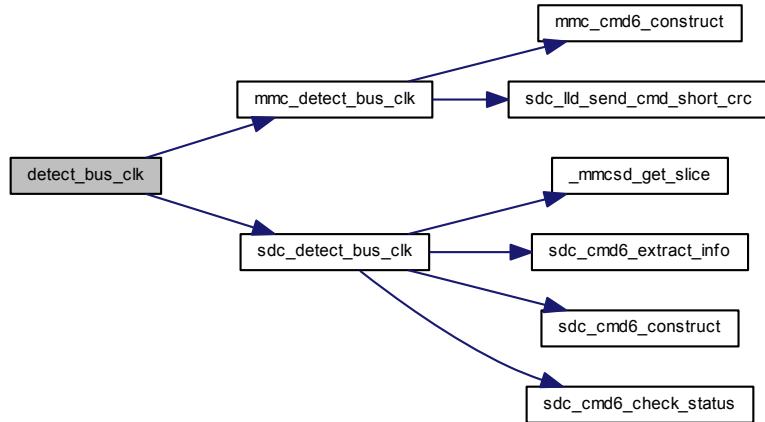
**Return values**

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



### 7.31.7.11 static bool sdc\_set\_bus\_width ( SDCDriver \* sdc ) [static]

Sets bus width for SDC.

#### Parameters

in	<i>sdc</i>	pointer to the <a href="#">SDCDriver</a> object
----	------------	---

#### Returns

The operation status.

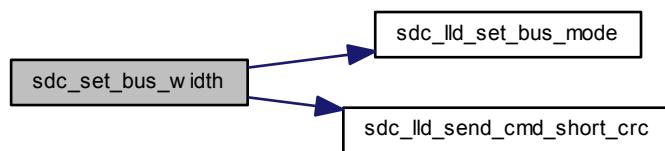
#### Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

#### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



**7.31.7.12 static bool mmc\_set\_bus\_width( **SDCDriver** \* *sdcp* ) [static]**

Sets bus width for MMC.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
----	-------------	---

**Returns**

The operation status.

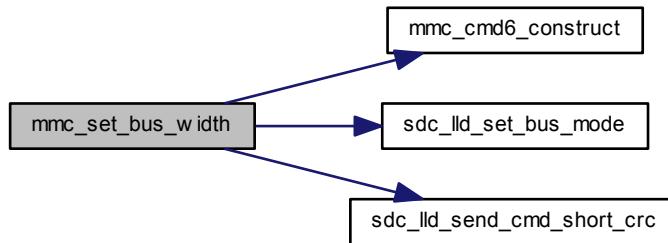
**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.31.7.13 bool \_sdc\_wait\_for\_transfer\_state ( [SDCDriver](#) \* *sdc* )**

Wait for the card to complete pending operations.

**Parameters**

in	<i>sdc</i>	pointer to the <a href="#">SDCDriver</a> object
----	------------	---

**Returns**

The operation status.

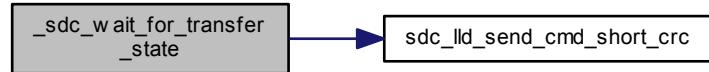
**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.31.7.14 void sdcInit( void )

SDC Driver initialization.

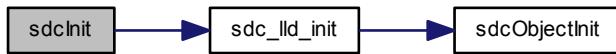
##### Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.31.7.15 void sdcObjectInit( SDCDriver \* sdcp )

Initializes the standard part of a [SDCDriver](#) structure.

##### Parameters

out	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
-----	-------------	---

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

#### 7.31.7.16 void sdcStart( SDCDriver \* sdcp, const SDCCConfig \* config )

Configures and activates the SDC peripheral.

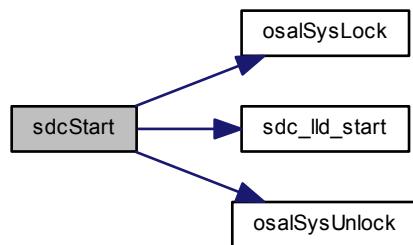
**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
in	<i>config</i>	pointer to the <a href="#">SDCConfig</a> object, can be NULL if the driver supports a default configuration or requires no configuration

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.31.7.17 void sdcStop ( SDCDriver \* *sdcp* )**

Deactivates the SDC peripheral.

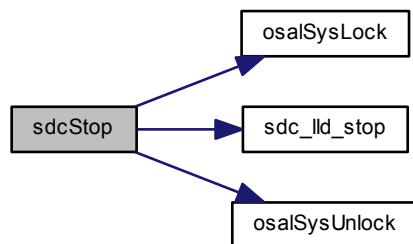
**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.31.7.18 bool sdcConnect ( SDCDriver \* sdc )

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the BLK\_READY state where it is possible to perform read and write operations.

#### Parameters

in	<i>sdc</i>	pointer to the <a href="#">SDCDriver</a> object
----	------------	---

#### Returns

The operation status.

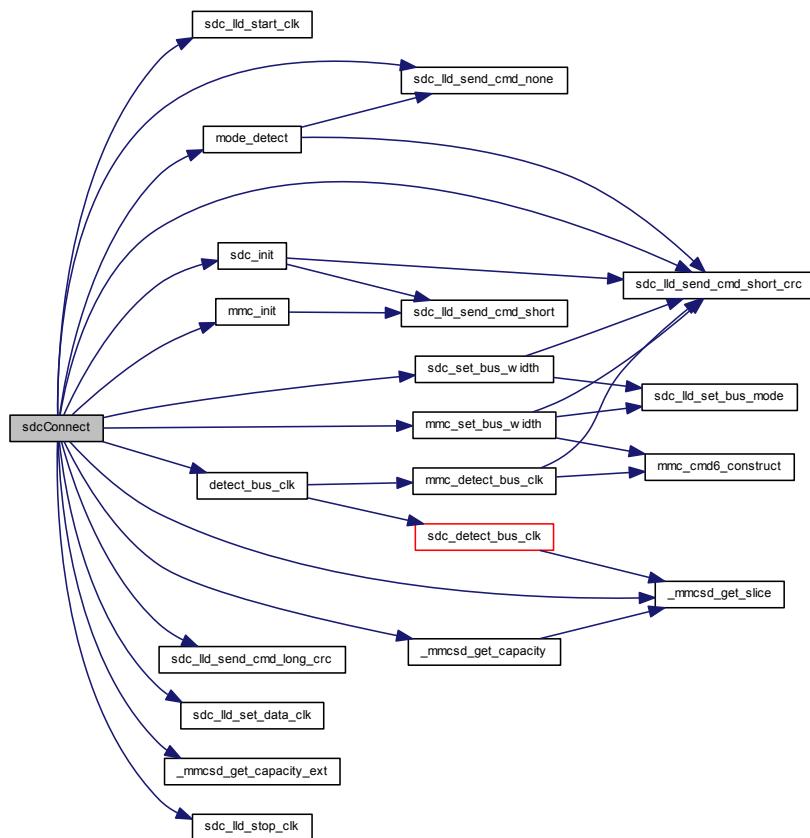
#### Return values

<a href="#">HAL_SUCCESS</a>	operation succeeded.
<a href="#">HAL_FAILED</a>	operation failed.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



**7.31.7.19 bool sdcDisconnect ( **SDCDriver** \* *sdcp* )**

Brings the driver in a state safe for card removal.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
----	-------------	---

**Returns**

The operation status.

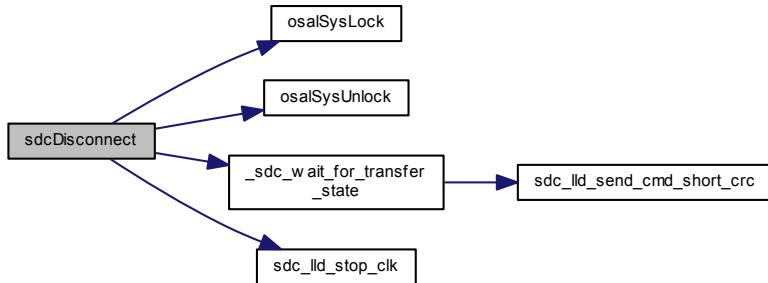
**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.31.7.20 bool sdcRead ( SDCDriver \* *sdcp*, uint32\_t *startblk*, uint8\_t \* *buf*, uint32\_t *n* )**

Reads one or more blocks.

**Precondition**

The driver must be in the `BLK_READY` state after a successful `sdcConnect()` invocation.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
in	<i>startblk</i>	first block to read
out	<i>buf</i>	pointer to the read buffer
in	<i>n</i>	number of blocks to read

**Returns**

The operation status.

### Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.31.7.21 bool sdcWrite ( SDCDriver \* *sdc*, uint32\_t *startblk*, const uint8\_t \* *buf*, uint32\_t *n* )

Writes one or more blocks.

#### Precondition

The driver must be in the `BLK_READY` state after a successful `sdcConnect()` invocation.

#### Parameters

in	<i>sdc</i>	pointer to the <code>SDCDriver</code> object
in	<i>startblk</i>	first block to write
out	<i>buf</i>	pointer to the write buffer
in	<i>n</i>	number of blocks to write

#### Returns

The operation status.

#### Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.31.7.22 `sdcflags_t sdcGetAndClearErrors ( SDCDriver * sdc )`

Returns the errors mask associated to the previous operation.

#### Parameters

in	<code>sdc</code>	pointer to the <code>SDCDriver</code> object
----	------------------	--

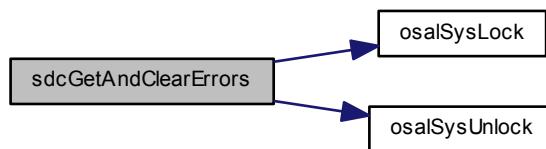
#### Returns

The errors mask.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.31.7.23 `bool sdcSync ( SDCDriver * sdc )`

Waits for card idle condition.

#### Parameters

in	<code>sdc</code>	pointer to the <code>SDCDriver</code> object
----	------------------	--

#### Returns

The operation status.

#### Return values

<code>HAL_SUCCESS</code>	the operation succeeded.
<code>HAL_FAILED</code>	the operation failed.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.31.7.24 bool sdcGetInfo ( SDCDriver \* *sdcp*, BlockDeviceInfo \* *b dip* )

Returns the media info.

#### Parameters

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
out	<i>b dip</i>	pointer to a <a href="#">BlockDeviceInfo</a> structure

#### Returns

The operation status.

#### Return values

<a href="#">HAL_SUCCESS</a>	the operation succeeded.
<a href="#">HAL_FAILED</a>	the operation failed.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.31.7.25 bool sdcErase ( SDCDriver \* *sdcp*, uint32\_t *startblk*, uint32\_t *endblk* )

Erases the supplied blocks.

#### Parameters

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
in	<i>startblk</i>	starting block number
in	<i>endblk</i>	ending block number

#### Returns

The operation status.

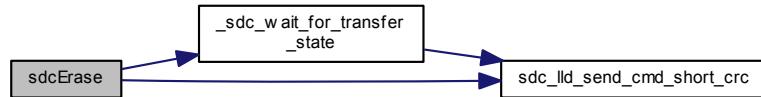
#### Return values

<a href="#">HAL_SUCCESS</a>	the operation succeeded.
<a href="#">HAL_FAILED</a>	the operation failed.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.31.7.26 void sdcIldInit( void )

Low level SDC driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.31.7.27 void sdcIldStart( SDCDriver \* sdcp )

Configures and activates the SDC peripheral.

**Parameters**

in	sdcp	pointer to the <a href="#">SDCDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

#### 7.31.7.28 void sdcIldStop( SDCDriver \* sdcp )

Deactivates the SDC peripheral.

**Parameters**

in	sdcp	pointer to the <a href="#">SDCDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.29 void sdc\_lld\_start\_clk( SDCDriver \* sdcp )**

Starts the SDIO clock and sets it to init mode (400kHz or less).

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.30 void sdc\_lld\_set\_data\_clk ( SDCDriver \* *sdcp*, sdcbusclk\_t *clk* )**

Sets the SDIO clock to data mode (25MHz or less).

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
in	<i>clk</i>	the clock mode

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.31 void sdc\_lld\_stop\_clk ( SDCDriver \* *sdcp* )**

Stops the SDIO clock.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.32 void sdc\_lld\_set\_bus\_mode ( SDCDriver \* *sdcp*, sdcbusmode\_t *mode* )**

Switches the bus to 4 bits mode.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
in	<i>mode</i>	bus mode

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.33 void sdc\_lld\_send\_cmd\_none ( SDCDriver \* *sdcp*, uint8\_t *cmd*, uint32\_t *arg* )**

Sends an SDIO command with no response expected.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument

**Function Class:**

Not an API, this function is for internal use only.

7.31.7.34 `bool sdc_lld_send_cmd_short( SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp )`

Sends an SDIO command with a short response expected.

#### Note

The CRC is not verified.

#### Parameters

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (one word)

#### Returns

The operation status.

#### Return values

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

#### Function Class:

Not an API, this function is for internal use only.

7.31.7.35 `bool sdc_lld_send_cmd_short_crc( SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp )`

Sends an SDIO command with a short response expected and CRC.

#### Parameters

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (one word)

#### Returns

The operation status.

#### Return values

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

#### Function Class:

Not an API, this function is for internal use only.

7.31.7.36 `bool sdc_lld_send_cmd_long_crc( SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp )`

Sends an SDIO command with a long response expected and CRC.

**Parameters**

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (four words)

**Returns**

The operation status.

**Return values**

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.37 bool sdc\_lld\_read ( `SDCDriver` \* *sdcp*, `uint32_t` *startblk*, `uint8_t` \* *buf*, `uint32_t` *n* )**

Reads one or more blocks.

**Parameters**

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>startblk</i>	first block to read
out	<i>buf</i>	pointer to the read buffer
in	<i>n</i>	number of blocks to read

**Returns**

The operation status.

**Return values**

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.38 bool sdc\_lld\_write ( `SDCDriver` \* *sdcp*, `uint32_t` *startblk*, `const uint8_t` \* *buf*, `uint32_t` *n* )**

Writes one or more blocks.

**Parameters**

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>startblk</i>	first block to write
out	<i>buf</i>	pointer to the write buffer
in	<i>n</i>	number of blocks to write

**Returns**

The operation status.

**Return values**

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

**Function Class:**

Not an API, this function is for internal use only.

**7.31.7.39 bool sdc\_lld\_sync ( SDCDriver \* sdc )**

Waits for card idle condition.

**Parameters**

in	<i>sdc</i>	pointer to the <a href="#">SDCDriver</a> object
----	------------	---

**Returns**

The operation status.

**Return values**

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	the operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.31.8 Variable Documentation****7.31.8.1 const struct SDCDriverVMT sdc\_vmt [static]****Initial value:**

```
= {
    (bool (*)(void *))sdc_lld_is_card_inserted,
    (bool (*)(void *))sdc_lld_is_write_protected,
    (bool (*)(void *))sdcConnect,
    (bool (*)(void *))sdcDisconnect,
    (bool (*)(void *, uint32_t, uint8_t *, uint32_t))sdcRead,
    (bool (*)(void *, uint32_t, const uint8_t *, uint32_t))sdcWrite,
    (bool (*)(void *))sdcSync,
    (bool (*)(void *, BlockDeviceInfo *))sdcGetInfo
}
```

Virtual methods table.

**7.31.8.2 SDCDriver SDCD1**

SDCD1 driver identifier.

## 7.32 Serial Driver

Generic Serial Driver.

### 7.32.1 Detailed Description

Generic Serial Driver.

This module implements a generic full duplex serial driver. The driver implements a [SerialDriver](#) interface and uses I/O Queues for communication between the upper and the lower driver. Event flags are used to notify the application about incoming data, outgoing data and other I/O events.

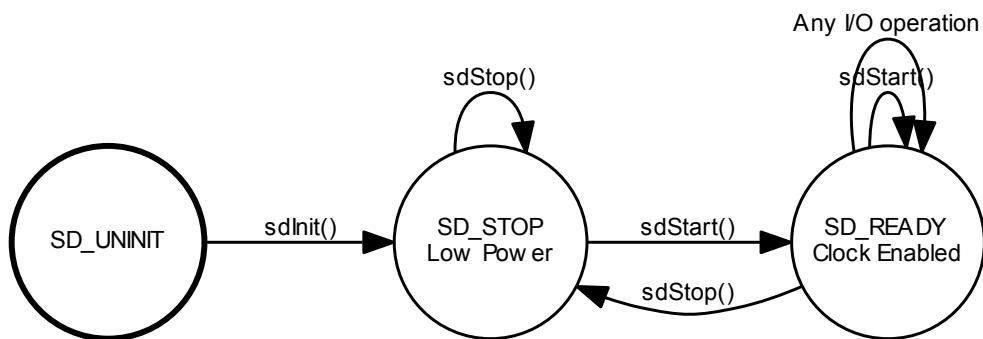
The module also contains functions that make the implementation of the interrupt service routines much easier.

#### Precondition

In order to use the SERIAL driver the `HAL_USE_SERIAL` option must be enabled in `halconf.h`.

### 7.32.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



#### Macros

- `#define _serial_driver_methods _base_asynchronous_channel_methods`  
*SerialDriver specific methods.*
- `#define _serial_driver_data`  
*SerialDriver specific data.*

#### Serial status flags

- `#define SD_PARITY_ERROR (eventflags_t)32`

- `#define SD_FRAMING_ERROR (eventflags_t)64`  
*Framing.*
- `#define SD_OVERRUN_ERROR (eventflags_t)128`  
*Overflow.*
- `#define SD_NOISE_ERROR (eventflags_t)256`  
*Line noise.*
- `#define SD_BREAK_DETECTED (eventflags_t)512`  
*LIN Break.*

## Serial configuration options

- `#define SERIAL_DEFAULT_BITRATE 38400`  
*Default bit rate.*
- `#define SERIAL_BUFFERS_SIZE 16`  
*Serial buffers size.*

## Macro Functions

- `#define sdPut(sdp, b) oqPut(&(sdp)->oqueue, b)`  
*Direct write to a `SerialDriver`.*
- `#define sdPutTimeout(sdp, b, t) oqPutTimeout(&(sdp)->oqueue, b, t)`  
*Direct write to a `SerialDriver` with timeout specification.*
- `#define sdGet(sdp) iqGet(&(sdp)->iqueue)`  
*Direct read from a `SerialDriver`.*
- `#define sdGetTimeout(sdp, t) iqGetTimeout(&(sdp)->iqueue, t)`  
*Direct read from a `SerialDriver` with timeout specification.*
- `#define sdWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)`  
*Direct blocking write to a `SerialDriver`.*
- `#define sdWriteTimeout(sdp, b, n, t) oqWriteTimeout(&(sdp)->oqueue, b, n, t)`  
*Direct blocking write to a `SerialDriver` with timeout specification.*
- `#define sdAsynchronousWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)`  
*Direct non-blocking write to a `SerialDriver`.*
- `#define sdRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)`  
*Direct blocking read from a `SerialDriver`.*
- `#define sdReadTimeout(sdp, b, n, t) iqReadTimeout(&(sdp)->iqueue, b, n, t)`  
*Direct blocking read from a `SerialDriver` with timeout specification.*
- `#define sdAsynchronousRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)`  
*Direct non-blocking read from a `SerialDriver`.*

## PLATFORM configuration options

- `#define PLATFORM_SERIAL_USE_USART1 FALSE`  
*USART1 driver enable switch.*

## Typedefs

- `typedef struct SerialDriver SerialDriver`  
*Structure representing a serial driver.*

## Data Structures

- struct `SerialDriverVMT`  
`SerialDriver` virtual methods table.
- struct `SerialDriver`  
`Full duplex serial driver class.`
- struct `SerialConfig`  
`PLATFORM Serial Driver configuration structure.`

## Functions

- void `sdInit` (void)  
`Serial Driver initialization.`
- void `sdObjectInit` (`SerialDriver` \*`sdp`, `qnotify_t` `inotify`, `qnotify_t` `onotify`)  
`Initializes a generic full duplex driver object.`
- void `sdStart` (`SerialDriver` \*`sdp`, const `SerialConfig` \*`config`)  
`Configures and starts the driver.`
- void `sdStop` (`SerialDriver` \*`sdp`)  
`Stops the driver.`
- void `sdIncomingData` (`SerialDriver` \*`sdp`, `uint8_t` `b`)  
`Handles incoming data.`
- `msg_t` `sdRequestData` (`SerialDriver` \*`sdp`)  
`Handles outgoing data.`
- bool `sdPutWouldBlock` (`SerialDriver` \*`sdp`)  
`Direct output check on a SerialDriver.`
- bool `sdGetWouldBlock` (`SerialDriver` \*`sdp`)  
`Direct input check on a SerialDriver.`
- void `sd_lld_init` (void)  
`Low level serial driver initialization.`
- void `sd_lld_start` (`SerialDriver` \*`sdp`, const `SerialConfig` \*`config`)  
`Low level serial driver configuration and (re)start.`
- void `sd_lld_stop` (`SerialDriver` \*`sdp`)  
`Low level serial driver stop.`

## Enumerations

- enum `sdstate_t` { `SD_UNINIT` = 0, `SD_STOP` = 1, `SD_READY` = 2 }  
`Driver state machine possible states.`

## Variables

- `SerialDriver SD1`  
`USART1 serial driver identifier.`
- static const `SerialConfig default_config`  
`Driver default configuration.`

### 7.32.3 Macro Definition Documentation

#### 7.32.3.1 #define SD\_PARITY\_ERROR (eventflags\_t)32

Parity.

7.32.3.2 #define SD\_FRAMING\_ERROR (eventflags\_t)64

Framing.

7.32.3.3 #define SD\_OVERRUN\_ERROR (eventflags\_t)128

Overflow.

7.32.3.4 #define SD\_NOISE\_ERROR (eventflags\_t)256

Line noise.

7.32.3.5 #define SD\_BREAK\_DETECTED (eventflags\_t)512

LIN Break.

7.32.3.6 #define SERIAL\_DEFAULT\_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.32.3.7 #define SERIAL\_BUFFERS\_SIZE 16

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

#### Note

The default is 16 bytes for both the transmission and receive buffers.

7.32.3.8 #define \_serial\_driver\_methods \_base\_asynchronous\_channel\_methods

[SerialDriver](#) specific methods.

7.32.3.9 #define sdPut( sdp, b ) oqPut(&(sdp)->oqueue, b)

Direct write to a [SerialDriver](#).

#### Note

This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

#### See also

[chnPutTimeout\(\)](#)

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

```
7.32.3.10 #define sdPutTimeout( sdp, b, t ) oqPutTimeout(&(sdp)->oqueue, b, t)
```

Direct write to a [SerialDriver](#) with timeout specification.

**Note**

This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

**See also**

[chnPutTimeout\(\)](#)

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

```
7.32.3.11 #define sdGet( sdp ) iqGet(&(sdp)->iqueue)
```

Direct read from a [SerialDriver](#).

**Note**

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

**See also**

[chnGetTimeout\(\)](#)

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

```
7.32.3.12 #define sdGetTimeout( sdp, t ) iqGetTimeout(&(sdp)->iqueue, t)
```

Direct read from a [SerialDriver](#) with timeout specification.

**Note**

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

**See also**

[chnGetTimeout\(\)](#)

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

```
7.32.3.13 #define sdWrite( sdp, b, n ) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)
```

Direct blocking write to a [SerialDriver](#).

#### Note

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write from different channels implementations.

#### See also

[chnWrite\(\)](#)

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

```
7.32.3.14 #define sdWriteTimeout( sdp, b, n, t ) oqWriteTimeout(&(sdp)->oqueue, b, n, t)
```

Direct blocking write to a [SerialDriver](#) with timeout specification.

#### Note

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write to different channels implementations.

#### See also

[chnWriteTimeout\(\)](#)

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

```
7.32.3.15 #define sdAsynchronousWrite( sdp, b, n ) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)
```

Direct non-blocking write to a [SerialDriver](#).

#### Note

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write to different channels implementations.

#### See also

[chnWriteTimeout\(\)](#)

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

```
7.32.3.16 #define sdRead( sdp, b, n ) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)
```

Direct blocking read from a [SerialDriver](#).

**Note**

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

**See also**

[chnRead\(\)](#)

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

```
7.32.3.17 #define sdReadTimeout( sdp, b, n, t ) iqReadTimeout(&(sdp)->iqueue, b, n, t)
```

Direct blocking read from a [SerialDriver](#) with timeout specification.

**Note**

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

**See also**

[chnReadTimeout\(\)](#)

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

```
7.32.3.18 #define sdAsynchronousRead( sdp, b, n ) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)
```

Direct non-blocking read from a [SerialDriver](#).

**Note**

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

**See also**

[chnReadTimeout\(\)](#)

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

```
7.32.3.19 #define PLATFORM_SERIAL_USE_USART1 FALSE
```

USART1 driver enable switch.

If set to TRUE the support for USART1 is included.

**Note**

The default is FALSE.

### 7.32.3.20 #define \_serial\_driver\_data

**Value:**

```
_base_asynchronous_channel_data
/* Driver state.*/
sdstate_t state;
/* Input queue.*/
input_queue_t iqueue;
/* Output queue.*/
output_queue_t oqueue;
/* Input circular buffer.*/
uint8_t ib[SERIAL_BUFFERS_SIZE];
/* Output circular buffer.*/
uint8_t ob[SERIAL_BUFFERS_SIZE];
/* End of the mandatory fields.*/
```



SerialDriver specific data.

## 7.32.4 Typedef Documentation

### 7.32.4.1 typedef struct SerialDriver SerialDriver

Structure representing a serial driver.

## 7.32.5 Enumeration Type Documentation

### 7.32.5.1 enum sdstate\_t

Driver state machine possible states.

Enumerator

**SD\_UNINIT** Not initialized.

**SD\_STOP** Stopped.

**SD\_READY** Ready.

## 7.32.6 Function Documentation

### 7.32.6.1 void sdlInit( void )

Serial Driver initialization.

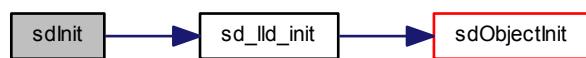
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.32.6.2 void sdObjectInit ( *SerialDriver* \* *sdp*, *qnotify\_t* *inotify*, *qnotify\_t* *onotify* )

Initializes a generic full duplex driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.

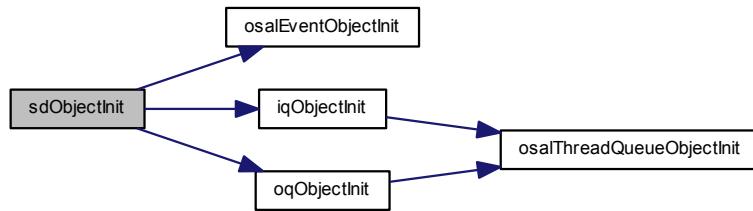
#### Parameters

out	<i>sdp</i>	pointer to a <i>SerialDriver</i> structure
in	<i>inotify</i>	pointer to a callback function that is invoked when some data is read from the Queue. The value can be NULL.
in	<i>onotify</i>	pointer to a callback function that is invoked when some data is written in the Queue. The value can be NULL.

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.32.6.3 void sdStart ( *SerialDriver* \* *sdp*, const *SerialConfig* \* *config* )

Configures and starts the driver.

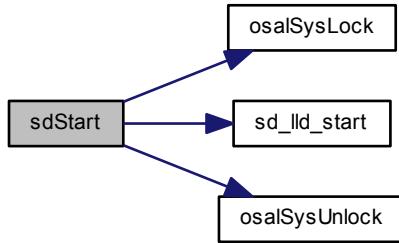
#### Parameters

in	<i>sdp</i>	pointer to a <i>SerialDriver</i> object
in	<i>config</i>	the architecture-dependent serial driver configuration. If this parameter is set to NULL then a default configuration is used.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.32.6.4 void sdStop ( SerialDriver \* sdp )

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message Q\_RESET.

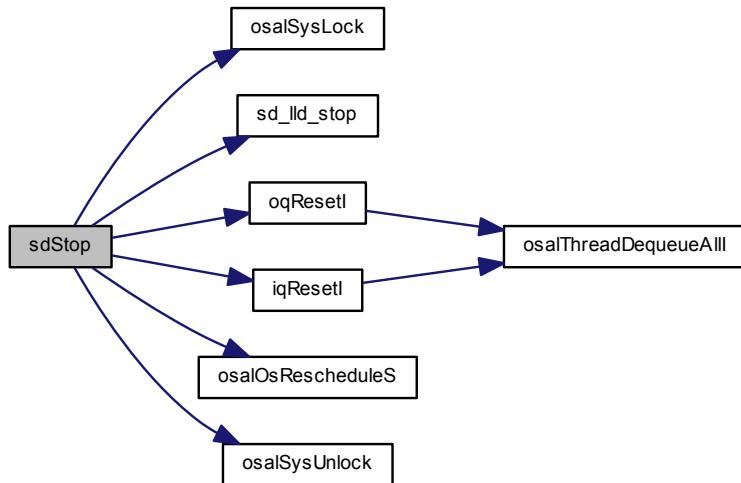
##### Parameters

in	<code>sdp</code>	pointer to a <code>SerialDriver</code> object
----	------------------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.32.6.5 void sdlIncomingData ( *SerialDriver* \* *sdp*, *uint8\_t* *b* )

Handles incoming data.

This function must be called from the input interrupt service routine in order to enqueue incoming data and generate the related events.

#### Note

The incoming data event is only generated when the input queue becomes non-empty.

In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

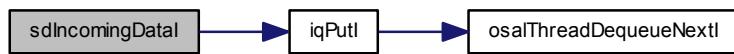
#### Parameters

in	<i>sdp</i>	pointer to a <i>SerialDriver</i> structure
in	<i>b</i>	the byte to be written in the driver's Input Queue

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.32.6.6 *msg\_t* sdrequestData ( *SerialDriver* \* *sdp* )

Handles outgoing data.

Must be called from the output interrupt service routine in order to get the next byte to be transmitted.

#### Note

In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

#### Parameters

in	<i>sdp</i>	pointer to a <i>SerialDriver</i> structure
----	------------	--

#### Returns

The byte value read from the driver's output queue.

**Return values**

<code>Q_EMPTY</code>	if the queue is empty (the lower driver usually disables the interrupt source when this happens).
----------------------	---

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**7.32.6.7 bool sdPutWouldBlock ( SerialDriver \* sdp )**

Direct output check on a [SerialDriver](#).

**Note**

This function bypasses the indirect access to the channel and checks directly the output queue. This is faster but cannot be used to check different channels implementations.

**Parameters**

in	<code>sdp</code>	pointer to a <a href="#">SerialDriver</a> structure
----	------------------	---

**Returns**

The queue status.

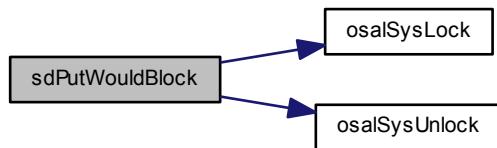
**Return values**

<code>false</code>	if the next write operation would not block.
<code>true</code>	if the next write operation would block.

**Deprecated****Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.32.6.8 bool sdGetWouldBlock ( *SerialDriver* \* *sdp* )

Direct input check on a [SerialDriver](#).

##### Note

This function bypasses the indirect access to the channel and checks directly the input queue. This is faster but cannot be used to check different channels implementations.

##### Parameters

in	<i>sdp</i>	pointer to a <a href="#">SerialDriver</a> structure
----	------------	---

##### Returns

The queue status.

##### Return values

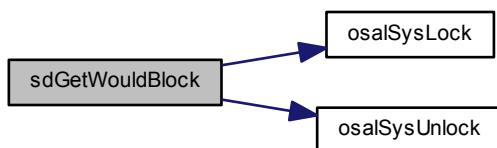
<i>false</i>	if the next write operation would not block.
<i>true</i>	if the next write operation would block.

##### Deprecated

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



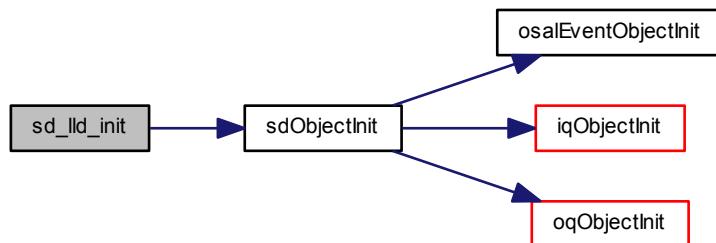
### 7.32.6.9 void sd\_lld\_init( void )

Low level serial driver initialization.

#### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



### 7.32.6.10 void sd\_lld\_start( SerialDriver \* sdp, const SerialConfig \* config )

Low level serial driver configuration and (re)start.

#### Parameters

in	<i>sdp</i>	pointer to a <a href="#">SerialDriver</a> object
in	<i>config</i>	the architecture-dependent serial driver configuration. If this parameter is set to NULL then a default configuration is used.

#### Function Class:

Not an API, this function is for internal use only.

### 7.32.6.11 void sd\_lld\_stop( SerialDriver \* sdp )

Low level serial driver stop.

De-initializes the USART, stops the associated clock, resets the interrupt vector.

#### Parameters

in	<i>sdp</i>	pointer to a <a href="#">SerialDriver</a> object
----	------------	--

#### Function Class:

Not an API, this function is for internal use only.

## 7.32.7 Variable Documentation

### 7.32.7.1 SerialDriver SD1

USART1 serial driver identifier.

**7.32.7.2 const SerialConfig default\_config [static]****Initial value:**

```
= {  
    38400  
}
```

Driver default configuration.

## 7.33 Serial over USB Driver

Serial over USB Driver.

### 7.33.1 Detailed Description

Serial over USB Driver.

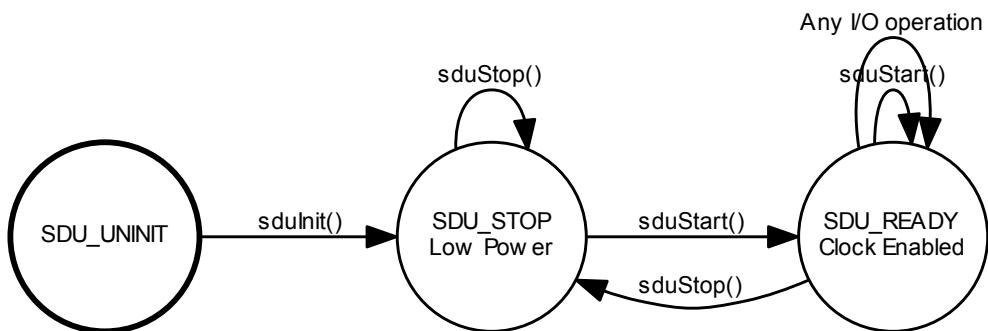
This module implements an USB Communication Device Class (CDC) as a normal serial communication port accessible from the device application.

#### Precondition

In order to use the USB over Serial driver the `HAL_USE_SERIAL_USB` option must be enabled in `halconf.h`.

### 7.33.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



#### Macros

- `#define _serial_usb_driver_data`  
*SerialDriver specific data.*
- `#define _serial_usb_driver_methods _base_asynchronous_channel_methods`  
*SerialUSBDriver specific methods.*

#### SERIAL\_USB configuration options

- `#define SERIAL_USB_BUFFERS_SIZE 256`  
*Serial over USB buffers size.*

- `#define SERIAL_USB_BUFFERS_NUMBER 2`  
*Serial over USB number of buffers.*

## Typedefs

- `typedef struct SerialUSBDriver SerialUSBDriver`  
*Structure representing a serial over USB driver.*

## Data Structures

- `struct SerialUSBConfig`  
*Serial over USB Driver configuration structure.*
- `struct SerialUSBDriverVMT`  
*SerialDriver virtual methods table.*
- `struct SerialUSBDriver`  
*Full duplex serial driver class.*

## Functions

- `static void ibnotify (io_buffers_queue_t *bqp)`  
*Notification of empty buffer released into the input buffers queue.*
- `static void obnotify (io_buffers_queue_t *bqp)`  
*Notification of filled buffer inserted into the output buffers queue.*
- `void sduInit (void)`  
*Serial Driver initialization.*
- `void sduObjectInit (SerialUSBDriver *sdup)`  
*Initializes a generic full duplex driver object.*
- `void sduStart (SerialUSBDriver *sdup, const SerialUSBConfig *config)`  
*Configures and starts the driver.*
- `void sduStop (SerialUSBDriver *sdup)`  
*Stops the driver.*
- `void sduDisconnect (SerialUSBDriver *sdup)`  
*USB device disconnection handler.*
- `void sduConfigureHookI (SerialUSBDriver *sdup)`  
*USB device configured handler.*
- `bool sduRequestsHook (USBDriver *usbp)`  
*Default requests hook.*
- `void sduSOFHookI (SerialUSBDriver *sdup)`  
*SOF handler.*
- `void sduDataTransmitted (USBDriver *usbp, usbep_t ep)`  
*Default data transmitted callback.*
- `void sduDataReceived (USBDriver *usbp, usbep_t ep)`  
*Default data received callback.*
- `void sduInterruptTransmitted (USBDriver *usbp, usbep_t ep)`  
*Default data received callback.*

## Enumerations

- `enum sdustate_t { SDU_UNINIT = 0, SDU_STOP = 1, SDU_READY = 2 }`  
*Driver state machine possible states.*

### 7.33.3 Macro Definition Documentation

#### 7.33.3.1 #define SERIAL\_USB\_BUFFERS\_SIZE 256

Serial over USB buffers size.

Configuration parameter, the buffer size must be a multiple of the USB data endpoint maximum packet size.

##### Note

The default is 256 bytes for both the transmission and receive buffers.

#### 7.33.3.2 #define SERIAL\_USB\_BUFFERS\_NUMBER 2

Serial over USB number of buffers.

##### Note

The default is 2 buffers.

#### 7.33.3.3 #define \_serial\_usb\_driver\_data

##### Value:

```
_base_asynchronous_channel_data
/* Driver state.*/
sdustate_t state;
/* Input buffers queue.*/
input_buffers_queue_t ibqueue;
/* Output queue.*/
output_buffers_queue_t obqueue;
/* Input buffer.*/
uint8_t ib[BQ_BUFFER_SIZE(
    SERIAL_USB_BUFFERS_NUMBER, \
        SERIAL_USB_BUFFERS_SIZE)];
/* Output buffer.*/
uint8_t ob[BQ_BUFFER_SIZE(
    SERIAL_USB_BUFFERS_NUMBER, \
        SERIAL_USB_BUFFERS_SIZE)];
/* End of the mandatory fields.*/
/* Current configuration data.*/
const SerialUSBCConfig *config;
```

[SerialDriver](#) specific data.

#### 7.33.3.4 #define \_serial\_usb\_driver\_methods \_base\_asynchronous\_channel\_methods

[SerialUSB](#)[Driver](#) specific methods.

### 7.33.4 Typedef Documentation

#### 7.33.4.1 typedef struct SerialUSBDriver SerialUSBDriver

Structure representing a serial over USB driver.

### 7.33.5 Enumeration Type Documentation

#### 7.33.5.1 enum sdustate\_t

Driver state machine possible states.

## Enumerator

**SDU\_UNINIT** Not initialized.

**SDU\_STOP** Stopped.

**SDU\_READY** Ready.

## 7.33.6 Function Documentation

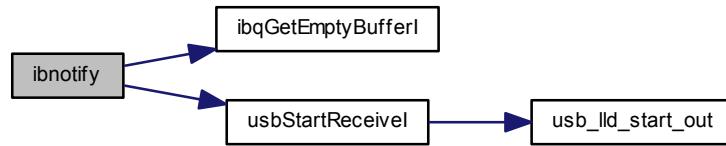
## 7.33.6.1 static void ibnotify ( io\_buffers\_queue\_t \* bqp ) [static]

Notification of empty buffer released into the input buffers queue.

## Parameters

in	bqp	the buffers queue pointer.
----	-----	----------------------------

Here is the call graph for this function:



## 7.33.6.2 static void obnotify ( io\_buffers\_queue\_t \* bqp ) [static]

Notification of filled buffer inserted into the output buffers queue.

## Parameters

in	bqp	the buffers queue pointer.
----	-----	----------------------------

Here is the call graph for this function:



## 7.33.6.3 void sdulinit ( void )

Serial Driver initialization.

**Note**

This function is implicitly invoked by [halInit \(\)](#), there is no need to explicitly initialize the driver.

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

**7.33.6.4 void sduObjectInit ( [SerialUSBDriver](#) \* *sdup* )**

Initializes a generic full duplex driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.

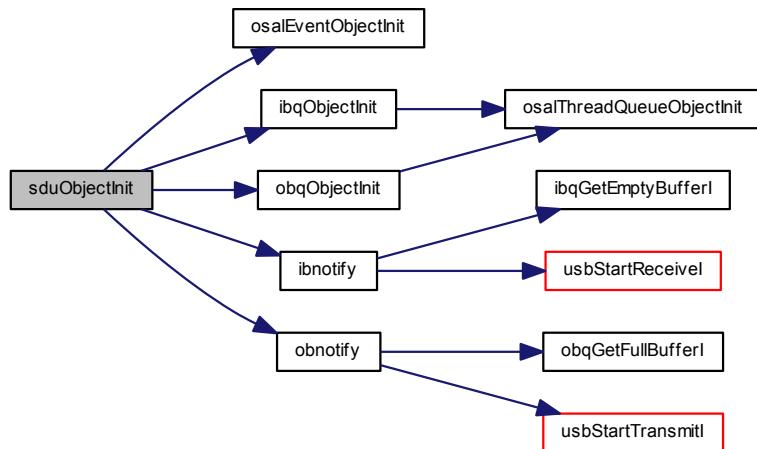
**Parameters**

out	<i>sdup</i>	pointer to a <a href="#">SerialUSBDriver</a> structure
-----	-------------	--

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

**7.33.6.5 void sduStart ( [SerialUSBDriver](#) \* *sdup*, const [SerialUSBConfig](#) \* *config* )**

Configures and starts the driver.

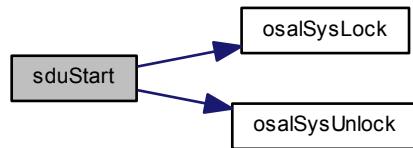
**Parameters**

in	<i>sdup</i>	pointer to a <a href="#">SerialUSBDriver</a> object
in	<i>config</i>	the serial over USB driver configuration

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.33.6.6 void sduStop ( SerialUSBDriver \* *sdup* )

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message Q\_RESET.

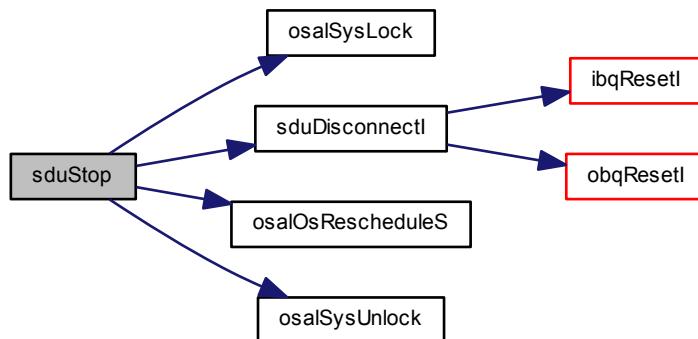
##### Parameters

in	<i>sdup</i>	pointer to a <a href="#">SerialUSBDriver</a> object
----	-------------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.33.6.7 void sduDisconnectl ( SerialUSBDriver \* *sdup* )

USB device disconnection handler.

##### Note

If this function is not called from an ISR then an explicit call to [osalOsRescheduleS\(\)](#) is necessary afterward.

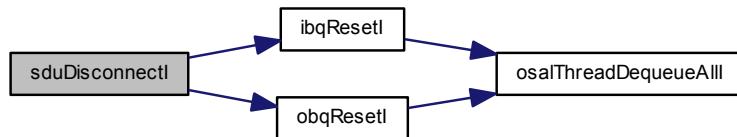
## Parameters

in	<i>sdu</i>	pointer to a <a href="#">SerialUSBDriver</a> object
----	------------	---

## Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

7.33.6.8 void sduConfigureHookI ( [SerialUSBDriver](#) \* *sdu* )

USB device configured handler.

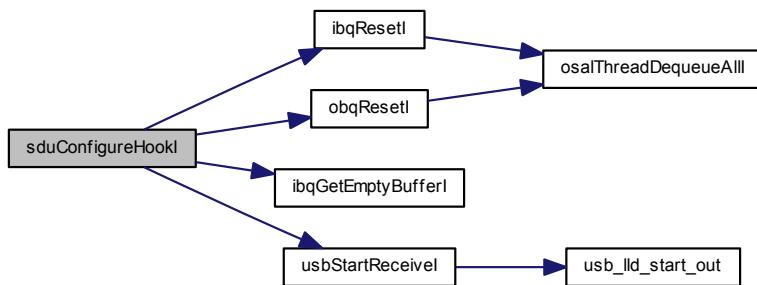
## Parameters

in	<i>sdu</i>	pointer to a <a href="#">SerialUSBDriver</a> object
----	------------	---

## Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

7.33.6.9 bool sduRequestsHook ( [USBDriver](#) \* *usdp* )

Default requests hook.

Applications wanting to use the Serial over USB driver can use this function as requests hook in the USB configuration. The following requests are emulated:

- CDC\_GET\_LINE\_CODING.
- CDC\_SET\_LINE\_CODING.
- CDC\_SET\_CONTROL\_LINE\_STATE.

#### Parameters

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

#### Returns

The hook status.

#### Return values

<i>true</i>	Message handled internally.
<i>false</i>	Message not handled.

### 7.33.6.10 void sduSOFHookl ( [SerialUSBDriver](#) \* *sdup* )

SOF handler.

The SOF interrupt is used for automatic flushing of incomplete buffers pending in the output queue.

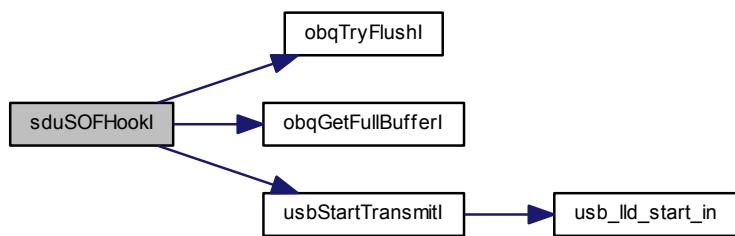
#### Parameters

in	<i>sdup</i>	pointer to a <a href="#">SerialUSBDriver</a> object
----	-------------	---

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.33.6.11 void sduDataTransmitted ( [USBDriver](#) \* *usbp*, [usbep\\_t](#) *ep* )

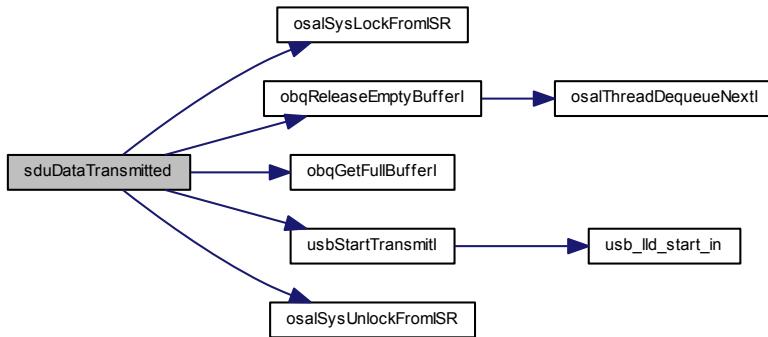
Default data transmitted callback.

The application must use this function as callback for the IN data endpoint.

## Parameters

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	IN endpoint number

Here is the call graph for this function:



### 7.33.6.12 void sduDataReceived ( [USBDriver](#) \* *usbp*, [usbep\\_t](#) *ep* )

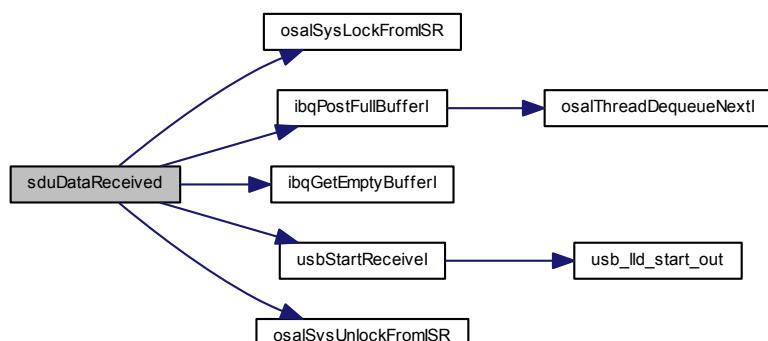
Default data received callback.

The application must use this function as callback for the OUT data endpoint.

## Parameters

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	OUT endpoint number

Here is the call graph for this function:



**7.33.6.13 void sduInterruptTransmitted ( **USBDriver** \* *usbp*, **usbep\_t** *ep* )**

Default data received callback.

The application must use this function as callback for the IN interrupt endpoint.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

## 7.34 SPI Driver

Generic SPI Driver.

### 7.34.1 Detailed Description

Generic SPI Driver.

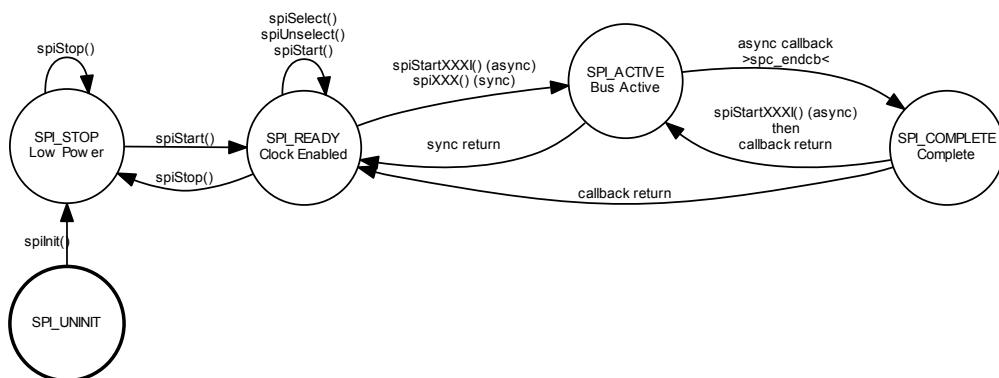
This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

#### Precondition

In order to use the SPI driver the `HAL_USE_SPI` option must be enabled in `halconf.h`.

### 7.34.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



The driver is not thread safe for performance reasons, if you need to access the SPI bus from multiple threads then use the `spiAcquireBus()` and `spiReleaseBus()` APIs in order to gain exclusive access.

#### SPI configuration options

- `#define SPI_USE_WAIT TRUE`  
*Enables synchronous APIs.*
- `#define SPI_USE_MUTUAL_EXCLUSION TRUE`  
*Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.*

#### Macro Functions

- `#define spiSelectI(spip)`

- **#define spiUnselectl(spi)**  
*Asserts the slave select signal and prepares for transfers.*
- **#define spiStartIgnoreL(spi, n)**  
*Deasserts the slave select signal.*
- **#define spiStartExchangeL(spi, n, txbuf, rxbuf)**  
*Ignores data on the SPI bus.*
- **#define spiStartSendL(spi, n, txbuf)**  
*Exchanges data on the SPI bus.*
- **#define spiStartReceiveL(spi, n, rxbuf)**  
*Sends data over the SPI bus.*
- **#define spiPolledExchange(spi, frame) spi\_lld\_polled\_exchange(spi, frame)**  
*Receives data from the SPI bus.*
- **#define spiPolledExchange(spi, frame) spi\_lld\_polled\_exchange(spi, frame)**  
*Exchanges one frame using a polled wait.*

## Low level driver helper macros

- **#define \_spi\_wakeup\_isr(spi)**  
*Wakes up the waiting thread.*
- **#define \_spi\_isr\_code(spi)**  
*Common ISR code.*

## PLATFORM configuration options

- **#define PLATFORM\_SPI\_USE\_SPI1 FALSE**  
*SPI1 driver enable switch.*

## Typedefs

- **typedef struct SPIDriver SPIDriver**  
*Type of a structure representing an SPI driver.*
- **typedef void(\* spicallback\_t) (SPIDriver \*spip)**  
*SPI notification callback type.*

## Data Structures

- **struct SPIConfig**  
*Driver configuration structure.*
- **struct SPIDriver**  
*Structure representing an SPI driver.*

## Functions

- **void spiInit (void)**  
*SPI Driver initialization.*
- **void spiObjectInit (SPIDriver \*spip)**  
*Initializes the standard part of a `SPIDriver` structure.*
- **void spiStart (SPIDriver \*spip, const SPIConfig \*config)**  
*Configures and activates the SPI peripheral.*
- **void spiStop (SPIDriver \*spip)**

- **Deactivates the SPI peripheral.**
- void **spiSelect (SPIDriver \*spip)**  
*Asserts the slave select signal and prepares for transfers.*
- void **spiUnselect (SPIDriver \*spip)**  
*Deasserts the slave select signal.*
- void **spiStartIgnore (SPIDriver \*spip, size\_t n)**  
*Ignores data on the SPI bus.*
- void **spiStartExchange (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)**  
*Exchanges data on the SPI bus.*
- void **spiStartSend (SPIDriver \*spip, size\_t n, const void \*txbuf)**  
*Sends data over the SPI bus.*
- void **spiStartReceive (SPIDriver \*spip, size\_t n, void \*rxbuf)**  
*Receives data from the SPI bus.*
- void **spignore (SPIDriver \*spip, size\_t n)**  
*Ignores data on the SPI bus.*
- void **spiExchange (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)**  
*Exchanges data on the SPI bus.*
- void **spiSend (SPIDriver \*spip, size\_t n, const void \*txbuf)**  
*Sends data over the SPI bus.*
- void **spiReceive (SPIDriver \*spip, size\_t n, void \*rxbuf)**  
*Receives data from the SPI bus.*
- void **spiAcquireBus (SPIDriver \*spip)**  
*Gains exclusive access to the SPI bus.*
- void **spiReleaseBus (SPIDriver \*spip)**  
*Releases exclusive access to the SPI bus.*
- void **spi\_lld\_init (void)**  
*Low level SPI driver initialization.*
- void **spi\_lld\_start (SPIDriver \*spip)**  
*Configures and activates the SPI peripheral.*
- void **spi\_lld\_stop (SPIDriver \*spip)**  
*Deactivates the SPI peripheral.*
- void **spi\_lld\_select (SPIDriver \*spip)**  
*Asserts the slave select signal and prepares for transfers.*
- void **spi\_lld\_unselect (SPIDriver \*spip)**  
*Deasserts the slave select signal.*
- void **spi\_lld\_ignore (SPIDriver \*spip, size\_t n)**  
*Ignores data on the SPI bus.*
- void **spi\_lld\_exchange (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)**  
*Exchanges data on the SPI bus.*
- void **spi\_lld\_send (SPIDriver \*spip, size\_t n, const void \*txbuf)**  
*Sends data over the SPI bus.*
- void **spi\_lld\_receive (SPIDriver \*spip, size\_t n, void \*rxbuf)**  
*Receives data from the SPI bus.*
- uint16\_t **spi\_lld\_polled\_exchange (SPIDriver \*spip, uint16\_t frame)**  
*Exchanges one frame using a polled wait.*

## Enumerations

- enum **spistate\_t {**  
**SPI\_UNINIT** = 0, **SPI\_STOP** = 1, **SPI\_READY** = 2, **SPI\_ACTIVE** = 3,  
**SPI\_COMPLETE** = 4 **}**  
*Driver state machine possible states.*

## Variables

- **SPIDriver SPID1**

*SPI1 driver identifier.*

### 7.34.3 Macro Definition Documentation

#### 7.34.3.1 #define SPI\_USE\_WAIT TRUE

Enables synchronous APIs.

##### Note

Disabling this option saves both code and data space.

#### 7.34.3.2 #define SPI\_USE\_MUTUAL\_EXCLUSION TRUE

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

##### Note

Disabling this option saves both code and data space.

#### 7.34.3.3 #define spiSelect( spip )

##### Value:

```
{                                     \
    spi_lld_select(spip);           \
}
```

Asserts the slave select signal and prepares for transfers.

##### Parameters

in	<code>spip</code>	pointer to the <code>SPIDriver</code> object
----	-------------------	--

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

#### 7.34.3.4 #define spiUnselect( spip )

##### Value:

```
{                                     \
    spi_lld_unselect(spip);           \
}
```

Deasserts the slave select signal.

The previously selected peripheral is unselected.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.34.3.5 #define spiStartIgnore( *spip*, *n* )****Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    \
    spi_llld_ignore(spip, n);
}
```

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

**Precondition**

A slave must have been selected using [spiSelect\(\)](#) or [spiSelectI\(\)](#).

**Postcondition**

At the end of the operation the configured callback is invoked.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
in	<i>n</i>	number of words to be ignored

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.34.3.6 #define spiStartExchange( *spip*, *n*, *txbuf*, *rxbuf* )****Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    \
    spi_llld_exchange(spip, n, txbuf, rxbuf);
}
```

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

**Precondition**

A slave must have been selected using [spiSelect\(\)](#) or [spiSelectI\(\)](#).

**Postcondition**

At the end of the operation the configured callback is invoked.

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive buffer

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.34.3.7 #define spiStartSend( *spip*, *n*, *txbuf* )****Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    \
    spi_lld_send(spip, n, txbuf);
}
```

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

**Precondition**

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.34.3.8 #define spiStartReceive( *spip*, *n*, *rxbuf* )

#### Value:

```
{
  (spip) ->state = SPI_ACTIVE;
  \
  \spi_lld_receive(spip, n, rxbuf);
}
```

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

#### Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

#### Postcondition

At the end of the operation the configured callback is invoked.

#### Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

#### Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to receive
out	<i>rxbuf</i>	the pointer to the receive buffer

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.34.3.9 #define spiPolledExchange( *spip*, *frame* ) spi\_lld\_polled\_exchange(*spip*, *frame*)

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

#### Note

This API is implemented as a macro in order to minimize latency.

#### Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>frame</i>	the data frame to send over the SPI bus

#### Returns

The received data frame from the SPI bus.

## 7.34.3.10 #define \_spi\_wakeup\_isr( spip )

**Value:**

```
{
    osalSysLockFromISR();
    \
    osalThreadResumeI(&(spip)->thread, MSG_OK);
    \
    osalSysUnlockFromISR();
}
```

Wakes up the waiting thread.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

## 7.34.3.11 #define \_spi\_isr\_code( spip )

**Value:**

```
{
    if ((spip)->config->end_cb) {
        (spip)->state = SPI_COMPLETE;
        (spip)->config->end_cb(spip);
        if ((spip)->state == SPI_COMPLETE)
            (spip)->state = SPI_READY;
    }
    else
        (spip)->state = SPI_READY;
    \
    _spi_wakeup_isr(spip);
}
```

Common ISR code.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

7.34.3.12 #define PLATFORM\_SPI\_USE\_SPI1 FALSE

SPI1 driver enable switch.

If set to TRUE the support for SPI1 is included.

#### Note

The default is FALSE.

### 7.34.4 Typedef Documentation

#### 7.34.4.1 typedef struct SPIDriver SPIDriver

Type of a structure representing an SPI driver.

#### 7.34.4.2 typedef void(\* spicallback\_t)(SPIDriver \*spip)

SPI notification callback type.

##### Parameters

in	spip	pointer to the <a href="#">SPIDriver</a> object triggering the callback
----	------	---

### 7.34.5 Enumeration Type Documentation

#### 7.34.5.1 enum spistate\_t

Driver state machine possible states.

##### Enumerator

**SPI\_UNINIT** Not initialized.

**SPI\_STOP** Stopped.

**SPI\_READY** Ready.

**SPI\_ACTIVE** Exchanging data.

**SPI\_COMPLETE** Asynchronous operation complete.

### 7.34.6 Function Documentation

#### 7.34.6.1 void spilinit( void )

SPI Driver initialization.

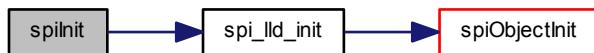
##### Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.34.6.2 void spiObjectInit ( [SPIDriver](#) \* *spip* )

Initializes the standard part of a [SPIDriver](#) structure.

##### Parameters

out	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
-----	-------------	---

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.34.6.3 void spiStart ( [SPIDriver](#) \* *spip*, const [SPIConfig](#) \* *config* )

Configures and activates the SPI peripheral.

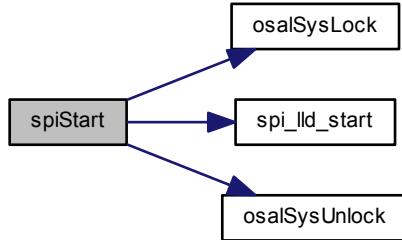
##### Parameters

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
in	<i>config</i>	pointer to the <a href="#">SPIConfig</a> object

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.34.6.4 void spiStop ( SPIDriver \* spip )

Deactivates the SPI peripheral.

##### Note

Deactivating the peripheral also enforces a release of the slave select line.

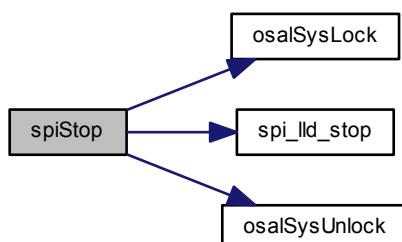
##### Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
----	-------------	--

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.34.6.5 void spiSelect ( SPIDriver \* spip )

Asserts the slave select signal and prepares for transfers.

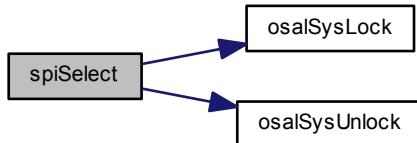
**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.34.6.6 void spiUnselect ( SPIDriver \* spip )**

Deasserts the slave select signal.

The previously selected peripheral is unselected.

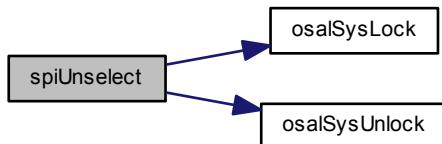
**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.34.6.7 void spiStartIgnore ( SPIDriver \* spip, size\_t n )**

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

**Precondition**

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

**Postcondition**

At the end of the operation the configured callback is invoked.

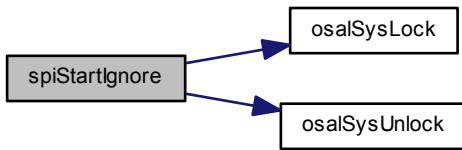
**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be ignored

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.34.6.8 void spiStartExchange ( SPIDriver \* spip, size\_t n, const void \* txbuf, void \* rdbuf )**

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

**Precondition**

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

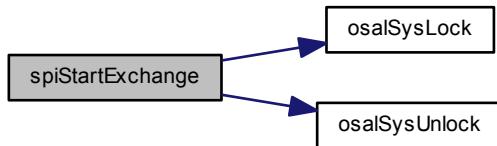
**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive buffer

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.34.6.9 void spiStartSend ( SPIDriver \* *spip*, size\_t *n*, const void \* *txbuf* )**

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

**Precondition**

A slave must have been selected using [spiSelect\(\)](#) or [spiSelectI\(\)](#).

**Postcondition**

At the end of the operation the configured callback is invoked.

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

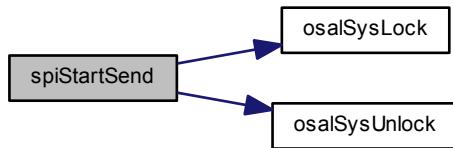
**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.34.6.10 void spiStartReceive ( *SPIDriver* \* *spip*, *size\_t* *n*, *void* \* *rxbuf* )

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

##### Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

##### Postcondition

At the end of the operation the configured callback is invoked.

##### Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

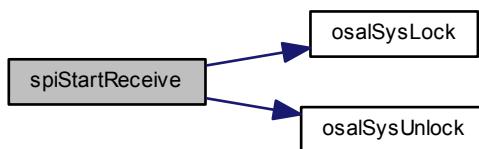
##### Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to receive
out	<i>rxbuf</i>	the pointer to the receive buffer

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.34.6.11 void spilgnore ( *SPIDriver \* spip*, *size\_t n* )

Ignores data on the SPI bus.

This synchronous function performs the transmission of a series of idle words on the SPI bus and ignores the received data.

##### Precondition

In order to use this function the option SPI\_USE\_WAIT must be enabled.

In order to use this function the driver must have been configured without callbacks (end\_cb = NULL).

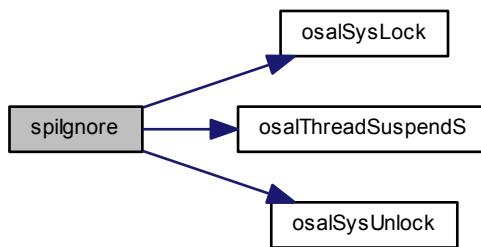
##### Parameters

in	<i>spip</i>	pointer to the <i>SPIDriver</i> object
in	<i>n</i>	number of words to be ignored

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.34.6.12 void spiExchange ( *SPIDriver \* spip*, *size\_t n*, *const void \* txbuf*, *void \* rxbuf* )

Exchanges data on the SPI bus.

This synchronous function performs a simultaneous transmit/receive operation.

##### Precondition

In order to use this function the option SPI\_USE\_WAIT must be enabled.

In order to use this function the driver must have been configured without callbacks (end\_cb = NULL).

##### Note

The buffers are organized as uint8\_t arrays for data sizes below or equal to 8 bits else it is organized as uint16\_t arrays.

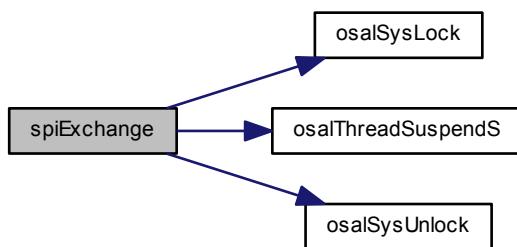
**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive buffer

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.34.6.13 void spiSend ( `SPIDriver` \* *spip*, `size_t` *n*, `const void` \* *txbuf* )**

Sends data over the SPI bus.

This synchronous function performs a transmit operation.

**Precondition**

In order to use this function the option `SPI_USE_WAIT` must be enabled.

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

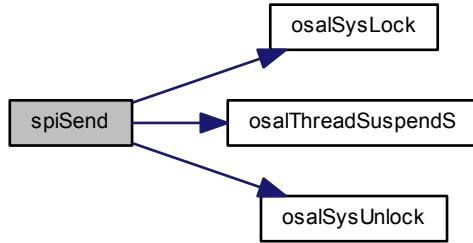
**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.34.6.14 void spiReceive ( `SPI_driver * spip`, `size_t n`, `void * rdbuf` )

Receives data from the SPI bus.

This synchronous function performs a receive operation.

#### Precondition

In order to use this function the option `SPI_USE_WAIT` must be enabled.

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

#### Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

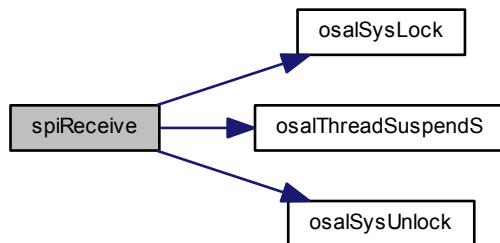
#### Parameters

<code>in</code>	<code>spip</code>	pointer to the <code>SPI_driver</code> object
<code>in</code>	<code>n</code>	number of words to receive
<code>out</code>	<code>rdbuf</code>	the pointer to the receive buffer

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.34.6.15 void spiAcquireBus ( SPIDriver \* spip )

Gains exclusive access to the SPI bus.

This function tries to gain ownership to the SPI bus, if the bus is already being used then the invoking thread is queued.

##### Precondition

In order to use this function the option SPI\_USE\_MUTUAL\_EXCLUSION must be enabled.

##### Parameters

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.34.6.16 void spiReleaseBus ( SPIDriver \* spip )

Releases exclusive access to the SPI bus.

##### Precondition

In order to use this function the option SPI\_USE\_MUTUAL\_EXCLUSION must be enabled.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.34.6.17 void spi\_lld\_init( void )**

Low level SPI driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.34.6.18 void spi\_lld\_start( SPIDriver \* spip )**

Configures and activates the SPI peripheral.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.34.6.19 void spi\_lld\_stop( SPIDriver \* spip )**

Deactivates the SPI peripheral.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.34.6.20 void spi\_lld\_select ( [SPIDriver](#) \* *spip* )**

Asserts the slave select signal and prepares for transfers.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.34.6.21 void spi\_lld\_unselect ( [SPIDriver](#) \* *spip* )**

Deasserts the slave select signal.

The previously selected peripheral is unselected.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.34.6.22 void spi\_lld\_ignore ( [SPIDriver](#) \* *spip*, size\_t *n* )**

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
in	<i>n</i>	number of words to be ignored

**Function Class:**

Not an API, this function is for internal use only.

**7.34.6.23 void spi\_lld\_exchange ( [SPIDriver](#) \* *spip*, size\_t *n*, const void \* *txbuf*, void \* *rxbuf* )**

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Note**

The buffers are organized as uint8\_t arrays for data sizes below or equal to 8 bits else it is organized as uint16\_t arrays.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive buffer

**Function Class:**

Not an API, this function is for internal use only.

**7.34.6.24 void spi\_lld\_send ( SPIDriver \* *spip*, size\_t *n*, const void \* *txbuf* )**

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Note**

The buffers are organized as uint8\_t arrays for data sizes below or equal to 8 bits else it is organized as uint16\_t arrays.

**Parameters**

in	<i>spip</i>	pointer to the <a href="#">SPIDriver</a> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

**Function Class:**

Not an API, this function is for internal use only.

**7.34.6.25 void spi\_lld\_receive ( SPIDriver \* *spip*, size\_t *n*, void \* *rxbuf* )**

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Note**

The buffers are organized as uint8\_t arrays for data sizes below or equal to 8 bits else it is organized as uint16\_t arrays.

**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to receive
out	<i>rdbuf</i>	the pointer to the receive buffer

**Function Class:**

Not an API, this function is for internal use only.

**7.34.6.26 uint16\_t spi\_lld\_polled\_exchange ( `SPIDriver` \* *spip*, `uint16_t` *frame* )**

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>frame</i>	the data frame to send over the SPI bus

**Returns**

The received data frame from the SPI bus.

**7.34.7 Variable Documentation****7.34.7.1 SPIDriver SPID1**

SPI1 driver identifier.

## 7.35 ST Driver

Generic System Tick Driver.

### 7.35.1 Detailed Description

Generic System Tick Driver.

This module implements a system tick timer in order to support the underlying operating system.

#### Macro Functions

- `#define stGetCounter() st_lld_get_counter()`  
*Returns the time counter value.*
- `#define stIsAlarmActive() st_lld_is_alarm_active()`  
*Determines if the alarm is active.*

#### Functions

- `void stInit (void)`  
*ST Driver initialization.*
- `void stStartAlarm (systime_t abstime)`  
*Starts the alarm.*
- `void stStopAlarm (void)`  
*Stops the alarm interrupt.*
- `void stSetAlarm (systime_t abstime)`  
*Sets the alarm time.*
- `systime_t stGetAlarm (void)`  
*Returns the current alarm time.*
- `void st_lld_init (void)`  
*Low level ST driver initialization.*
- `static systime_t st_lld_get_counter (void)`  
*Returns the time counter value.*
- `static void st_lld_start_alarm (systime_t abstime)`  
*Starts the alarm.*
- `static void st_lld_stop_alarm (void)`  
*Stops the alarm interrupt.*
- `static void st_lld_set_alarm (systime_t abstime)`  
*Sets the alarm time.*
- `static systime_t st_lld_get_alarm (void)`  
*Returns the current alarm time.*
- `static bool st_lld_is_alarm_active (void)`  
*Determines if the alarm is active.*

### 7.35.2 Macro Definition Documentation

#### 7.35.2.1 #define stGetCounter( ) st\_lld\_get\_counter()

Returns the time counter value.

**Note**

This functionality is only available in free running mode, the behaviour in periodic mode is undefined.

**Returns**

The counter value.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.35.2.2 #define stIsAlarmActive( ) st\_lld\_is\_alarm\_active()**

Determines if the alarm is active.

**Returns**

The alarm status.

**Return values**

<i>false</i>	if the alarm is not active.
<i>true</i>	is the alarm is active

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.35.3 Function Documentation****7.35.3.1 void stlInit( void )**

ST Driver initialization.

**Note**

This function is implicitly invoked by [halInit \(\)](#), there is no need to explicitly initialize the driver.

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.35.3.2 void stStartAlarm ( *systime\_t abstime* )

Starts the alarm.

#### Note

Makes sure that no spurious alarms are triggered after this call.

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

#### Parameters

in	<i>abstime</i>	the time to be set for the first alarm
----	----------------	--

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.35.3.3 void stStopAlarm ( void )

Stops the alarm interrupt.

#### Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.35.3.4 void stSetAlarm ( *systime\_t abstime* )

Sets the alarm time.

**Note**

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

**Parameters**

in	<i>abstime</i>	the time to be set for the next alarm
----	----------------	---------------------------------------

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.35.3.5 systime\_t stGetAlarm( void )**

Returns the current alarm time.

**Note**

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

**Returns**

The currently set alarm time.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.35.3.6 void st\_lld\_init( void )**

Low level ST driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

**7.35.3.7 static systime\_t st\_lld\_get\_counter( void ) [inline], [static]**

Returns the time counter value.

**Returns**

The counter value.

**Function Class:**

Not an API, this function is for internal use only.

**7.35.3.8 static void st\_lld\_start\_alarm( systime\_t abstime ) [inline], [static]**

Starts the alarm.

**Note**

Makes sure that no spurious alarms are triggered after this call.

**Parameters**

in	abstime	the time to be set for the first alarm
----	---------	--

**Function Class:**

Not an API, this function is for internal use only.

**7.35.3.9 static void st\_lld\_stop\_alarm( void ) [inline], [static]**

Stops the alarm interrupt.

**Function Class:**

Not an API, this function is for internal use only.

**7.35.3.10 static void st\_lld\_set\_alarm( systime\_t abstime ) [inline], [static]**

Sets the alarm time.

**Parameters**

in	abstime	the time to be set for the next alarm
----	---------	---------------------------------------

**Function Class:**

Not an API, this function is for internal use only.

**7.35.3.11 static systime\_t st\_lld\_get\_alarm( void ) [inline], [static]**

Returns the current alarm time.

**Returns**

The currently set alarm time.

**Function Class:**

Not an API, this function is for internal use only.

**7.35.3.12 static bool st\_lld\_is\_alarm\_active( void ) [inline], [static]**

Determines if the alarm is active.

**Returns**

The alarm status.

**Return values**

<i>false</i>	if the alarm is not active.
<i>true</i>	is the alarm is active

**Function Class:**

Not an API, this function is for internal use only.

## 7.36 UART Driver

Generic UART Driver.

### 7.36.1 Detailed Description

Generic UART Driver.

This driver abstracts a generic UART (Universal Asynchronous Receiver Transmitter) peripheral, the API is designed to be:

- Unbuffered and copy-less, transfers are always directly performed from/to the application-level buffers without extra copy operations.
- Asynchronous, the API is always non blocking.
- Callbacks capable, operations completion and other events are notified using callbacks.

Special hardware features like deep hardware buffers, DMA transfers are hidden to the user but fully supportable by the low level implementations.

This driver model is best used where communication events are meant to drive an higher level state machine, as example:

- RS485 drivers.
- Multipoint network drivers.
- Serial protocol decoders.

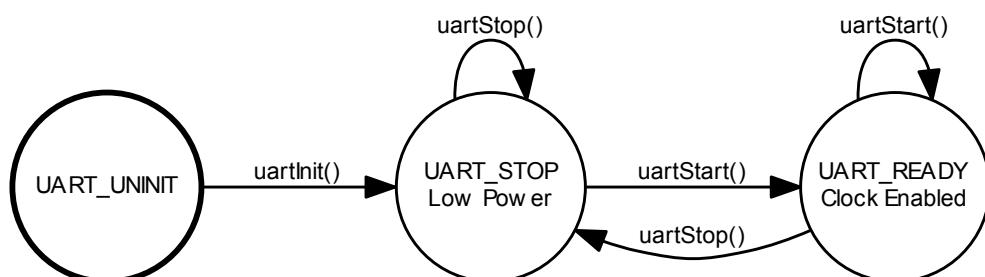
If your application requires a synchronous buffered driver then the [Serial Driver](#) should be used instead.

#### Precondition

In order to use the UART driver the `HAL_USE_UART` option must be enabled in [`halconf.h`](#).

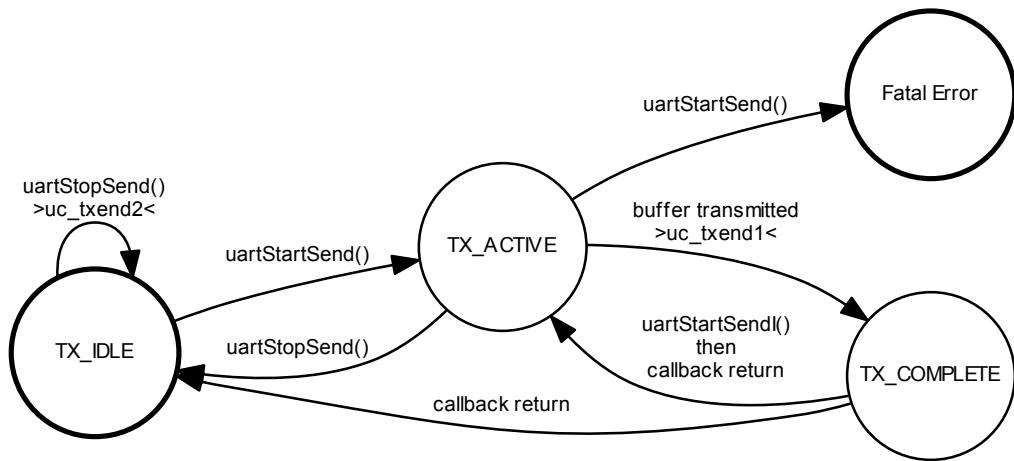
### 7.36.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



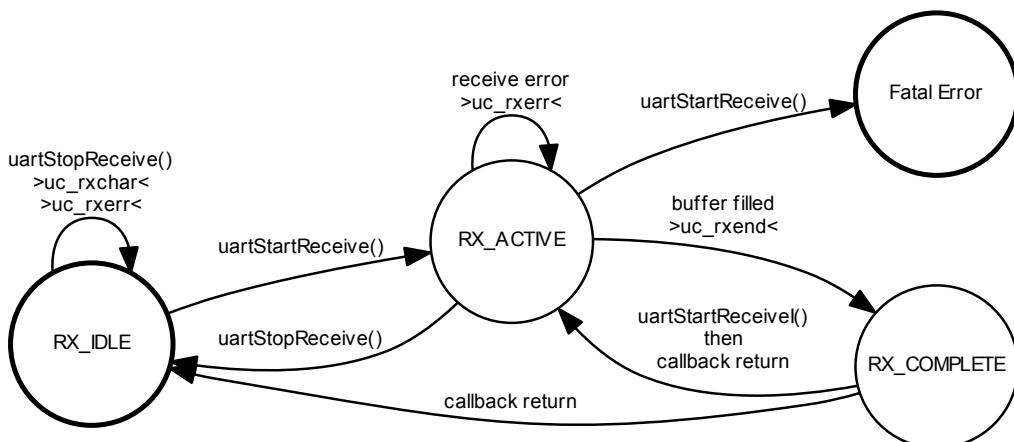
### 7.36.2.1 Transmitter sub State Machine

The follow diagram describes the transmitter state machine, this diagram is valid while the driver is in the `UART←T_READY` state. This state machine is automatically reset to the `TX_IDLE` state each time the driver enters the `UART_READY` state.



### 7.36.2.2 Receiver sub State Machine

The follow diagram describes the receiver state machine, this diagram is valid while the driver is in the `UART←READY` state. This state machine is automatically reset to the `RX_IDLE` state each time the driver enters the `UART_READY` state.



## UART status flags

- `#define UART_NO_ERROR 0`  
*No pending conditions.*
- `#define UART_PARITY_ERROR 4`  
*Parity error happened.*
- `#define UART_FRAMING_ERROR 8`  
*Framing error happened.*
- `#define UART_OVERRUN_ERROR 16`  
*Overflow happened.*
- `#define UART_NOISE_ERROR 32`  
*Noise on the line.*
- `#define UART_BREAK_DETECTED 64`  
*Break detected.*

## UART configuration options

- `#define UART_USE_WAIT FALSE`  
*Enables synchronous APIs.*
- `#define UART_USE_MUTUAL_EXCLUSION FALSE`  
*Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.*

## Low level driver helper macros

- `#define _uart_wakeup_tx1_isr(uartp)`  
*Wakes up the waiting thread in case of early TX complete.*
- `#define _uart_wakeup_tx2_isr(uartp)`  
*Wakes up the waiting thread in case of late TX complete.*
- `#define _uart_wakeup_rx_complete_isr(uartp)`  
*Wakes up the waiting thread in case of RX complete.*
- `#define _uart_wakeup_rx_error_isr(uartp)`  
*Wakes up the waiting thread in case of RX error.*
- `#define _uart_tx1_isr_code(uartp)`  
*Common ISR code for early TX.*
- `#define _uart_tx2_isr_code(uartp)`  
*Common ISR code for late TX.*
- `#define _uart_rx_complete_isr_code(uartp)`  
*Common ISR code for RX complete.*
- `#define _uart_rx_error_isr_code(uartp, errors)`  
*Common ISR code for RX error.*
- `#define _uart_rx_idle_code(uartp)`  
*Common ISR code for RX on idle.*

## PLATFORM configuration options

- `#define PLATFORM_UART_USE_UART1 FALSE`  
*UART driver enable switch.*

## Typedefs

- **typedef uint32\_t uartflags\_t**  
*UART driver condition flags type.*
- **typedef struct UARTDriver UARTDriver**  
*Type of structure representing an UART driver.*
- **typedef void(\* uartcb\_t) (UARTDriver \*uartp)**  
*Generic UART notification callback type.*
- **typedef void(\* uartccb\_t) (UARTDriver \*uartp, uint16\_t c)**  
*Character received UART notification callback type.*
- **typedef void(\* uarceb\_t) (UARTDriver \*uartp, uartflags\_t e)**  
*Receive error UART notification callback type.*

## Data Structures

- **struct UARTConfig**  
*Driver configuration structure.*
- **struct UARTDriver**  
*Structure representing an UART driver.*

## Functions

- **void uartInit (void)**  
*UART Driver initialization.*
- **void uartObjectInit (UARTDriver \*uartp)**  
*Initializes the standard part of a `UARTDriver` structure.*
- **void uartStart (UARTDriver \*uartp, const UARTConfig \*config)**  
*Configures and activates the UART peripheral.*
- **void uartStop (UARTDriver \*uartp)**  
*Deactivates the UART peripheral.*
- **void uartStartSend (UARTDriver \*uartp, size\_t n, const void \*txbuf)**  
*Starts a transmission on the UART peripheral.*
- **void uartStartSendl (UARTDriver \*uartp, size\_t n, const void \*txbuf)**  
*Starts a transmission on the UART peripheral.*
- **size\_t uartStopSend (UARTDriver \*uartp)**  
*Stops any ongoing transmission.*
- **size\_t uartStopSendl (UARTDriver \*uartp)**  
*Stops any ongoing transmission.*
- **void uartStartReceive (UARTDriver \*uartp, size\_t n, void \*rdbuf)**  
*Starts a receive operation on the UART peripheral.*
- **void uartStartReceive1 (UARTDriver \*uartp, size\_t n, void \*rdbuf)**  
*Starts a receive operation on the UART peripheral.*
- **size\_t uartStopReceive (UARTDriver \*uartp)**  
*Stops any ongoing receive operation.*
- **size\_t uartStopReceive1 (UARTDriver \*uartp)**  
*Stops any ongoing receive operation.*
- **msg\_t uartSendTimeout (UARTDriver \*uartp, size\_t \*np, const void \*txbuf, systime\_t timeout)**  
*Performs a transmission on the UART peripheral.*
- **msg\_t uartSendFullTimeout (UARTDriver \*uartp, size\_t \*np, const void \*txbuf, systime\_t timeout)**  
*Performs a transmission on the UART peripheral.*

- `msg_t uartReceiveTimeout (UARTDriver *uartp, size_t *np, void *rdbuf, systime_t timeout)`  
*Performs a receive operation on the UART peripheral.*
- `void uartAcquireBus (UARTDriver *uartp)`  
*Gains exclusive access to the UART bus.*
- `void uartReleaseBus (UARTDriver *uartp)`  
*Releases exclusive access to the UART bus.*
- `void uart_lld_init (void)`  
*Low level UART driver initialization.*
- `void uart_lld_start (UARTDriver *uartp)`  
*Configures and activates the UART peripheral.*
- `void uart_lld_stop (UARTDriver *uartp)`  
*Deactivates the UART peripheral.*
- `void uart_lld_start_send (UARTDriver *uartp, size_t n, const void *txbuf)`  
*Starts a transmission on the UART peripheral.*
- `size_t uart_lld_stop_send (UARTDriver *uartp)`  
*Stops any ongoing transmission.*
- `void uart_lld_start_receive (UARTDriver *uartp, size_t n, void *rdbuf)`  
*Starts a receive operation on the UART peripheral.*
- `size_t uart_lld_stop_receive (UARTDriver *uartp)`  
*Stops any ongoing receive operation.*

## Enumerations

- enum `uartstate_t { UART_UNINIT = 0, UART_STOP = 1, UART_READY = 2 }`  
*Driver state machine possible states.*
- enum `uartxstate_t { UART_TX_IDLE = 0, UART_TX_ACTIVE = 1, UART_TX_COMPLETE = 2 }`  
*Transmitter state machine states.*
- enum `uartrxstate_t { UART_RX_IDLE = 0, UART_RX_ACTIVE = 1, UART_RX_COMPLETE = 2 }`  
*Receiver state machine states.*

## Variables

- `UARTDriver UARTD1`  
*UART1 driver identifier.*

### 7.36.3 Macro Definition Documentation

#### 7.36.3.1 #define UART\_NO\_ERROR 0

No pending conditions.

#### 7.36.3.2 #define UART\_PARITY\_ERROR 4

Parity error happened.

#### 7.36.3.3 #define UART\_FRAMING\_ERROR 8

Framing error happened.

#### 7.36.3.4 #define UART\_OVERRUN\_ERROR 16

Overflow happened.

#### 7.36.3.5 #define UART\_NOISE\_ERROR 32

Noise on the line.

#### 7.36.3.6 #define UART\_BREAK\_DETECTED 64

Break detected.

#### 7.36.3.7 #define UART\_USE\_WAIT FALSE

Enables synchronous APIs.

##### Note

Disabling this option saves both code and data space.

#### 7.36.3.8 #define UART\_USE\_MUTUAL\_EXCLUSION FALSE

Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

##### Note

Disabling this option saves both code and data space.

#### 7.36.3.9 #define \_uart\_wakeup\_tx1\_isr( *uartp* )

##### Value:

```
{
    if ((uartp)->early == true) {
        osalSysLockFromISR();
        \
        osalThreadResumeI(&(uartp)->threadtx, MSG_OK);
        \
        osalSysUnlockFromISR();
    }
}
```

Wakes up the waiting thread in case of early TX complete.

##### Parameters

in	<i>uartp</i>	pointer to the <code>UARTDriver</code> object
----	--------------	---

##### Function Class:

Not an API, this function is for internal use only.

7.36.3.10 #define \_uart\_wakeup\_tx2\_isr( *uartp* )**Value:**

```
{
    if ((uartp)->early == false) {
        \
        osalSysLockFromISR();
        \
        osalThreadResumeI(&(uartp)->threadtx, MSG_OK);
        \
        osalSysUnlockFromISR();
    }
}
```

Wakes up the waiting thread in case of late TX complete.

**Parameters**

in	<i>uartp</i>	pointer to the <a href="#">UARTDriver</a> object
----	--------------	--

**Function Class:**

Not an API, this function is for internal use only.

7.36.3.11 #define \_uart\_wakeup\_rx\_complete\_isr( *uartp* )**Value:**

```
{
    osalSysLockFromISR();
    \
    osalThreadResumeI(&(uartp)->threadrx, MSG_OK);
    \
    osalSysUnlockFromISR();
}
```

Wakes up the waiting thread in case of RX complete.

**Parameters**

in	<i>uartp</i>	pointer to the <a href="#">UARTDriver</a> object
----	--------------	--

**Function Class:**

Not an API, this function is for internal use only.

7.36.3.12 #define \_uart\_wakeup\_rx\_error\_isr( *uartp* )**Value:**

```
{
    osalSysLockFromISR();
    \
    osalThreadResumeI(&(uartp)->threadrx, MSG_RESET);
    \
    osalSysUnlockFromISR();
}
```

Wakes up the waiting thread in case of RX error.

**Parameters**

in	<i>uartp</i>	pointer to the <a href="#">UARTDriver</a> object
----	--------------	--

**Function Class:**

Not an API, this function is for internal use only.

**7.36.3.13 #define \_uart\_tx1\_isr\_code( *uartp* )****Value:**

```
{
    (uartp)->txstate = UART\_TX\_COMPLETE;
    if ((uartp)->config->txend1_cb != NULL) {
        (uartp)->config->txend1_cb(uartp);
    }
    if ((uartp)->txstate == UART\_TX\_COMPLETE) {
        (uartp)->txstate = UART\_TX\_IDLE;
    }
    \_uart\_wakeup\_tx1\_isr(uartp);
}
```



Common ISR code for early TX.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>uartp</i>	pointer to the <a href="#">UARTDriver</a> object
----	--------------	--

**Function Class:**

Not an API, this function is for internal use only.

**7.36.3.14 #define \_uart\_tx2\_isr\_code( *uartp* )****Value:**

```
{
    if ((uartp)->config->txend2_cb != NULL) {
        (uartp)->config->txend2_cb(uartp);
    }
    \_uart\_wakeup\_tx2\_isr(uartp);
}
```



Common ISR code for late TX.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>uartp</i>	pointer to the <a href="#">UARTDriver</a> object
----	--------------	--

**Function Class:**

Not an API, this function is for internal use only.

**7.36.3.15 #define \_uart\_rx\_complete\_isr\_code( *uartp* )****Value:**

```
{
    (uartp)->rxstate = UART\_RX\_COMPLETE;
    if ((uartp)->config->rxend_cb != NULL) {
        (uartp)->config->rxend_cb(uartp);
    }
    if ((uartp)->rxstate == UART\_RX\_COMPLETE) {
        (uartp)->rxstate = UART\_RX\_IDLE;
        uart\_enter\_rx\_idle\_loop(uartp);
    }
    \_uart\_wakeup\_rx\_complete\_isr(uartp);
}
```

Common ISR code for RX complete.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>uartp</i>	pointer to the <a href="#">UARTDriver</a> object
----	--------------	--

**Function Class:**

Not an API, this function is for internal use only.

**7.36.3.16 #define \_uart\_rx\_error\_isr\_code( *uartp, errors* )****Value:**

```
{
    if ((uartp)->config->rxerr_cb != NULL) {
        (uartp)->config->rxerr_cb(uartp, errors);
    }
    \_uart\_wakeup\_rx\_error\_isr(uartp);
}
```

Common ISR code for RX error.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>uartp</i>	pointer to the <a href="#">UARTDriver</a> object
in	<i>errors</i>	mask of errors to be reported

**Function Class:**

Not an API, this function is for internal use only.

**7.36.3.17 #define \_uart\_rx\_idle\_code( *uartp* )****Value:**

```
{
    if ((uartp)->config->rxchar_cb != NULL)
        (uartp)->config->rxchar_cb(uartp, (uartp)->rxbuf);
}
```

Common ISR code for RX on idle.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

in	<i>uartp</i>	pointer to the <a href="#">UARTDriver</a> object
----	--------------	--

**Function Class:**

Not an API, this function is for internal use only.

**7.36.3.18 #define PLATFORM\_UART\_USE\_UART1 FALSE**

UART driver enable switch.

If set to TRUE the support for UART1 is included.

**Note**

The default is FALSE.

### 7.36.4 Typedef Documentation

#### 7.36.4.1 `typedef uint32_t uartflags_t`

UART driver condition flags type.

#### 7.36.4.2 `typedef struct UARTDriver UARTDriver`

Type of structure representing an UART driver.

#### 7.36.4.3 `typedef void(* uartcb_t) (UARTDriver *uartp)`

Generic UART notification callback type.

##### Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
----	--------------------	---

#### 7.36.4.4 `typedef void(* uartccb_t) (UARTDriver *uartp, uint16_t c)`

Character received UART notification callback type.

##### Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object triggering the callback
in	<code>c</code>	received character

#### 7.36.4.5 `typedef void(* uarteccb_t) (UARTDriver *uartp, uartflags_t e)`

Receive error UART notification callback type.

##### Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object triggering the callback
in	<code>e</code>	receive error mask

### 7.36.5 Enumeration Type Documentation

#### 7.36.5.1 `enum uartstate_t`

Driver state machine possible states.

##### Enumerator

**`UART_UNINIT`** Not initialized.

**`UART_STOP`** Stopped.

**`UART_READY`** Ready.

#### 7.36.5.2 `enum uarttxstate_t`

Transmitter state machine states.

##### Enumerator

**`UART_TX_IDLE`** Not transmitting.

**UART\_TX\_ACTIVE** Transmitting.

**UART\_TX\_COMPLETE** Buffer complete.

#### 7.36.5.3 enum uartrxstate\_t

Receiver state machine states.

Enumerator

**UART\_RX\_IDLE** Not receiving.

**UART\_RX\_ACTIVE** Receiving.

**UART\_RX\_COMPLETE** Buffer complete.

### 7.36.6 Function Documentation

#### 7.36.6.1 void uartInit( void )

UART Driver initialization.

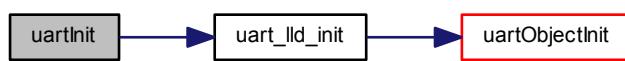
Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.36.6.2 void uartObjectInit( UARTDriver \* uartp )

Initializes the standard part of a [UARTDriver](#) structure.

Parameters

out	<i>uartp</i>	pointer to the <a href="#">UARTDriver</a> object
-----	--------------	--

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.36.6.3 void uartStart ( **UARTDriver** \* *uartp*, const **UARTConfig** \* *config* )

Configures and activates the UART peripheral.

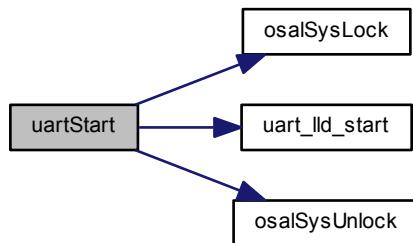
##### Parameters

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
in	<i>config</i>	pointer to the <b>UARTConfig</b> object

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.36.6.4 void uartStop ( **UARTDriver** \* *uartp* )

Deactivates the UART peripheral.

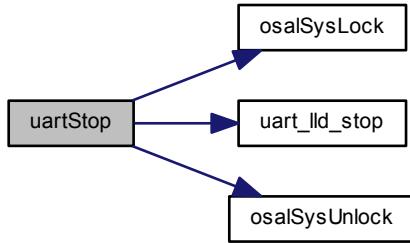
##### Parameters

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
----	--------------	---

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.36.6.5 void uartStartSend ( `UARTDriver *uartp`, `size_t n`, `const void *txbuf` )

Starts a transmission on the UART peripheral.

##### Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

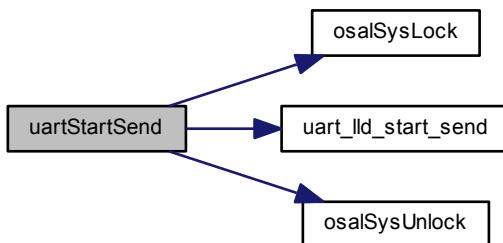
##### Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
in	<code>n</code>	number of data frames to send
in	<code>txbuf</code>	the pointer to the transmit buffer

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.36.6.6 void uartStartSend( **UARTDriver** \* *uartp*, **size\_t** *n*, const void \* *txbuf* )

Starts a transmission on the UART peripheral.

#### Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

This function has to be invoked from a lock zone.

#### Parameters

in	<i>uartp</i>	pointer to the <code>UARTDriver</code> object
in	<i>n</i>	number of data frames to send
in	<i>txbuf</i>	the pointer to the transmit buffer

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.36.6.7 **size\_t** uartStopSend( **UARTDriver** \* *uartp* )

Stops any ongoing transmission.

#### Note

Stopping a transmission also suppresses the transmission callbacks.

#### Parameters

in	<i>uartp</i>	pointer to the <code>UARTDriver</code> object
----	--------------	---

#### Returns

The number of data frames not transmitted by the stopped transmit operation.

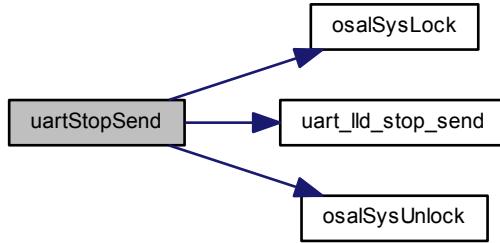
#### Return values

0	There was no transmit operation in progress.
---	--

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.36.6.8 size\_t uartStopSendl ( [UARTDriver](#) \* *uartp* )

Stops any ongoing transmission.

#### Note

Stopping a transmission also suppresses the transmission callbacks.  
This function has to be invoked from a lock zone.

#### Parameters

in	<i>uartp</i>	pointer to the <a href="#">UARTDriver</a> object
----	--------------	--

#### Returns

The number of data frames not transmitted by the stopped transmit operation.

#### Return values

0	There was no transmit operation in progress.
---	--

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.36.6.9 void uartStartReceive ( **UARTDriver** \* *uartp*, **size\_t** *n*, **void** \* *rxbuf* )

Starts a receive operation on the UART peripheral.

#### Note

The buffers are organized as **uint8\_t** arrays for data sizes below or equal to 8 bits else it is organized as **uint16\_t** arrays.

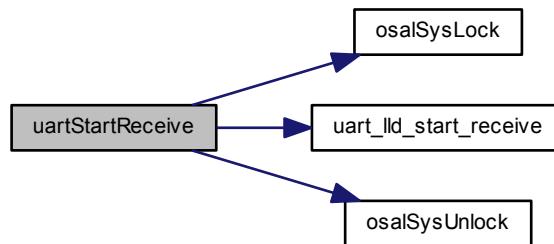
#### Parameters

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
in	<i>n</i>	number of data frames to receive
in	<i>rxbuf</i>	the pointer to the receive buffer

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.36.6.10 void uartStartReceive1 ( **UARTDriver** \* *uartp*, **size\_t** *n*, **void** \* *rxbuf* )

Starts a receive operation on the UART peripheral.

#### Note

The buffers are organized as **uint8\_t** arrays for data sizes below or equal to 8 bits else it is organized as **uint16\_t** arrays.

This function has to be invoked from a lock zone.

#### Parameters

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
in	<i>n</i>	number of data frames to receive
out	<i>rxbuf</i>	the pointer to the receive buffer

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.36.6.11 size\_t uartStopReceive ( **UARTDriver** \* *uartp* )

Stops any ongoing receive operation.

#### Note

Stopping a receive operation also suppresses the receive callbacks.

#### Parameters

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
----	--------------	---

#### Returns

The number of data frames not received by the stopped receive operation.

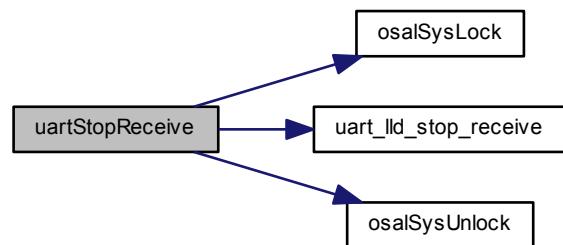
#### Return values

0	There was no receive operation in progress.
---	---

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.36.6.12 `size_t uartStopReceive( UARTDriver *uartp )`

Stops any ongoing receive operation.

#### Note

Stopping a receive operation also suppresses the receive callbacks.  
This function has to be invoked from a lock zone.

#### Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
----	--------------------	---

#### Returns

The number of data frames not received by the stopped receive operation.

#### Return values

0	There was no receive operation in progress.
---	---

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.36.6.13 `msg_t uartSendTimeout( UARTDriver *uartp, size_t *np, const void *txbuf, systime_t timeout )`

Performs a transmission on the UART peripheral.

#### Note

The function returns when the specified number of frames have been sent to the UART or on timeout.  
The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

#### Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
in, out	<code>np</code>	number of data frames to transmit, on exit the number of frames actually transmitted

in	<i>txbuf</i>	the pointer to the transmit buffer
in	<i>timeout</i>	operation timeout

**Returns**

The operation status.

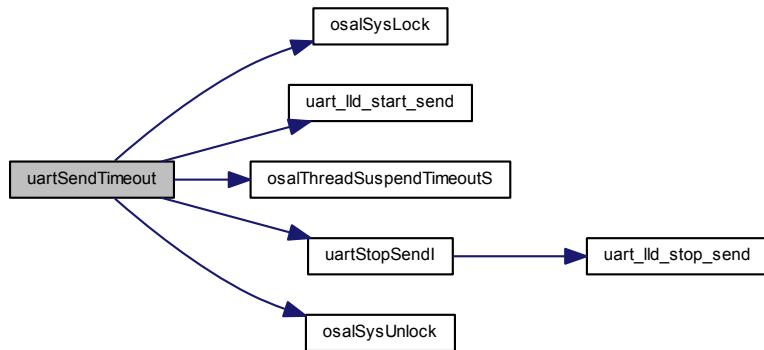
**Return values**

<i>MSG_OK</i>	if the operation completed successfully.
<i>MSG_TIMEOUT</i>	if the operation timed out.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



### 7.36.6.14 `msg_t uartSendFullTimeout( UARTDriver * uartp, size_t * np, const void * txbuf, systime_t timeout )`

Performs a transmission on the UART peripheral.

**Note**

The function returns when the specified number of frames have been physically transmitted or on timeout. The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

**Parameters**

in	<i>uartp</i>	pointer to the <code>UARTDriver</code> object
in, out	<i>np</i>	number of data frames to transmit, on exit the number of frames actually transmitted

in	<i>txbuf</i>	the pointer to the transmit buffer
in	<i>timeout</i>	operation timeout

**Returns**

The operation status.

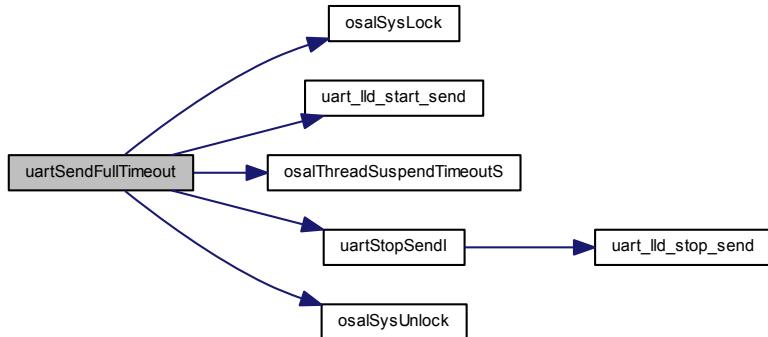
**Return values**

<i>MSG_OK</i>	if the operation completed successfully.
<i>MSG_TIMEOUT</i>	if the operation timed out.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.36.6.15 `msg_t uartReceiveTimeout ( UARTDriver * uartp, size_t * np, void * rxbuf, systime_t timeout )`**

Performs a receive operation on the UART peripheral.

**Note**

The function returns when the specified number of frames have been received or on error/timeout.  
The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

**Parameters**

in	<i>uartp</i>	pointer to the <code>UARTDriver</code> object
in, out	<i>np</i>	number of data frames to receive, on exit the number of frames actually received
in	<i>rxbuf</i>	the pointer to the receive buffer

in	<i>timeout</i>	operation timeout
----	----------------	-------------------

**Returns**

The operation status.

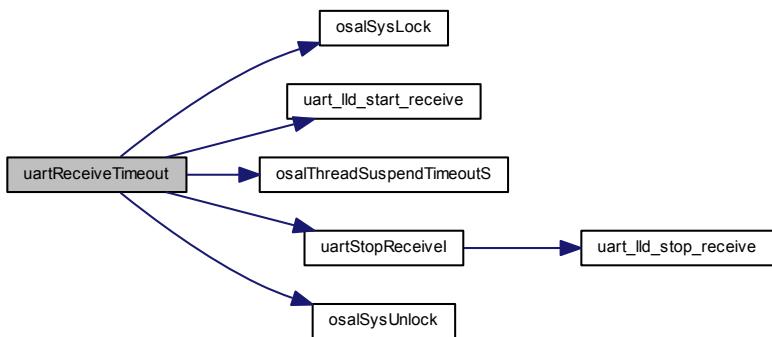
**Return values**

<i>MSG_OK</i>	if the operation completed successfully.
<i>MSG_TIMEOUT</i>	if the operation timed out.
<i>MSG_RESET</i>	in case of a receive error.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.36.6.16 void uartAcquireBus ( **UARTDriver** \* *uartp* )**

Gains exclusive access to the UART bus.

This function tries to gain ownership to the UART bus, if the bus is already being used then the invoking thread is queued.

**Precondition**

In order to use this function the option **UART\_USE\_MUTUAL\_EXCLUSION** must be enabled.

**Parameters**

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
----	--------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.36.6.17 void uartReleaseBus ( [UARTDriver \\* uartp](#) )

Releases exclusive access to the UART bus.

##### Precondition

In order to use this function the option `UART_USE_MUTUAL_EXCLUSION` must be enabled.

##### Parameters

in	<code>uartp</code>	pointer to the <a href="#">UARTDriver</a> object
----	--------------------	--

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.36.6.18 void uart\_lld\_init ( void )

Low level UART driver initialization.

##### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.36.6.19 void uart\_lld\_start ( **UARTDriver** \* *uartp* )

Configures and activates the UART peripheral.

##### Parameters

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
----	--------------	---

##### Function Class:

Not an API, this function is for internal use only.

#### 7.36.6.20 void uart\_lld\_stop ( **UARTDriver** \* *uartp* )

Deactivates the UART peripheral.

##### Parameters

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
----	--------------	---

##### Function Class:

Not an API, this function is for internal use only.

#### 7.36.6.21 void uart\_lld\_start\_send ( **UARTDriver** \* *uartp*, **size\_t** *n*, const void \* *txbuf* )

Starts a transmission on the UART peripheral.

##### Note

The buffers are organized as **uint8\_t** arrays for data sizes below or equal to 8 bits else it is organized as **uint16\_t** arrays.

##### Parameters

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
in	<i>n</i>	number of data frames to send
in	<i>txbuf</i>	the pointer to the transmit buffer

##### Function Class:

Not an API, this function is for internal use only.

**7.36.6.22 size\_t uart\_lld\_stop\_send ( **UARTDriver** \* *uartp* )**

Stops any ongoing transmission.

**Note**

Stopping a transmission also suppresses the transmission callbacks.

**Parameters**

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
----	--------------	---

**Returns**

The number of data frames not transmitted by the stopped transmit operation.

**Function Class:**

Not an API, this function is for internal use only.

**7.36.6.23 void uart\_lld\_start\_receive ( **UARTDriver** \* *uartp*, size\_t *n*, void \* *rxbuf* )**

Starts a receive operation on the UART peripheral.

**Note**

The buffers are organized as uint8\_t arrays for data sizes below or equal to 8 bits else it is organized as uint16\_t arrays.

**Parameters**

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
in	<i>n</i>	number of data frames to send
out	<i>rxbuf</i>	the pointer to the receive buffer

**Function Class:**

Not an API, this function is for internal use only.

**7.36.6.24 size\_t uart\_lld\_stop\_receive ( **UARTDriver** \* *uartp* )**

Stops any ongoing receive operation.

**Note**

Stopping a receive operation also suppresses the receive callbacks.

**Parameters**

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
----	--------------	---

**Returns**

The number of data frames not received by the stopped receive operation.

**Function Class:**

Not an API, this function is for internal use only.

## 7.36.7 Variable Documentation

### 7.36.7.1 **UARTDriver** **UARTD1**

UART1 driver identifier.

## 7.37 USB Driver

Generic USB Driver.

### 7.37.1 Detailed Description

Generic USB Driver.

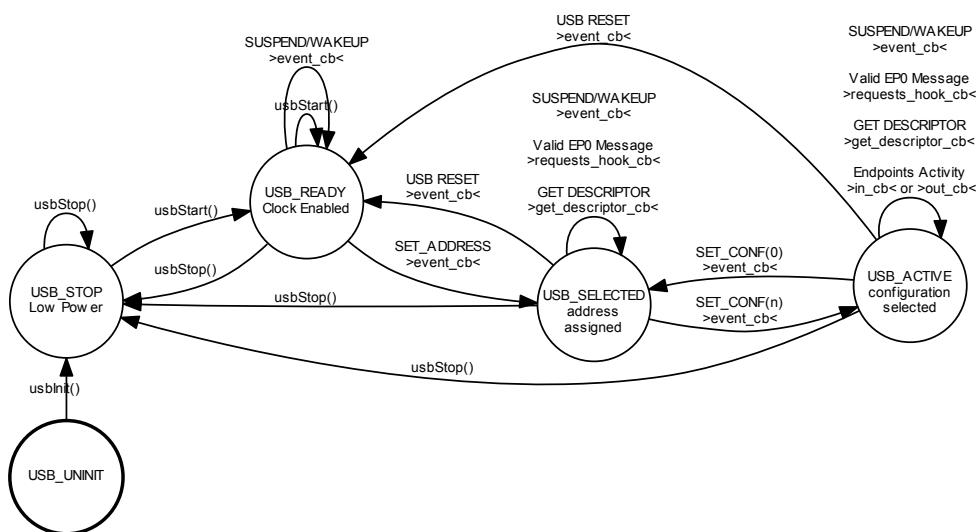
This module implements a generic USB (Universal Serial Bus) driver supporting device-mode operations.

#### Precondition

In order to use the USB driver the `HAL_USE_USB` option must be enabled in `halconf.h`.

### 7.37.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



### 7.37.3 USB Operations

The USB driver is quite complex and USB is complex in itself, it is recommended to study the USB specification before trying to use the driver.

#### 7.37.3.1 USB Implementation

The USB driver abstracts the inner details of the underlying USB hardware. The driver works asynchronously and communicates with the application using callbacks. The application is responsible of the descriptors and strings

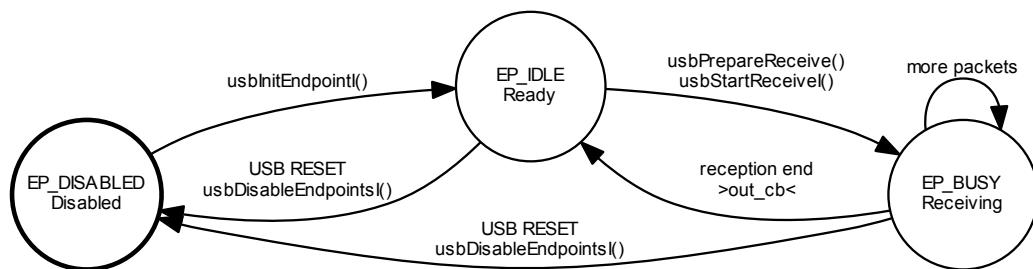
required by the USB device class to be implemented and of the handling of the specific messages sent over the endpoint zero. Standard messages are handled internally to the driver. The application can use hooks in order to handle custom messages or override the handling of the default handling of standard messages.

### 7.37.3.2 USB Endpoints

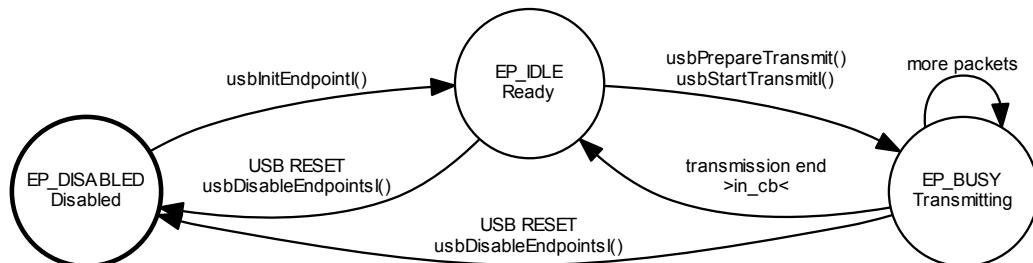
USB endpoints are the objects that the application uses to exchange data with the host. There are two kind of endpoints:

- **IN** endpoints are used by the application to transmit data to the host.
- **OUT** endpoints are used by the application to receive data from the host.

The driver invokes a callback after finishing an IN or OUT transaction. States diagram for OUT endpoints in transaction mode:



States diagram for IN endpoints in transaction mode:



### 7.37.3.3 USB Callbacks

The USB driver uses callbacks in order to interact with the application. There are several kinds of callbacks to be handled:

- Driver events callback. As example errors, suspend event, reset event etc.
- Messages Hook callback. This hook allows the application to implement handling of custom messages or to override the default handling of standard messages on endpoint zero.
- Descriptor Requested callback. When the driver endpoint zero handler receives a GET DESCRIPTOR message and needs to send a descriptor to the host it queries the application using this callback.
- Start of Frame callback. This callback is invoked each time a SOF packet is received.
- Endpoint callbacks. Each endpoint informs the application about I/O conditions using those callbacks.

## Macros

- `#define USB_USE_WAIT FALSE`  
*Enables synchronous APIs.*
- `#define USB_MAX_ENDPOINTS 4`  
*Maximum endpoint address.*
- `#define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW`  
*Status stage handling method.*
- `#define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS`  
*The address can be changed immediately upon packet reception.*
- `#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW`  
*Method for set address acknowledge.*
- `#define usb_ll_get_frame_number(usbp) 0`  
*Returns the current frame number.*
- `#define usb_ll_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)`  
*Returns the exact size of a receive transaction.*
- `#define usb_ll_connect_bus(usbp)`  
*Connects the USB device.*
- `#define usb_ll_disconnect_bus(usbp)`  
*Disconnect the USB device.*

## Helper macros for USB descriptors

- `#define USB_DESC_INDEX(i) ((uint8_t)(i))`  
*Helper macro for index values into descriptor strings.*
- `#define USB_DESC_BYTE(b) ((uint8_t)(b))`  
*Helper macro for byte values into descriptor strings.*
- `#define USB_DESC_WORD(w)`  
*Helper macro for word values into descriptor strings.*
- `#define USB_DESC_BCD(bcd)`  
*Helper macro for BCD values into descriptor strings.*
- `#define USB_DESC_DEVICE_SIZE 18U`
- `#define USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations)`  
*Device Descriptor helper macro.*
- `#define USB_DESC_CONFIGURATION_SIZE 9U`

- Configuration Descriptor size.
- #define **USB\_DESC\_CONFIGURATION**(wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower)
  - Configuration Descriptor helper macro.
- #define **USB\_DESC\_INTERFACE\_SIZE** 9U
  - Interface Descriptor size.
- #define **USB\_DESC\_INTERFACE**(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol, iInterface)
  - Interface Descriptor helper macro.
- #define **USB\_DESC\_INTERFACE\_ASSOCIATION\_SIZE** 8U
  - Interface Association Descriptor size.
- #define **USB\_DESC\_INTERFACE\_ASSOCIATION**(bFirstInterface, bInterfaceCount, bFunctionClass, bFunctionSubClass, bFunctionProtocol, iInterface)
  - Interface Association Descriptor helper macro.
- #define **USB\_DESC\_ENDPOINT\_SIZE** 7U
  - Endpoint Descriptor size.
- #define **USB\_DESC\_ENDPOINT**(bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval)
  - Endpoint Descriptor helper macro.

## Endpoint types and settings

- #define **USB\_EP\_MODE\_TYPE** 0x0003U
- #define **USB\_EP\_MODE\_TYPE\_CTRL** 0x0000U
- #define **USB\_EP\_MODE\_TYPE\_ISOC** 0x0001U
- #define **USB\_EP\_MODE\_TYPE\_BULK** 0x0002U
- #define **USB\_EP\_MODE\_TYPE\_INTR** 0x0003U

## Macro Functions

- #define **usbGetDriverState**(usbp) ((usbp)->state)
  - Returns the driver state.
- #define **usbConnectBus**(usbp) **usb\_lld\_connect\_bus**(usbp)
  - Connects the USB device.
- #define **usbDisconnectBus**(usbp) **usb\_lld\_disconnect\_bus**(usbp)
  - Disconnect the USB device.
- #define **usbGetFrameNumberX**(usbp) **usb\_lld\_get\_frame\_number**(usbp)
  - Returns the current frame number.
- #define **usbGetTransmitStatus**(usbp, ep) (((usbp)->transmitting & (uint16\_t)((unsigned)1U << (unsigned)(ep))) != 0U)
  - Returns the status of an IN endpoint.
- #define **usbGetReceiveStatus**(usbp, ep) (((usbp)->receiving & (uint16\_t)((unsigned)1U << (unsigned)(ep))) != 0U)
  - Returns the status of an OUT endpoint.
- #define **usbGetReceiveTransactionSizeX**(usbp, ep) **usb\_lld\_get\_transaction\_size**(usbp, ep)
  - Returns the exact size of a receive transaction.
- #define **usbSetupTransfer**(usbp, buf, n, endcb)
  - Request transfer setup.
- #define **usbReadSetup**(usbp, ep, buf) **usb\_lld\_read\_setup**(usbp, ep, buf)
  - Reads a setup packet from the dedicated packet buffer.

## Low level driver helper macros

- `#define _usb_isr_invoke_event_cb(usbp, evt)`  
*Common ISR code, usb event callback.*
- `#define _usb_isr_invoke_sof_cb(usbp)`  
*Common ISR code, SOF callback.*
- `#define _usb_isr_invoke_setup_cb(usbp, ep)`  
*Common ISR code, setup packet callback.*
- `#define _usb_isr_invoke_in_cb(usbp, ep)`  
*Common ISR code, IN endpoint callback.*
- `#define _usb_isr_invoke_out_cb(usbp, ep)`  
*Common ISR code, OUT endpoint event.*

## PLATFORM configuration options

- `#define PLATFORM_USB_USE_USB1 FALSE`  
*USB driver enable switch.*

## Typedefs

- `typedef struct USBDriver USBDriver`  
*Type of a structure representing an USB driver.*
- `typedef uint8_t usbep_t`  
*Type of an endpoint identifier.*
- `typedef void(* usbcallback_t) (USBDriver *usbp)`  
*Type of an USB generic notification callback.*
- `typedef void(* usbeppcallback_t) (USBDriver *usbp, usbep_t ep)`  
*Type of an USB endpoint callback.*
- `typedef void(* usbeventcb_t) (USBDriver *usbp, usbevent_t event)`  
*Type of an USB event notification callback.*
- `typedef bool(* usbreqhandler_t) (USBDriver *usbp)`  
*Type of a requests handler callback.*
- `typedef const USBDescriptor *(* usbgetdescriptor_t) (USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`  
*Type of an USB descriptor-retrieving callback.*

## Data Structures

- `struct USBDescriptor`  
*Type of an USB descriptor.*
- `struct USBInEndpointState`  
*Type of an IN endpoint state structure.*
- `struct USBOutEndpointState`  
*Type of an OUT endpoint state structure.*
- `struct USBEndpointConfig`  
*Type of an USB endpoint configuration structure.*
- `struct USBConfig`  
*Type of an USB driver configuration structure.*
- `struct USBDriver`  
*Structure representing an USB driver.*

## Functions

- static void `set_address (USBDriver *usbp)`  
*SET ADDRESS transaction callback.*
- static bool `default_handler (USBDriver *usbp)`  
*Standard requests handler.*
- void `usbInit (void)`  
*USB Driver initialization.*
- void `usbObjectInit (USBDriver *usbp)`  
*Initializes the standard part of a `USBDriver` structure.*
- void `usbStart (USBDriver *usbp, const USBConfig *config)`  
*Configures and activates the USB peripheral.*
- void `usbStop (USBDriver *usbp)`  
*Deactivates the USB peripheral.*
- void `usbInitEndpoint (USBDriver *usbp, usbep_t ep, const USBEndpointConfig *epcp)`  
*Enables an endpoint.*
- void `usbDisableEndpoints (USBDriver *usbp)`  
*Disables all the active endpoints.*
- void `usbStartReceive (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)`  
*Starts a receive transaction on an OUT endpoint.*
- void `usbStartTransmit (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)`  
*Starts a transmit transaction on an IN endpoint.*
- msg\_t `usbReceive (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)`  
*Performs a receive transaction on an OUT endpoint.*
- msg\_t `usbTransmit (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)`  
*Performs a transmit transaction on an IN endpoint.*
- bool `usbStallReceive (USBDriver *usbp, usbep_t ep)`  
*Stalls an OUT endpoint.*
- bool `usbStallTransmit (USBDriver *usbp, usbep_t ep)`  
*Stalls an IN endpoint.*
- void `_usb_reset (USBDriver *usbp)`  
*USB reset routine.*
- void `_usb_suspend (USBDriver *usbp)`  
*USB suspend routine.*
- void `_usb_wakeup (USBDriver *usbp)`  
*USB wake-up routine.*
- void `_usb_ep0setup (USBDriver *usbp, usbep_t ep)`  
*Default EP0 SETUP callback.*
- void `_usb_ep0in (USBDriver *usbp, usbep_t ep)`  
*Default EP0 IN callback.*
- void `_usb_ep0out (USBDriver *usbp, usbep_t ep)`  
*Default EP0 OUT callback.*
- void `usb_lld_init (void)`  
*Low level USB driver initialization.*
- void `usb_lld_start (USBDriver *usbp)`  
*Configures and activates the USB peripheral.*
- void `usb_lld_stop (USBDriver *usbp)`  
*Deactivates the USB peripheral.*
- void `usb_lld_reset (USBDriver *usbp)`  
*USB low level reset routine.*
- void `usb_lld_set_address (USBDriver *usbp)`

- Sets the USB address.
- void `usb_lld_init_endpoint (USBDriver *usbp, usbep_t ep)`  
Enables an endpoint.
- void `usb_lld_disable_endpoints (USBDriver *usbp)`  
Disables all the active endpoints except the endpoint zero.
- `usbepstatus_t usb_lld_get_status_out (USBDriver *usbp, usbep_t ep)`  
Returns the status of an OUT endpoint.
- `usbepstatus_t usb_lld_get_status_in (USBDriver *usbp, usbep_t ep)`  
Returns the status of an IN endpoint.
- void `usb_lld_read_setup (USBDriver *usbp, usbep_t ep, uint8_t *buf)`  
Reads a setup packet from the dedicated packet buffer.
- void `usb_lld_prepare_receive (USBDriver *usbp, usbep_t ep)`  
Prepares for a receive operation.
- void `usb_lld_prepare_transmit (USBDriver *usbp, usbep_t ep)`  
Prepares for a transmit operation.
- void `usb_lld_start_out (USBDriver *usbp, usbep_t ep)`  
Starts a receive operation on an OUT endpoint.
- void `usb_lld_start_in (USBDriver *usbp, usbep_t ep)`  
Starts a transmit operation on an IN endpoint.
- void `usb_lld_stall_out (USBDriver *usbp, usbep_t ep)`  
Brings an OUT endpoint in the stalled state.
- void `usb_lld_stall_in (USBDriver *usbp, usbep_t ep)`  
Brings an IN endpoint in the stalled state.
- void `usb_lld_clear_out (USBDriver *usbp, usbep_t ep)`  
Brings an OUT endpoint in the active state.
- void `usb_lld_clear_in (USBDriver *usbp, usbep_t ep)`  
Brings an IN endpoint in the active state.

## Enumerations

- enum `usbstate_t` {
 `USB_UNINIT` = 0, `USB_STOP` = 1, `USB_READY` = 2, `USB_SELECTED` = 3,  
`USB_ACTIVE` = 4, `USB_SUSPENDED` = 5 }
   
Type of a driver state machine possible states.
- enum `usbepstatus_t` { `EP_STATUS_DISABLED` = 0, `EP_STATUS_STALLED` = 1, `EP_STATUS_ACTIVE` = 2 }
   
Type of an endpoint status.
- enum `usbep0state_t` {
 `USB_EP0_WAITING_SETUP`, `USB_EP0_TX`, `USB_EP0_WAITING_TX0`, `USB_EP0_WAITING_STS`,  
`USB_EP0_RX`, `USB_EP0_SENDING_STS`, `USB_EP0_ERROR` }
   
Type of an endpoint zero state machine states.
- enum `usbevent_t` {
 `USB_EVENT_RESET` = 0, `USB_EVENT_ADDRESS` = 1, `USB_EVENT_CONFIGURED` = 2, `USB_EVENT_SUSPEND` = 3,  
`USB_EVENT_WAKEUP` = 4, `USB_EVENT_STALLED` = 5 }
   
Type of an enumeration of the possible USB events.

## Variables

- **USBDriver USBD1**  
*USB1 driver identifier.*
- union {
 **USBInEndpointState** in  
*IN EP0 state.*
**USBOutEndpointState** out  
*OUT EP0 state.*
} **ep0\_state**  
*EP0 state.*
- static const **USBEndpointConfig** **ep0config**  
*EP0 initialization structure.*

### 7.37.4 Macro Definition Documentation

#### 7.37.4.1 #define USB\_DESC\_INDEX( *i* ) ((uint8\_t)(*i*))

Helper macro for index values into descriptor strings.

#### 7.37.4.2 #define USB\_DESC\_BYTE( *b* ) ((uint8\_t)(*b*))

Helper macro for byte values into descriptor strings.

#### 7.37.4.3 #define USB\_DESC\_WORD( *w* )

##### Value:

```
(uint8_t) ((w) & 255U),                                     \
  (uint8_t) (((w) >> 8) & 255U)
```

Helper macro for word values into descriptor strings.

#### 7.37.4.4 #define USB\_DESC\_BCD( *bcd* )

##### Value:

```
(uint8_t) ((bcd) & 255U),                                     \
  (uint8_t) (((bcd) >> 8) & 255U)
```

Helper macro for BCD values into descriptor strings.

#### 7.37.4.5 #define USB\_DESC\_DEVICE( *bcdUSB*, *bDeviceClass*, *bDeviceSubClass*, *bDeviceProtocol*, *bMaxPacketSize*, *idVendor*, *idProduct*, *bcdDevice*, *iManufacturer*, *iProduct*, *iSerialNumber*, *bNumConfigurations* )

##### Value:

```
USB_DESC_BYTE (USB_DESC_DEVICE_SIZE),  

  \  

  USB_DESC_BYTE (USB_DESCRIPTOR_DEVICE),  

    \  

    USB_DESC_BCD (bcdUSB),  

      \  

      USB_DESC_BYTE (bDeviceClass),  

        \
```

```

USB_DESC_BYTE(bDeviceSubClass),
  \
  USB_DESC_BYTE(bDeviceProtocol),
    \
    USB_DESC_BYTE(bMaxPacketSize),
      \
      USB_DESC_WORD(idVendor),
        \
        USB_DESC_WORD(idProduct),
          \
          USB_DESC_BCD(bcdDevice),
            \
            USB_DESC_INDEX(iManufacturer),
              \
              USB_DESC_INDEX(iProduct),
                \
                USB_DESC_INDEX(iSerialNumber),
                  \
                  USB_DESC_BYTE(bNumConfigurations)

```

Device Descriptor helper macro.

#### 7.37.4.6 #define USB\_DESC\_CONFIGURATION\_SIZE 9U

Configuration Descriptor size.

#### 7.37.4.7 #define USB\_DESC\_CONFIGURATION( wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower )

**Value:**

```

USB_DESC_BYTE(USB_DESC_CONFIGURATION_SIZE),
  \
  USB_DESC_BYTE(USB_DESCRIPTOR_CONFIGURATION),
    \
    USB_DESC_WORD(wTotalLength),
      \
      USB_DESC_BYTE(bNumInterfaces),
        \
        USB_DESC_BYTE(bConfigurationValue),
          \
          USB_DESC_INDEX(iConfiguration),
            \
            USB_DESC_BYTE(bmAttributes),
              \
              USB_DESC_BYTE(bMaxPower)

```

Configuration Descriptor helper macro.

#### 7.37.4.8 #define USB\_DESC\_INTERFACE\_SIZE 9U

Interface Descriptor size.

#### 7.37.4.9 #define USB\_DESC\_INTERFACE( bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol, iInterface )

**Value:**

```

USB_DESC_BYTE(USB_DESC_INTERFACE_SIZE),
  \
  USB_DESC_BYTE(USB_DESCRIPTOR_INTERFACE),
    \
    USB_DESC_BYTE(bInterfaceNumber),
      \
      USB_DESC_BYTE(bAlternateSetting),
        \
        USB_DESC_BYTE(bNumEndpoints),
          \
          USB_DESC_BYTE(bInterfaceClass),

```

```

USB_DESC_BYTE(bInterfaceSubClass),
  \
USB_DESC_BYTE(bInterfaceProtocol),
    \
USB_DESC_INDEX(iInterface)

```

Interface Descriptor helper macro.

#### 7.37.4.10 #define USB\_DESC\_INTERFACE\_ASSOCIATION\_SIZE 8U

Interface Association Descriptor size.

#### 7.37.4.11 #define USB\_DESC\_INTERFACE\_ASSOCIATION( *bFirstInterface*, *bInterfaceCount*, *bFunctionClass*, *bFunctionSubClass*, *bFunctionProtocol*, *iInterface* )

**Value:**

```

USB_DESC_BYTE(USB_DESC_INTERFACE_ASSOCIATION_SIZE),
  \
USB_DESC_BYTE(USB_DESCRIPTOR_INTERFACE_ASSOCIATION),
  \
USB_DESC_BYTE(bFirstInterface),
  \
USB_DESC_BYTE(bInterfaceCount),
  \
USB_DESC_BYTE(bFunctionClass),
  \
USB_DESC_BYTE(bFunctionSubClass),
  \
USB_DESC_BYTE(bFunctionProtocol),
  \
USB_DESC_INDEX(iInterface)

```

Interface Association Descriptor helper macro.

#### 7.37.4.12 #define USB\_DESC\_ENDPOINT\_SIZE 7U

Endpoint Descriptor size.

#### 7.37.4.13 #define USB\_DESC\_ENDPOINT( *bEndpointAddress*, *bmAttributes*, *wMaxPacketSize*, *bInterval* )

**Value:**

```

USB_DESC_BYTE(USB_DESC_ENDPOINT_SIZE),
  \
USB_DESC_BYTE(USB_DESCRIPTOR_ENDPOINT),
  \
USB_DESC_BYTE(bEndpointAddress),
  \
USB_DESC_BYTE(bmAttributes),
  \
USB_DESC_WORD(wMaxPacketSize),
  \
USB_DESC_BYTE(bInterval)

```

Endpoint Descriptor helper macro.

#### 7.37.4.14 #define USB\_EP\_MODE\_TYPE 0x0003U

Endpoint type mask.

7.37.4.15 #define USB\_EP\_MODE\_TYPE\_CTRL 0x0000U

Control endpoint.

7.37.4.16 #define USB\_EP\_MODE\_TYPE\_ISOC 0x0001U

Isochronous endpoint.

7.37.4.17 #define USB\_EP\_MODE\_TYPE\_BULK 0x0002U

Bulk endpoint.

7.37.4.18 #define USB\_EP\_MODE\_TYPE\_INTR 0x0003U

Interrupt endpoint.

7.37.4.19 #define USB\_USE\_WAIT FALSE

Enables synchronous APIs.

#### Note

Disabling this option saves both code and data space.

7.37.4.20 #define usbGetDriverState( *usbp* ) ((*usbp*)>state)

Returns the driver state.

#### Parameters

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

#### Returns

The driver state.

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.37.4.21 #define usbConnectBus( *usbp* ) [usb\\_ll\\_connect\\_bus](#)(*usbp*)

Connects the USB device.

#### Parameters

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

#### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.37.4.22 #define usbDisconnectBus( *usbp* ) [usb\\_ll\\_disconnect\\_bus](#)(*usbp*)

Disconnect the USB device.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.37.4.23 #define usbGetFrameNumberX( *usbp* ) *usb\_lId\_get\_frame\_number(usbp)***

Returns the current frame number.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Returns**

The current frame number.

**Function Class:**

This is an **X-Class** API, this function can be invoked from any context.

**7.37.4.24 #define usbGetTransmitStatusI( *usbp*, *ep* ) (((usbp)->transmitting & (uint16\_t)((unsigned)1U << (unsigned)(ep))) != 0U)**

Returns the status of an IN endpoint.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Returns**

The operation status.

**Return values**

<i>false</i>	Endpoint ready.
<i>true</i>	Endpoint transmitting.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.37.4.25 #define usbGetReceiveStatusI( *usbp*, *ep* ) (((usbp)->receiving & (uint16\_t)((unsigned)1U << (unsigned)(ep))) != 0U)**

Returns the status of an OUT endpoint.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Returns**

The operation status.

**Return values**

<i>false</i>	Endpoint ready.
<i>true</i>	Endpoint receiving.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

**7.37.4.26 #define usbGetReceiveTransactionSizeX( *usbp*, *ep* ) *usb\_lld\_get\_transaction\_size(usbp,ep)***

Returns the exact size of a receive transaction.

The received size can be different from the size specified in [usbStartReceiveI\(\)](#) because the last packet could have a size different from the expected one.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Returns**

Received data size.

**Function Class:**

This is an **X-Class** API, this function can be invoked from any context.

**7.37.4.27 #define usbSetupTransfer( *usbp*, *buf*, *n*, *endcb* )****Value:**

```
{
    (usbp) ->ep0next   = (buf);
    (usbp) ->ep0n      = (n);
    (usbp) ->ep0endcb = (endcb);
}
```

Request transfer setup.

This macro is used by the request handling callbacks in order to prepare a transaction over the endpoint zero.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

in	<i>buf</i>	pointer to a buffer for the transaction data
in	<i>n</i>	number of bytes to be transferred
in	<i>endcb</i>	callback to be invoked after the transfer or NULL

**Function Class:**

Special function, this function has special requirements see the notes.

**7.37.4.28 #define usbReadSetup( *usbp*, *ep*, *buf* ) usb\_lld\_read\_setup(*usbp*, *ep*, *buf*)**

Reads a setup packet from the dedicated packet buffer.

This function must be invoked in the context of the `setup_cb` callback in order to read the received setup packet.

**Precondition**

In order to use this function the endpoint must have been initialized as a control endpoint.

**Note**

This function can be invoked both in thread and IRQ context.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the packet data

**Function Class:**

Special function, this function has special requirements see the notes.

**7.37.4.29 #define \_usb\_isr\_invoke\_event\_cb( *usbp*, *evt* )****Value:**

```
{
    if (((usbp) ->config->event_cb) != NULL) {
        (usbp) ->config->event_cb(usbp, evt);
    }
}
```

Common ISR code, usb event callback.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>evt</i>	USB event code

**Function Class:**

Not an API, this function is for internal use only.

**7.37.4.30 #define \_usb\_isr\_invoke\_sof\_cb( *usbp* )****Value:**

```
{\n    if (((usbp) ->config->saf_cb) != NULL) {\n        (usbp) ->config->saf_cb(usbp);\n    }\n}
```

Common ISR code, SAF callback.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.37.4.31 #define \_usb\_isr\_invoke\_setup\_cb( *usbp*, *ep* )****Value:**

```
{
  (usbp) ->epc[ep] ->setup_cb(usbp, ep);
}
```

Common ISR code, setup packet callback.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.4.32 #define \_usb\_isr\_invoke\_in\_cb( *usbp*, *ep* )****Value:**

```
{
  (usbp) ->transmitting &= ~(1 << (ep));
  if ((usbp) ->epc[ep] ->in_cb != NULL) {
    (usbp) ->epc[ep] ->in_cb(usbp, ep);
  }
  osalSysLockFromISR();
}
osalThreadResumeI(&(usbp) ->epc[ep] ->in_state ->thread, MSG_OK);
osalSysUnlockFromISR();
```

Common ISR code, IN endpoint callback.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.4.33 #define \_usb\_isr\_invoke\_out\_cb( *usbp*, *ep* )****Value:**

```
{
  (usbp) ->receiving &= ~(1 << (ep));
  if ((usbp) ->epc[ep] ->out_cb != NULL) {
    (usbp) ->epc[ep] ->out_cb(usbp, ep);
}
```

```

        }
        osalSysLockFromISR();
        \
        osalThreadResumeI (& (usbp) ->epc[ep] ->out_state ->thread,
                           usbGetReceiveTransactionSizeX (usbp, ep));
        \
        osalSysUnlockFromISR();
    }

```

Common ISR code, OUT endpoint event.

#### Parameters

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

#### Function Class:

Not an API, this function is for internal use only.

#### 7.37.4.34 #define USB\_MAX\_ENDPOINTS 4

Maximum endpoint address.

#### 7.37.4.35 #define USB\_EP0\_STATUS\_STAGE USB\_EP0\_STATUS\_STAGE\_SW

Status stage handling method.

#### 7.37.4.36 #define USB\_SET\_ADDRESS\_MODE USB\_LATE\_SET\_ADDRESS

The address can be changed immediately upon packet reception.

#### 7.37.4.37 #define USB\_SET\_ADDRESS\_ACK\_HANDLING USB\_SET\_ADDRESS\_ACK\_SW

Method for set address acknowledge.

#### 7.37.4.38 #define PLATFORM\_USB\_USE\_USB1 FALSE

USB driver enable switch.

If set to TRUE the support for USB1 is included.

#### Note

The default is FALSE.

#### 7.37.4.39 #define usb\_ll\_get\_frame\_number( *usbp* ) 0

Returns the current frame number.

#### Parameters

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Returns**

The current frame number.

**Function Class:**

Not an API, this function is for internal use only.

**7.37.4.40 #define usb\_lld\_get\_transaction\_size( usbp, ep ) ((usbp)->epc[ep]->out\_state->rxcnt)**

Returns the exact size of a receive transaction.

The received size can be different from the size specified in [usbStartReceiveI\(\)](#) because the last packet could have a size different from the expected one.

**Precondition**

The OUT endpoint must have been configured in transaction mode in order to use this function.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Returns**

Received data size.

**Function Class:**

Not an API, this function is for internal use only.

**7.37.4.41 #define usb\_lld\_connect\_bus( usbp )**

Connects the USB device.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.37.4.42 #define usb\_lld\_disconnect\_bus( usbp )**

Disconnect the USB device.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

## 7.37.5 Typedef Documentation

**7.37.5.1 typedef struct USBDriver USBDriver**

Type of a structure representing an USB driver.

**7.37.5.2 typedef uint8\_t usbep\_t**

Type of an endpoint identifier.

7.37.5.3 `typedef void(* usbcallback_t) (USBDriver *usbp)`

Type of an USB generic notification callback.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object triggering the callback
----	-------------	---

**7.37.5.4 [typedef void\(\\* usbepcallback\\_t\)\(USBDriver \\*usbp, usbep\\_t ep\)](#)**

Type of an USB endpoint callback.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object triggering the callback
in	<i>ep</i>	endpoint number

**7.37.5.5 [typedef void\(\\* usbeventcb\\_t\)\(USBDriver \\*usbp, usbevent\\_t event\)](#)**

Type of an USB event notification callback.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object triggering the callback
in	<i>event</i>	event type

**7.37.5.6 [typedef bool\(\\* usbreqhandler\\_t\)\(USBDriver \\*usbp\)](#)**

Type of a requests handler callback.

The request is encoded in the `usb_setup` buffer.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object triggering the callback
----	-------------	---

**Returns**

The request handling exit code.

**Return values**

<i>false</i>	Request not recognized by the handler.
<i>true</i>	Request handled.

**7.37.5.7 [typedef const USBDescriptor\\*\(\\* usbgetdescriptor\\_t\)\(USBDriver \\*usbp, uint8\\_t dtype, uint8\\_t dindex, uint16\\_t lang\)](#)**

Type of an USB descriptor-retrieving callback.

**7.37.6 Enumeration Type Documentation****7.37.6.1 enum [usbstate\\_t](#)**

Type of a driver state machine possible states.

**Enumerator**

**[USB\\_UNINIT](#)** Not initialized.

**USB\_STOP** Stopped.

**USB\_READY** Ready, after bus reset.

**USB\_SELECTED** Address assigned.

**USB\_ACTIVE** Active, configuration selected.

**USB\_SUSPENDED** Suspended, low power mode.

#### 7.37.6.2 enum usbepstatus\_t

Type of an endpoint status.

##### Enumerator

**EP\_STATUS\_DISABLED** Endpoint not active.

**EP\_STATUS\_STALLED** Endpoint opened but stalled.

**EP\_STATUS\_ACTIVE** Active endpoint.

#### 7.37.6.3 enum usbep0state\_t

Type of an endpoint zero state machine states.

##### Enumerator

**USB\_EP0\_WAITING\_SETUP** Waiting for SETUP data.

**USB\_EP0\_TX** Transmitting.

**USB\_EP0\_WAITING\_TX0** Waiting transmit 0.

**USB\_EP0\_WAITING\_STS** Waiting status.

**USB\_EP0\_RX** Receiving.

**USB\_EP0\_SENDING\_STS** Sending status.

**USB\_EP0\_ERROR** Error, EP0 stalled.

#### 7.37.6.4 enum usbevent\_t

Type of an enumeration of the possible USB events.

##### Enumerator

**USB\_EVENT\_RESET** Driver has been reset by host.

**USB\_EVENT\_ADDRESS** Address assigned.

**USB\_EVENT\_CONFIGURED** Configuration selected.

**USB\_EVENT\_SUSPEND** Entering suspend mode.

**USB\_EVENT\_WAKEUP** Leaving suspend mode.

**USB\_EVENT\_STALLED** Endpoint 0 error, stalled.

#### 7.37.7 Function Documentation

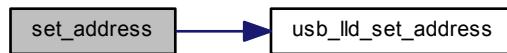
##### 7.37.7.1 static void set\_address ( **USBDriver** \* *usbp* ) [static]

SET ADDRESS transaction callback.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

Here is the call graph for this function:



### 7.37.7.2 static bool default\_handler ( [USBDriver](#) \* *usbp* ) [static]

Standard requests handler.

This is the standard requests default handler, most standard requests are handled here, the user can override the standard handling using the `requests_hook_cb` hook in the [USBConfig](#) structure.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

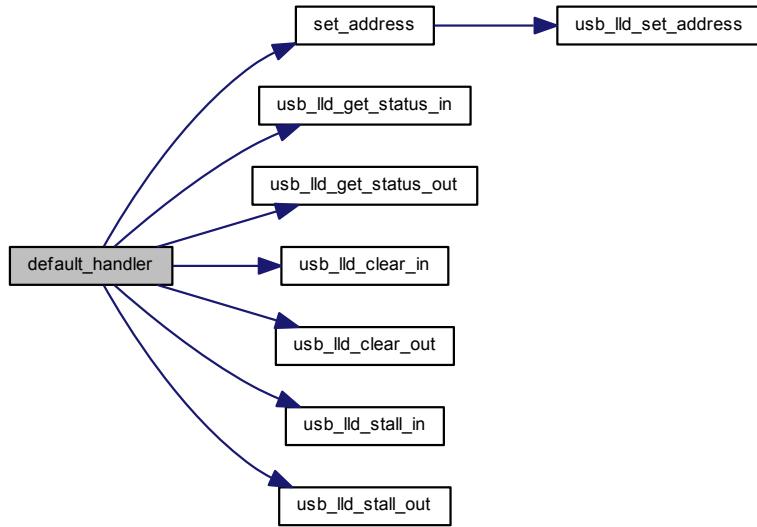
**Returns**

The request handling exit code.

**Return values**

<i>false</i>	Request not recognized by the handler or error.
<i>true</i>	Request handled.

Here is the call graph for this function:



### 7.37.7.3 void usblInit( void )

USB Driver initialization.

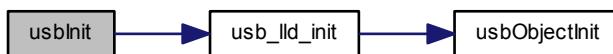
#### Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

#### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



### 7.37.7.4 void usbObjectInit( USBDriver \* usbp )

Initializes the standard part of a [USBDriver](#) structure.

**Parameters**

out	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
-----	-------------	---

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

**7.37.7.5 void usbStart ( [USBDriver](#) \* *usbp*, const [USBConfig](#) \* *config* )**

Configures and activates the USB peripheral.

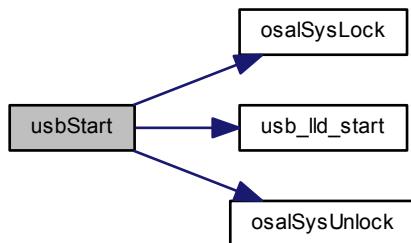
**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>config</i>	pointer to the <a href="#">USBConfig</a> object

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.37.7.6 void usbStop ( [USBDriver](#) \* *usbp* )**

Deactivates the USB peripheral.

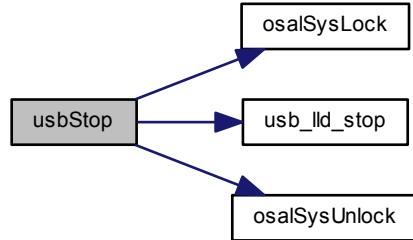
**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



#### 7.37.7.7 void usbInitEndpointl ( **USBDriver** \* *usbp*, **usbep\_t** *ep*, const **USBEndpointConfig** \* *epcp* )

Enables an endpoint.

This function enables an endpoint, both IN and/or OUT directions depending on the configuration structure.

##### Note

This function must be invoked in response of a SET\_CONFIGURATION or SET\_INTERFACE message.

##### Parameters

in	<i>usbp</i>	pointer to the <b>USBDriver</b> object
in	<i>ep</i>	endpoint number
in	<i>epcp</i>	the endpoint configuration

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.37.7.8 void usbDisableEndpointsl ( **USBDriver** \* *usbp* )

Disables all the active endpoints.

This function disables all the active endpoints except the endpoint zero.

**Note**

This function must be invoked in response of a SET\_CONFIGURATION message with configuration number zero.

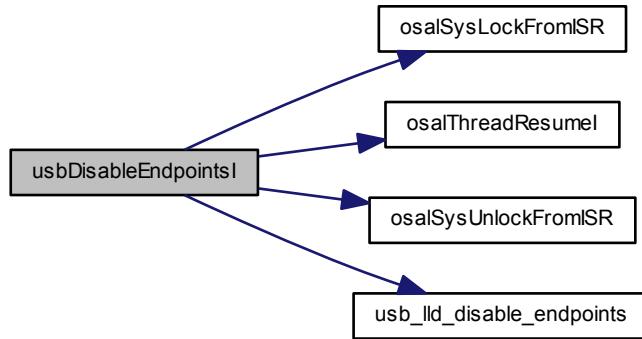
**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.37.7.9 void `usbStartReceiveI ( USBDriver * usbp, usbep_t ep, uint8_t * buf, size_t n )`

Starts a receive transaction on an OUT endpoint.

**Note**

This function is meant to be called from ISR context outside critical zones because there is a potentially slow operation inside.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the received data
in	<i>n</i>	transaction size. It is recommended a multiple of the packet size because the excess is discarded.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.37.7.10 void usbStartTransmitI ( **USBDriver** \* *usbp*, **usbep\_t** *ep*, const **uint8\_t** \* *buf*, **size\_t** *n* )

Starts a transmit transaction on an IN endpoint.

##### Note

This function is meant to be called from ISR context outside critical zones because there is a potentially slow operation inside.

##### Parameters

in	<i>usbp</i>	pointer to the <b>USBDriver</b> object
in	<i>ep</i>	endpoint number
in	<i>buf</i>	buffer where to fetch the data to be transmitted
in	<i>n</i>	transaction size

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



#### 7.37.7.11 **msg\_t** usbReceive ( **USBDriver** \* *usbp*, **usbep\_t** *ep*, **uint8\_t** \* *buf*, **size\_t** *n* )

Performs a receive transaction on an OUT endpoint.

##### Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the received data
in	<i>n</i>	transaction size. It is recommended a multiple of the packet size because the excess is discarded.

**Returns**

The received effective data size, it can be less than the amount specified.

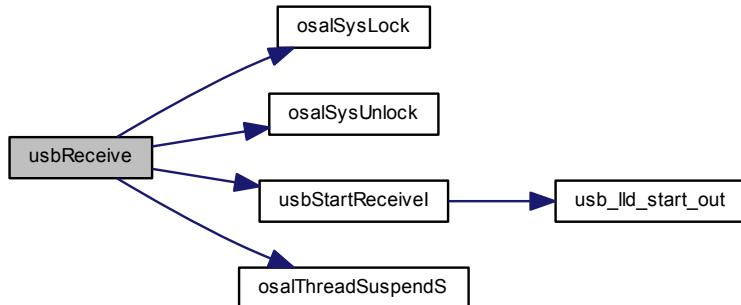
**Return values**

<code>MSG_RESET</code>	driver not in <code>USB_ACTIVE</code> state or the operation has been aborted by an USB reset or a transition to the <code>USB_SUSPENDED</code> state.
------------------------	--

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.37.7.12 `msg_t usbTransmit( USBDriver * usbp, usbep_t ep, const uint8_t * buf, size_t n )`**

Performs a transmit transaction on an IN endpoint.

**Parameters**

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number
in	<i>buf</i>	buffer where to fetch the data to be transmitted
in	<i>n</i>	transaction size

**Returns**

The operation status.

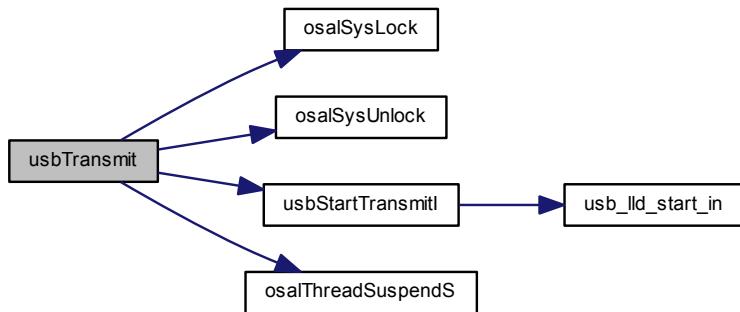
## Return values

<code>MSG_OK</code>	operation performed successfully.
<code>MSG_RESET</code>	driver not in <code>USB_ACTIVE</code> state or the operation has been aborted by an USB reset or a transition to the <code>USB_SUSPENDED</code> state.

## Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.37.7.13 `bool usbStallReceivel( USBDriver * usbp, usbep_t ep )`

Stalls an OUT endpoint.

## Parameters

in	<code>usbp</code>	pointer to the <code>USBDriver</code> object
in	<code>ep</code>	endpoint number

## Returns

The operation status.

## Return values

<code>false</code>	Endpoint stalled.
<code>true</code>	Endpoint busy, not stalled.

## Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.37.7.14 bool usbStallTransmit( USBDriver \* usbp, usbep\_t ep )

Stalls an IN endpoint.

#### Parameters

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

#### Returns

The operation status.

#### Return values

<i>false</i>	Endpoint stalled.
<i>true</i>	Endpoint busy, not stalled.

#### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



### 7.37.7.15 void \_usb\_reset( USBDriver \* usbp )

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.

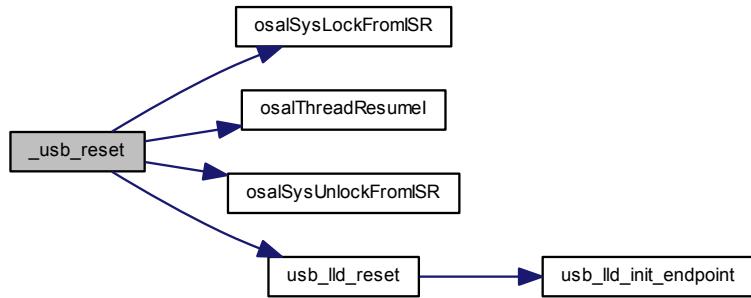
**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.37.7.16 void \_usb\_suspend( USBDriver \* usbp )**

USB suspend routine.

This function must be invoked when an USB bus suspend condition is detected.

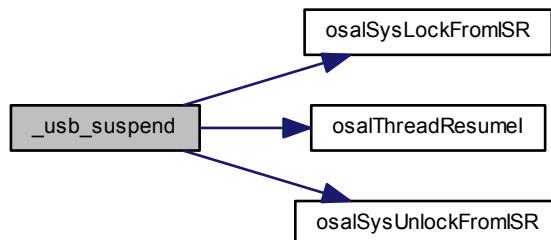
**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



**7.37.7.17 void \_usb\_wakeup ( **USBDriver** \* *usbp* )**

USB wake-up routine.

This function must be invoked when an USB bus wake-up condition is detected.

**Parameters**

in	<i>usbp</i>	pointer to the <b>USBDriver</b> object
----	-------------	--

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.18 void \_usb\_ep0setup ( **USBDriver** \* *usbp*, **usbep\_t** *ep* )**

Default EP0 SETUP callback.

This function is used by the low level driver as default handler for EP0 SETUP events.

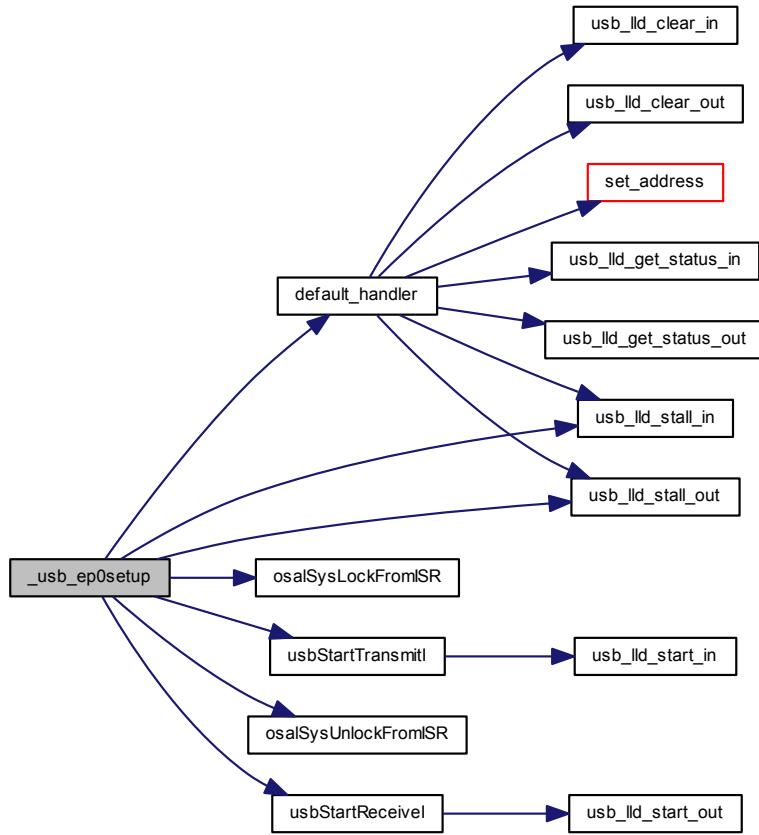
**Parameters**

in	<i>usbp</i>	pointer to the <b>USBDriver</b> object
in	<i>ep</i>	endpoint number, always zero

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.37.7.19 void \_usb\_ep0in ( USBDriver \* usbp, usbep\_t ep )

Default EP0 IN callback.

This function is used by the low level driver as default handler for EP0 IN events.

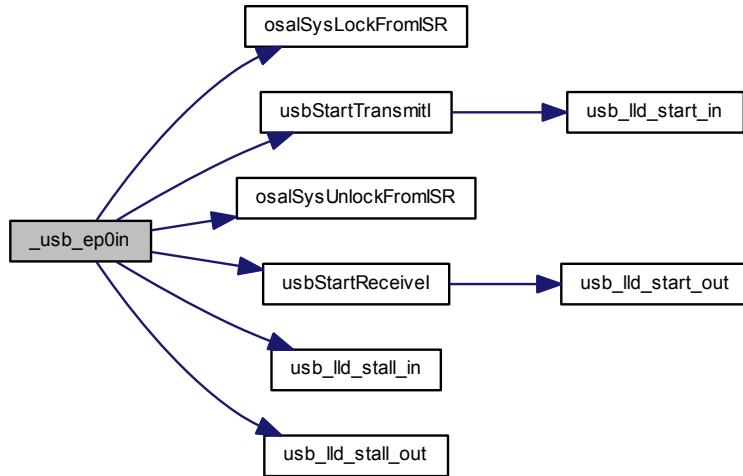
##### Parameters

in	<code>usbp</code>	pointer to the <code>USBDriver</code> object
in	<code>ep</code>	endpoint number, always zero

##### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



### 7.37.7.20 void \_usb\_ep0out ( **USBDriver** \* *usbp*, **usbep\_t** *ep* )

Default EP0 OUT callback.

This function is used by the low level driver as default handler for EP0 OUT events.

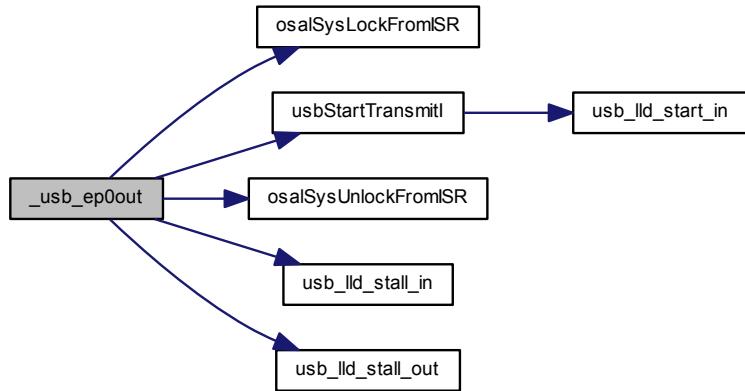
#### Parameters

in	<i>usbp</i>	pointer to the <b>USBDriver</b> object
in	<i>ep</i>	endpoint number, always zero

#### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.37.7.21 void usb\_lld\_init ( void )

Low level USB driver initialization.

##### Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



#### 7.37.7.22 void usb\_lld\_start ( USBDriver \* usbp )

Configures and activates the USB peripheral.

##### Parameters

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

##### Function Class:

Not an API, this function is for internal use only.

7.37.7.23 void **usb\_lld\_stop** ( **USBDriver** \* *usb* )

Deactivates the USB peripheral.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.24 void usb\_lld\_reset ( [USBDriver](#) \* *usbp* )**

USB low level reset routine.

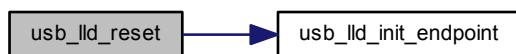
**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

**7.37.7.25 void usb\_lld\_set\_address ( [USBDriver](#) \* *usbp* )**

Sets the USB address.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.26 void usb\_lld\_init\_endpoint ( [USBDriver](#) \* *usbp*, [usbep\\_t](#) *ep* )**

Enables an endpoint.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.27 void usb\_lld\_disable\_endpoints ( **USBDriver** \* *usbP* )**

Disables all the active endpoints except the endpoint zero.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
----	-------------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.28 usbepstatus\_t usb\_lld\_get\_status\_out ( USBDriver \* *usbp*, usbep\_t *ep* )**

Returns the status of an OUT endpoint.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Returns**

The endpoint status.

**Return values**

<i>EP_STATUS_DISABLED</i>	The endpoint is not active.
<i>EP_STATUS_STALLED</i>	The endpoint is stalled.
<i>EP_STATUS_ACTIVE</i>	The endpoint is active.

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.29 usbepstatus\_t usb\_lld\_get\_status\_in ( USBDriver \* *usbp*, usbep\_t *ep* )**

Returns the status of an IN endpoint.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Returns**

The endpoint status.

**Return values**

<i>EP_STATUS_DISABLED</i>	The endpoint is not active.
<i>EP_STATUS_STALLED</i>	The endpoint is stalled.
<i>EP_STATUS_ACTIVE</i>	The endpoint is active.

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.30 void usb\_lld\_read\_setup ( USBDriver \* *usbp*, usbep\_t *ep*, uint8\_t \* *buf* )**

Reads a setup packet from the dedicated packet buffer.

This function must be invoked in the context of the `setup_cb` callback in order to read the received setup packet.

**Precondition**

In order to use this function the endpoint must have been initialized as a control endpoint.

**Postcondition**

The endpoint is ready to accept another packet.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the packet data

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.31 void usb\_lld\_prepare\_receive ( [USBDriver](#) \* *usbp*, [usbep\\_t](#) *ep* )**

Prepares for a receive operation.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.32 void usb\_lld\_prepare\_transmit ( [USBDriver](#) \* *usbp*, [usbep\\_t](#) *ep* )**

Prepares for a transmit operation.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.33 void usb\_lld\_start\_out ( [USBDriver](#) \* *usbp*, [usbep\\_t](#) *ep* )**

Starts a receive operation on an OUT endpoint.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.34 void usb\_lld\_start\_in ( [USBDriver](#) \* *usbp*, [usbep\\_t](#) *ep* )**

Starts a transmit operation on an IN endpoint.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.35 void usb\_lld\_stall\_out( USBDriver \* *usbp*, usbep\_t *ep* )**

Brings an OUT endpoint in the stalled state.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.36 void usb\_lld\_stall\_in( USBDriver \* *usbp*, usbep\_t *ep* )**

Brings an IN endpoint in the stalled state.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.37 void usb\_lld\_clear\_out( USBDriver \* *usbp*, usbep\_t *ep* )**

Brings an OUT endpoint in the active state.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.7.38 void usb\_lld\_clear\_in( USBDriver \* *usbp*, usbep\_t *ep* )**

Brings an IN endpoint in the active state.

**Parameters**

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number

**Function Class:**

Not an API, this function is for internal use only.

**7.37.8 Variable Documentation****7.37.8.1 USBDriver USBD1**

USB1 driver identifier.

**7.37.8.2 union { ... } ep0\_state [static]**

EP0 state.

**Note**

It is an union because IN and OUT endpoints are never used at the same time for EP0.

**7.37.8.3 USBInEndpointState { ... } in**

IN EP0 state.

**7.37.8.4 USBOutEndpointState { ... } out**

OUT EP0 state.

**7.37.8.5 const USBEndpointConfig ep0config [static]****Initial value:**

```
= {
  USB_EP_MODE_TYPE_CTRL,
  _usb_ep0setup,
  _usb_ep0in,
  _usb_ep0out,
  0x40,
  0x40,
  &ep0_state.in,
  &ep0_state.out
}
```

EP0 initialization structure.

## 7.38 USB CDC Header

USB CDC Support Header.

### 7.38.1 Detailed Description

USB CDC Support Header.

This header contains definitions and types related to USB CDC.

CDC specific messages.

- #define **CDC\_SEND\_ENCAPSULATED\_COMMAND** 0x00U
- #define **CDC\_GET\_ENCAPSULATED\_RESPONSE** 0x01U
- #define **CDC\_SET\_COMM\_FEATURE** 0x02U
- #define **CDC\_GET\_COMM\_FEATURE** 0x03U
- #define **CDC\_CLEAR\_COMM\_FEATURE** 0x04U
- #define **CDC\_SET\_AUX\_LINE\_STATE** 0x10U
- #define **CDC\_SET\_HOOK\_STATE** 0x11U
- #define **CDC\_PULSE\_SETUP** 0x12U
- #define **CDC\_SEND\_PULSE** 0x13U
- #define **CDC\_SET\_PULSE\_TIME** 0x14U
- #define **CDC\_RING\_AUX\_JACK** 0x15U
- #define **CDC\_SET\_LINE\_CODING** 0x20U
- #define **CDC\_GET\_LINE\_CODING** 0x21U
- #define **CDC\_SET\_CONTROL\_LINE\_STATE** 0x22U
- #define **CDC\_SEND\_BREAK** 0x23U
- #define **CDC\_SET\_RINGER\_PARMS** 0x30U
- #define **CDC\_GET\_RINGER\_PARMS** 0x31U
- #define **CDC\_SET\_OPERATION\_PARMS** 0x32U
- #define **CDC\_GET\_OPERATION\_PARMS** 0x33U

CDC classes

- #define **CDC\_COMMUNICATION\_INTERFACE\_CLASS** 0x02U
- #define **CDC\_DATA\_INTERFACE\_CLASS** 0x0AU

CDC subclasses

- #define **CDC\_ABSTRACT\_CONTROL\_MODEL** 0x02U

CDC descriptors

- #define **CDC\_CS\_INTERFACE** 0x24U

CDC subdescriptors

- #define **CDC\_HEADER** 0x00U
- #define **CDC\_CALL\_MANAGEMENT** 0x01U
- #define **CDC\_ABSTRACT\_CONTROL\_MANAGEMENT** 0x02U
- #define **CDC\_UNION** 0x06U

## Line Control bit definitions.

- #define **LC\_STOP\_1** 0U
- #define **LC\_STOP\_1P5** 1U
- #define **LC\_STOP\_2** 2U
- #define **LC\_PARITY\_NONE** 0U
- #define **LC\_PARITY\_ODD** 1U
- #define **LC\_PARITY\_EVEN** 2U
- #define **LC\_PARITY\_MARK** 3U
- #define **LC\_PARITY\_SPACE** 4U

## Data Structures

- struct [`cdc\_linecoding\_t`](#)  
*Type of Line Coding structure.*

## 7.39 WDG Driver

Generic WDG Driver.

### 7.39.1 Detailed Description

Generic WDG Driver.

This module defines an abstract interface for a watchdog timer.

#### Precondition

In order to use the WDG driver the `HAL_USE_WDG` option must be enabled in `halconf.h`.

#### Macros

- `#define wdgReset(wdgp) wdg_lld_reset(wdgp)`  
*Resets WDG's counter.*

#### Configuration options

- `#define PLATFORM_WDG_USE_WDG1 FALSE`  
*WDG1 driver enable switch.*

#### Typedefs

- `typedef struct WDGDriver WDGDriver`  
*Type of a structure representing an WDG driver.*

#### Data Structures

- `struct WDGConfig`  
*Driver configuration structure.*
- `struct WDGDriver`  
*Structure representing an WDG driver.*

#### Functions

- `void wdgInit (void)`  
*WDG Driver initialization.*
- `void wdgStart (WDGDriver *wdgp, const WDGConfig *config)`  
*Configures and activates the WDG peripheral.*
- `void wdgStop (WDGDriver *wdgp)`  
*Deactivates the WDG peripheral.*
- `void wdgReset (WDGDriver *wdgp)`  
*Resets WDG's counter.*
- `void wdg_lld_init (void)`  
*Low level WDG driver initialization.*
- `void wdg_lld_start (WDGDriver *wdgp)`  
*Configures and activates the WDG peripheral.*
- `void wdg_lld_stop (WDGDriver *wdgp)`

- void `wdg_lld_reset (WDGDriver *wdgp)`

*Deactivates the WDG peripheral.*

*Reloads WDG's counter.*

## Enumerations

- enum `wdgstate_t { WDG_UNINIT = 0, WDG_STOP = 1, WDG_READY = 2 }`

*Driver state machine possible states.*

### 7.39.2 Macro Definition Documentation

#### 7.39.2.1 #define wdgReset( wdgp ) wdg\_lld\_reset(wdgp)

Resets WDG's counter.

##### Parameters

in	<code>wdgp</code>	pointer to the <code>WDGDriver</code> object
----	-------------------	--

##### Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

#### 7.39.2.2 #define PLATFORM\_WDG\_USE\_WDG1 FALSE

WDG1 driver enable switch.

##### Note

The default is FALSE.

### 7.39.3 Typedef Documentation

#### 7.39.3.1 typedef struct WDGDriver WDGDriver

Type of a structure representing an WDG driver.

### 7.39.4 Enumeration Type Documentation

#### 7.39.4.1 enum wdgstate\_t

Driver state machine possible states.

##### Enumerator

**WDG\_UNINIT** Not initialized.

**WDG\_STOP** Stopped.

**WDG\_READY** Ready.

### 7.39.5 Function Documentation

#### 7.39.5.1 void wdglInit( void )

WDG Driver initialization.

##### Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

##### Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



#### 7.39.5.2 void wdglStart( WDGDriver \*wdgp, const WDGConfig \*config )

Configures and activates the WDG peripheral.

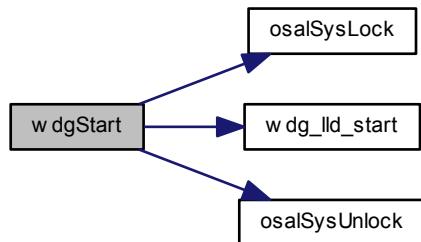
##### Parameters

in	<code>wdgp</code>	pointer to the <code>WDGDriver</code> object
in	<code>config</code>	pointer to the <code>WDGConfig</code> object

##### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.39.5.3 void wdgStop ( WDGDriver \* *wdgp* )

Deactivates the WDG peripheral.

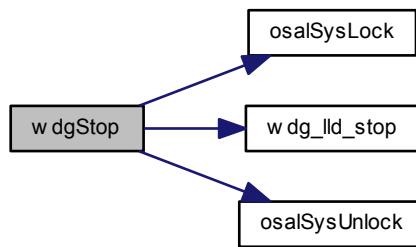
**Parameters**

in	<i>wdgp</i>	pointer to the <a href="#">WDGDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.39.5.4 void wdgReset ( WDGDriver \* *wdgp* )**

Resets WDG's counter.

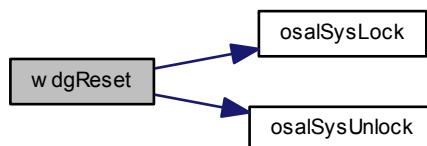
**Parameters**

in	<i>wdgp</i>	pointer to the <a href="#">WDGDriver</a> object
----	-------------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**7.39.5.5 void wdg\_Ild\_init ( void )**

Low level WDG driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

**7.39.5.6 void wdg\_lld\_start ( WDGDriver \* wdgp )**

Configures and activates the WDG peripheral.

**Parameters**

in	wdgp	pointer to the <a href="#">WDGDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

**7.39.5.7 void wdg\_lld\_stop ( WDGDriver \* wdgp )**

Deactivates the WDG peripheral.

**Parameters**

in	wdgp	pointer to the <a href="#">WDGDriver</a> object
----	------	---

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.39.5.8 void wdg\_lld\_reset ( WDGDriver \* wdgp )**

Reloads WDG's counter.

**Parameters**

in	wdgp	pointer to the <a href="#">WDGDriver</a> object
----	------	---

**Function Class:**

Not an API, this function is for internal use only.

## Chapter 8

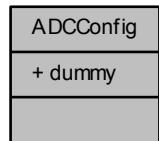
# Data Structure Documentation

### 8.1 ADCConfig Struct Reference

Driver configuration structure.

```
#include <adc_lld.h>
```

Collaboration diagram for ADCConfig:



#### 8.1.1 Detailed Description

Driver configuration structure.

**Note**

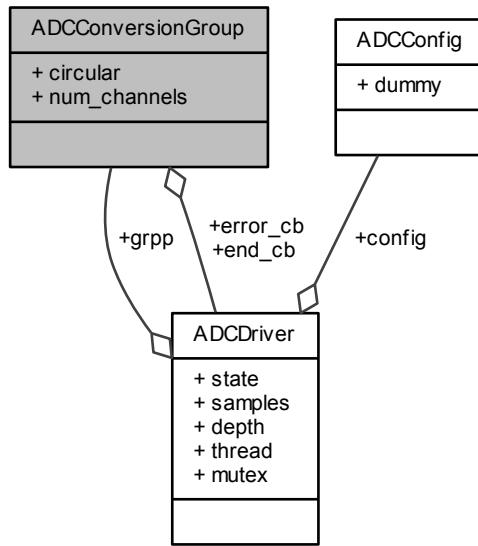
It could be empty on some architectures.

## 8.2 ADCConversionGroup Struct Reference

Conversion group configuration structure.

```
#include <adc_lld.h>
```

Collaboration diagram for ADCConversionGroup:



### Data Fields

- bool [circular](#)  
*Enables the circular buffer mode for the group.*
- [adc\\_channels\\_num\\_t num\\_channels](#)  
*Number of the analog channels belonging to the conversion group.*
- [adccallback\\_t end\\_cb](#)  
*Callback function associated to the group or `NULL`.*
- [adcerrorcallback\\_t error\\_cb](#)  
*Error callback or `NULL`.*

### 8.2.1 Detailed Description

Conversion group configuration structure.

This implementation-dependent structure describes a conversion operation.

**Note**

The use of this configuration structure requires knowledge of PLATFORM ADC cell registers interface, please refer to the PLATFORM reference manual for details.

## 8.2.2 Field Documentation

### 8.2.2.1 bool ADCConversionGroup::circular

Enables the circular buffer mode for the group.

### 8.2.2.2 adc\_channels\_num\_t ADCConversionGroup::num\_channels

Number of the analog channels belonging to the conversion group.

### 8.2.2.3 adccallback\_t ADCConversionGroup::end\_cb

Callback function associated to the group or NULL.

### 8.2.2.4 adcerrorcallback\_t ADCConversionGroup::error\_cb

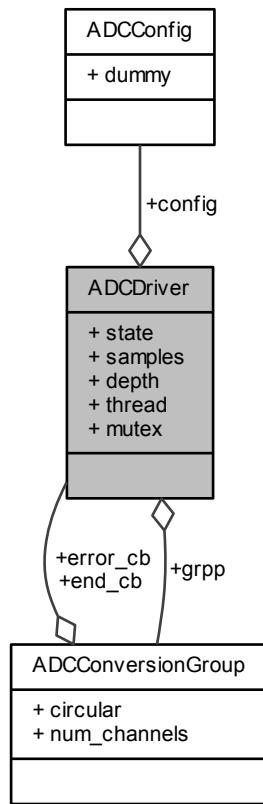
Error callback or NULL.

## 8.3 ADCDriver Struct Reference

Structure representing an ADC driver.

```
#include <adc_lld.h>
```

Collaboration diagram for ADCDriver:



## Data Fields

- `adcstate_t state`  
*Driver state.*
- `const ADCCConfig * config`  
*Current configuration data.*
- `adosample_t * samples`  
*Current samples buffer pointer or NULL.*
- `size_t depth`  
*Current samples buffer depth or 0.*
- `const ADCCConversionGroup * grpp`  
*Current conversion group pointer or NULL.*
- `thread_reference_t thread`  
*Waiting thread.*
- `mutex_t mutex`  
*Mutex protecting the peripheral.*

### 8.3.1 Detailed Description

Structure representing an ADC driver.

### 8.3.2 Field Documentation

#### 8.3.2.1 `adcstate_t` ADCDriver::state

Driver state.

#### 8.3.2.2 `const ADCConfig*` ADCDriver::config

Current configuration data.

#### 8.3.2.3 `adcsample_t*` ADCDriver::samples

Current samples buffer pointer or NULL.

#### 8.3.2.4 `size_t` ADCDriver::depth

Current samples buffer depth or 0.

#### 8.3.2.5 `const ADCConversionGroup*` ADCDriver::grpp

Current conversion group pointer or NULL.

#### 8.3.2.6 `thread_reference_t` ADCDriver::thread

Waiting thread.

#### 8.3.2.7 `mutex_t` ADCDriver::mutex

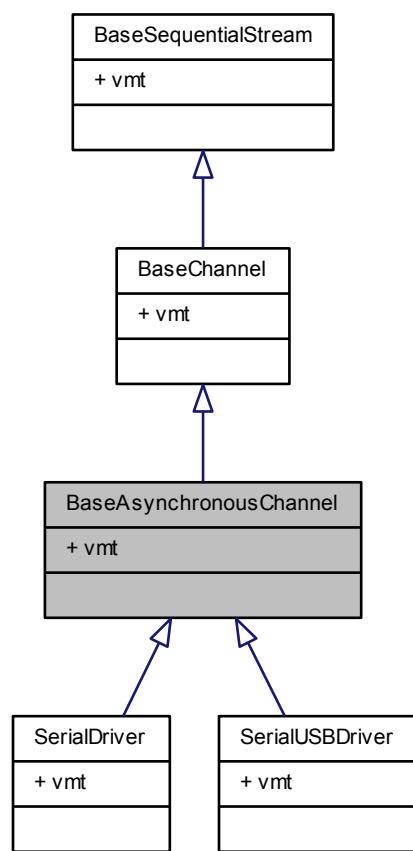
Mutex protecting the peripheral.

## 8.4 BaseAsynchronousChannel Struct Reference

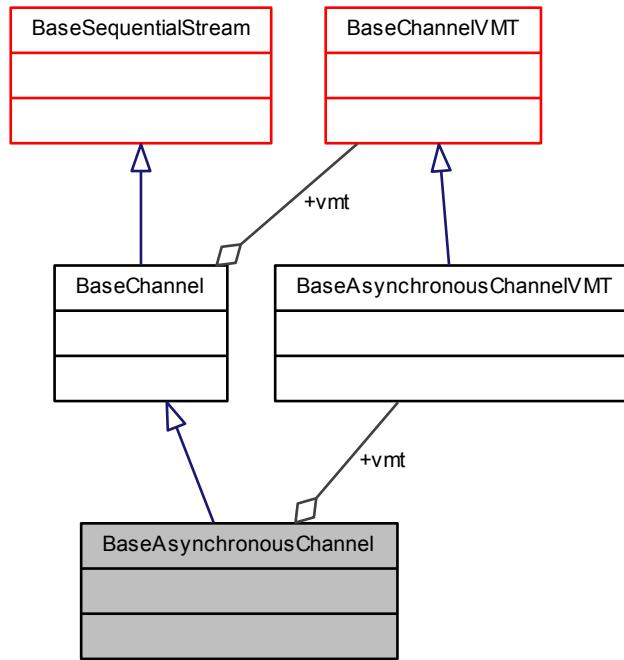
Base asynchronous channel class.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseAsynchronousChannel:



Collaboration diagram for BaseAsynchronousChannel:



## Data Fields

- const struct [BaseAsynchronousChannelVMT](#) \* vmt

*Virtual Methods Table.*

### 8.4.1 Detailed Description

Base asynchronous channel class.

This class extends [BaseChannel](#) by adding event sources fields for asynchronous I/O for use in an event-driven environment.

### 8.4.2 Field Documentation

#### 8.4.2.1 const struct [BaseAsynchronousChannelVMT](#)\* [BaseAsynchronousChannel::vmt](#)

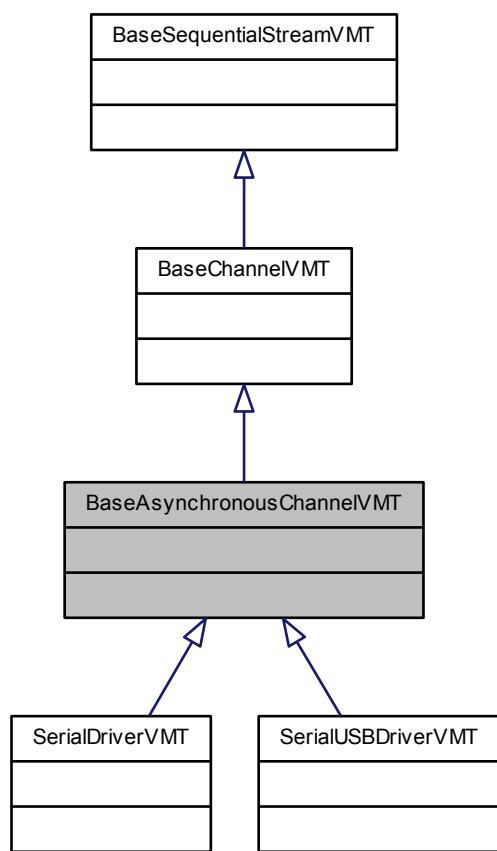
*Virtual Methods Table.*

## 8.5 BaseAsynchronousChannelVMT Struct Reference

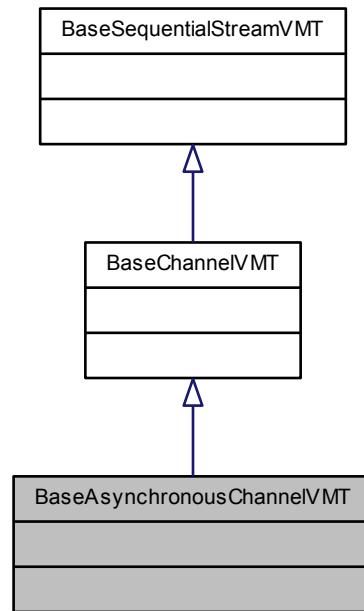
[BaseAsynchronousChannel](#) virtual methods table.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseAsynchronousChannelVMT:



Collaboration diagram for BaseAsynchronousChannelVMT:



### 8.5.1 Detailed Description

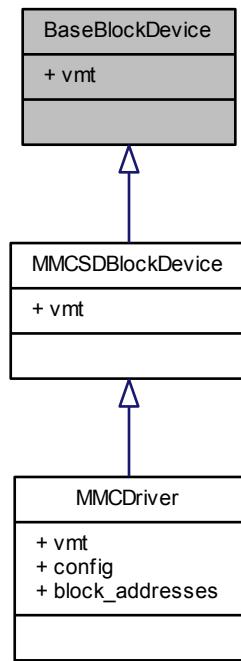
[BaseAsynchronousChannel](#) virtual methods table.

## 8.6 BaseBlockDevice Struct Reference

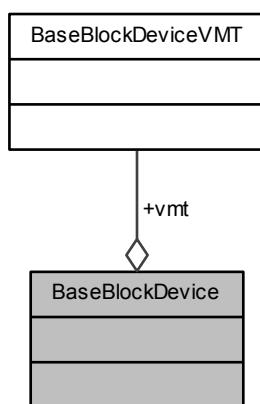
Base block device class.

```
#include <hal_ioblock.h>
```

Inheritance diagram for BaseBlockDevice:



Collaboration diagram for BaseBlockDevice:



## Data Fields

- const struct [BaseBlockDeviceVMT](#) \* `vmt`

*Virtual Methods Table.*

### 8.6.1 Detailed Description

Base block device class.

This class represents a generic, block-accessible, device.

### 8.6.2 Field Documentation

#### 8.6.2.1 const struct BaseBlockDeviceVMT\* BaseBlockDevice::vmt

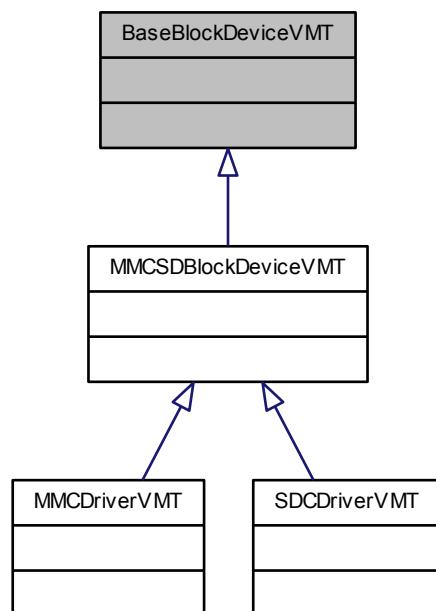
Virtual Methods Table.

## 8.7 BaseBlockDeviceVMT Struct Reference

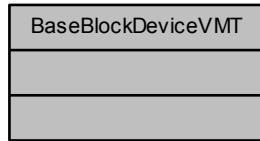
[BaseBlockDevice](#) virtual methods table.

```
#include <hal_ioblock.h>
```

Inheritance diagram for BaseBlockDeviceVMT:



Collaboration diagram for BaseBlockDeviceVMT:



### 8.7.1 Detailed Description

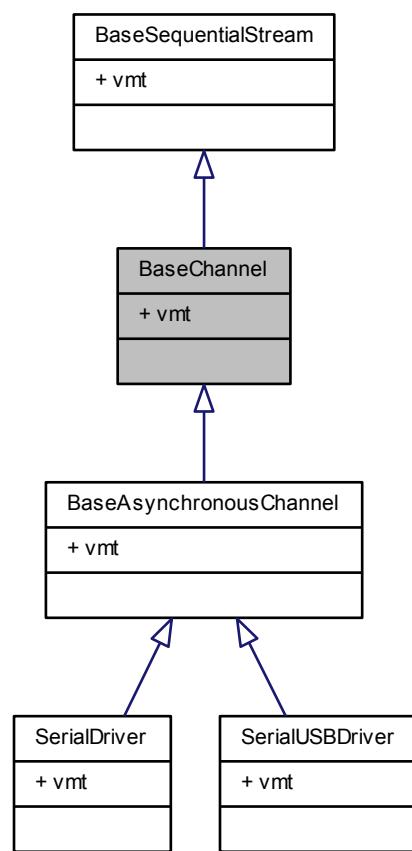
`BaseBlockDevice` virtual methods table.

## 8.8 BaseChannel Struct Reference

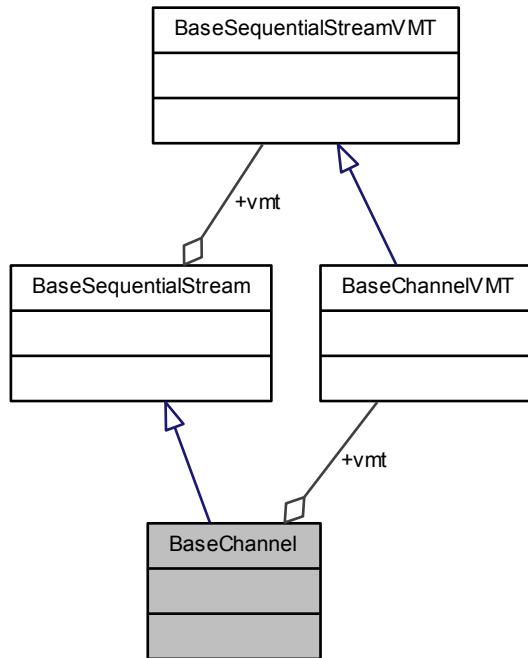
Base channel class.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseChannel:



Collaboration diagram for BaseChannel:



## Data Fields

- const struct `BaseChannelVMT` \* `vmt`

*Virtual Methods Table.*

### 8.8.1 Detailed Description

Base channel class.

This class represents a generic, byte-wide, I/O channel. This class introduces generic I/O primitives with timeout specification.

### 8.8.2 Field Documentation

#### 8.8.2.1 const struct `BaseChannelVMT`\* `BaseChannel::vmt`

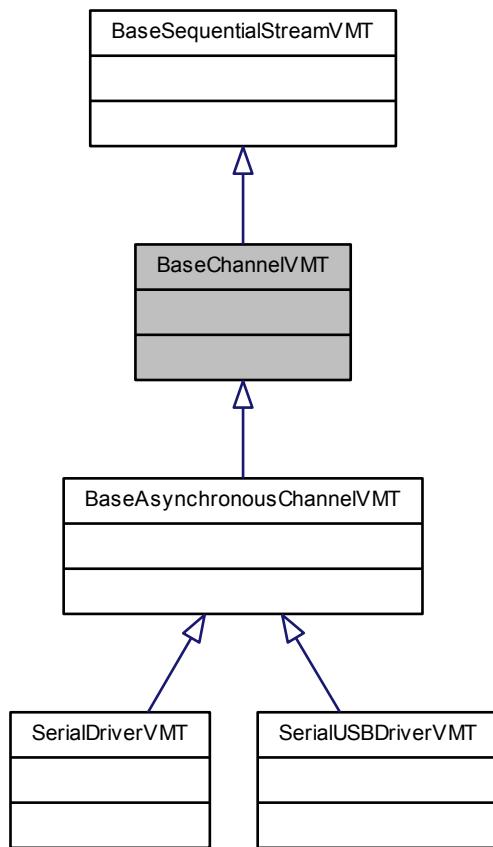
*Virtual Methods Table.*

## 8.9 BaseChannelVMT Struct Reference

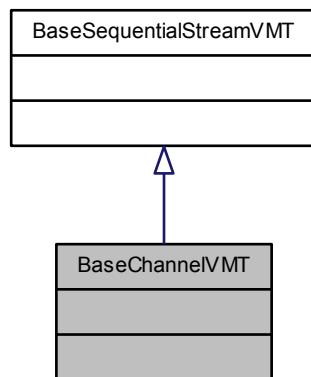
`BaseChannel` virtual methods table.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseChannelVMT:



Collaboration diagram for BaseChannelVMT:



### 8.9.1 Detailed Description

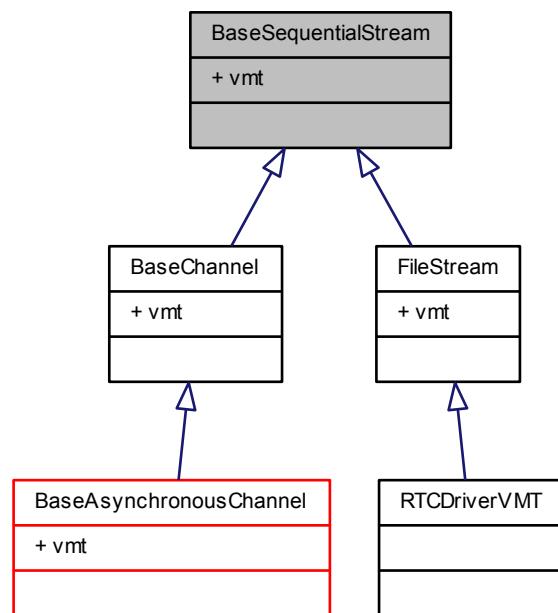
[BaseChannel](#) virtual methods table.

## 8.10 BaseSequentialStream Struct Reference

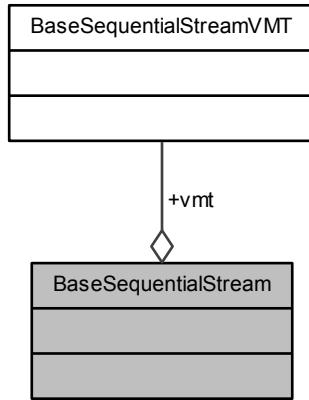
Base stream class.

```
#include <hal_streams.h>
```

Inheritance diagram for BaseSequentialStream:



Collaboration diagram for BaseSequentialStream:



## Data Fields

- const struct [BaseSequentialStreamVMT](#) \* vmt

*Virtual Methods Table.*

### 8.10.1 Detailed Description

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

### 8.10.2 Field Documentation

#### 8.10.2.1 const struct [BaseSequentialStreamVMT](#)\* [BaseSequentialStream::vmt](#)

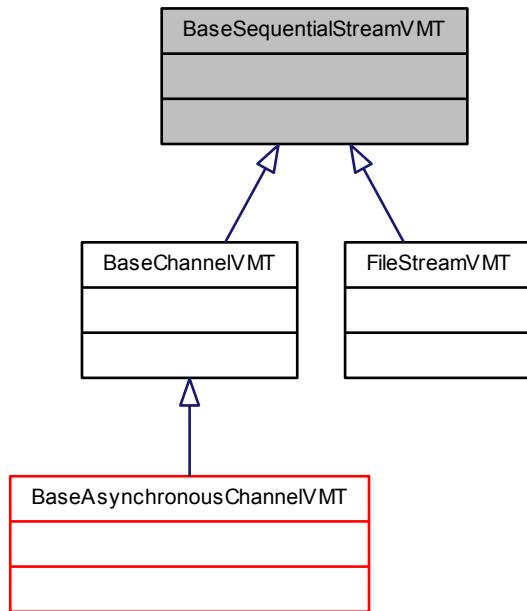
*Virtual Methods Table.*

## 8.11 BaseSequentialStreamVMT Struct Reference

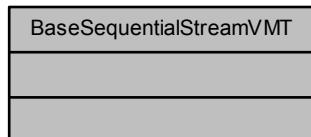
[BaseSequentialStream](#) virtual methods table.

```
#include <hal_streams.h>
```

Inheritance diagram for BaseSequentialStreamVMT:



Collaboration diagram for BaseSequentialStreamVMT:



### 8.11.1 Detailed Description

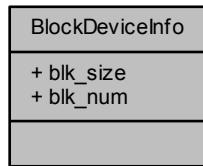
[BaseSequentialStream](#) virtual methods table.

## 8.12 BlockDeviceInfo Struct Reference

Block device info.

```
#include <hal_ioblock.h>
```

Collaboration diagram for BlockDeviceInfo:



## Data Fields

- `uint32_t blk_size`  
*Block size in bytes.*
- `uint32_t blk_num`  
*Total number of blocks.*

### 8.12.1 Detailed Description

Block device info.

### 8.12.2 Field Documentation

#### 8.12.2.1 `uint32_t BlockDeviceInfo::blk_size`

Block size in bytes.

#### 8.12.2.2 `uint32_t BlockDeviceInfo::blk_num`

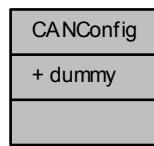
Total number of blocks.

## 8.13 CANConfig Struct Reference

Driver configuration structure.

```
#include <can_lld.h>
```

Collaboration diagram for CANConfig:



### 8.13.1 Detailed Description

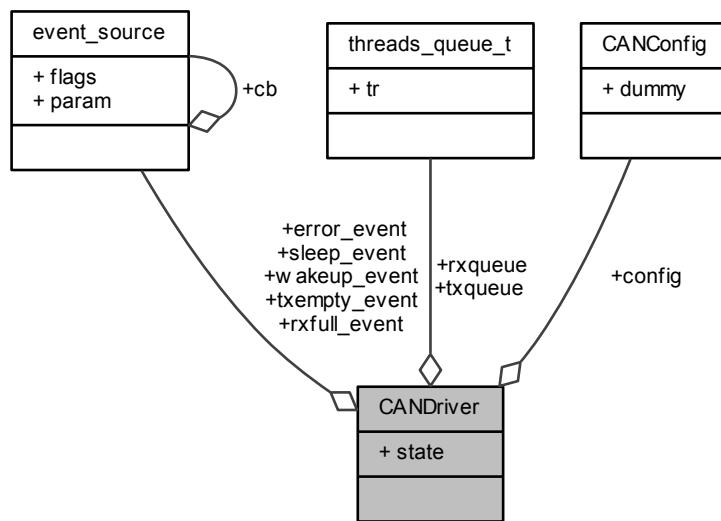
Driver configuration structure.

## 8.14 CANDriver Struct Reference

Structure representing an CAN driver.

```
#include <can_lld.h>
```

Collaboration diagram for CANDriver:



### Data Fields

- `canstate_t state`  
*Driver state.*

- `const CANConfig * config`  
*Current configuration data.*
- `threads_queue_t txqueue`  
*Transmission threads queue.*
- `threads_queue_t rxqueue`  
*Receive threads queue.*
- `event_source_t rxfull_event`  
*One or more frames become available.*
- `event_source_t txempty_event`  
*One or more transmission mailbox become available.*
- `event_source_t error_event`  
*A CAN bus error happened.*
- `event_source_t sleep_event`  
*Entering sleep state event.*
- `event_source_t wakeup_event`  
*Exiting sleep state event.*

### 8.14.1 Detailed Description

Structure representing an CAN driver.

### 8.14.2 Field Documentation

#### 8.14.2.1 canstate\_t CANDriver::state

Driver state.

#### 8.14.2.2 const CANConfig\* CANDriver::config

Current configuration data.

#### 8.14.2.3 threads\_queue\_t CANDriver::txqueue

Transmission threads queue.

#### 8.14.2.4 threads\_queue\_t CANDriver::rxqueue

Receive threads queue.

#### 8.14.2.5 event\_source\_t CANDriver::rxfull\_event

One or more frames become available.

##### Note

After broadcasting this event it will not be broadcasted again until the received frames queue has been completely emptied. It is **not** broadcasted for each received frame. It is responsibility of the application to empty the queue by repeatedly invoking `chReceive()` when listening to this event. This behavior minimizes the interrupt served by the system because CAN traffic.

The flags associated to the listeners will indicate which receive mailboxes become non-empty.

#### 8.14.2.6 `event_source_t CANDriver::txempty_event`

One or more transmission mailbox become available.

##### Note

The flags associated to the listeners will indicate which transmit mailboxes become empty.

#### 8.14.2.7 `event_source_t CANDriver::error_event`

A CAN bus error happened.

##### Note

The flags associated to the listeners will indicate the error(s) that have occurred.

#### 8.14.2.8 `event_source_t CANDriver::sleep_event`

Entering sleep state event.

#### 8.14.2.9 `event_source_t CANDriver::wakeup_event`

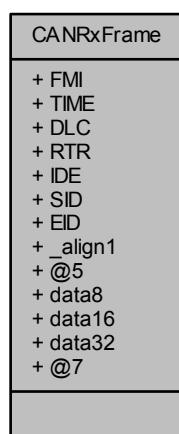
Exiting sleep state event.

## 8.15 CANRxFrame Struct Reference

CAN received frame.

```
#include <can_lld.h>
```

Collaboration diagram for CANRxFrame:



## Data Fields

- `uint8_t FMI`  
*Filter id.*
- `uint16_t TIME`  
*Time stamp.*
- `uint8_t DLC:4`  
*Data length.*
- `uint8_t RTR:1`  
*Frame type.*
- `uint8_t IDE:1`  
*Identifier type.*
- `uint32_t SID:11`  
*Standard identifier.*
- `uint32_t EID:29`  
*Extended identifier.*
- `uint8_t data8 [8]`  
*Frame data.*
- `uint16_t data16 [4]`  
*Frame data.*
- `uint32_t data32 [2]`  
*Frame data.*

### 8.15.1 Detailed Description

CAN received frame.

#### Note

Accessing the frame data as word16 or word32 is not portable because machine data endianness, it can be still useful for a quick filling.

### 8.15.2 Field Documentation

#### 8.15.2.1 `uint8_t CANRxFrame::FMI`

Filter id.

#### 8.15.2.2 `uint16_t CANRxFrame::TIME`

Time stamp.

#### 8.15.2.3 `uint8_t CANRxFrame::DLC`

Data length.

#### 8.15.2.4 `uint8_t CANRxFrame::RTR`

Frame type.

**8.15.2.5 uint8\_t CANRxFrame::IDE**

Identifier type.

**8.15.2.6 uint32\_t CANRxFrame::SID**

Standard identifier.

**8.15.2.7 uint32\_t CANRxFrame::EID**

Extended identifier.

**8.15.2.8 uint8\_t CANRxFrame::data8[8]**

Frame data.

**8.15.2.9 uint16\_t CANRxFrame::data16[4]**

Frame data.

**8.15.2.10 uint32\_t CANRxFrame::data32[2]**

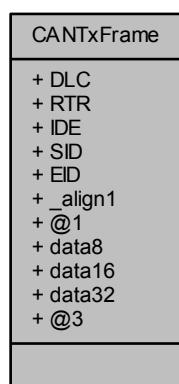
Frame data.

## 8.16 CANTxFrame Struct Reference

CAN transmission frame.

```
#include <can_ll.h>
```

Collaboration diagram for CANTxFrame:



## Data Fields

- `uint8_t DLC:4`  
*Data length.*
- `uint8_t RTR:1`  
*Frame type.*
- `uint8_t IDE:1`  
*Identifier type.*
- `uint32_t SID:11`  
*Standard identifier.*
- `uint32_t EID:29`  
*Extended identifier.*
- `uint8_t data8 [8]`  
*Frame data.*
- `uint16_t data16 [4]`  
*Frame data.*
- `uint32_t data32 [2]`  
*Frame data.*

### 8.16.1 Detailed Description

CAN transmission frame.

#### Note

Accessing the frame data as word16 or word32 is not portable because machine data endianness, it can be still useful for a quick filling.

### 8.16.2 Field Documentation

#### 8.16.2.1 `uint8_t CANTxFrame::DLC`

Data length.

#### 8.16.2.2 `uint8_t CANTxFrame::RTR`

Frame type.

#### 8.16.2.3 `uint8_t CANTxFrame::IDE`

Identifier type.

#### 8.16.2.4 `uint32_t CANTxFrame::SID`

Standard identifier.

#### 8.16.2.5 `uint32_t CANTxFrame::EID`

Extended identifier.

**8.16.2.6 uint8\_t CANTxFrame::data8[8]**

Frame data.

**8.16.2.7 uint16\_t CANTxFrame::data16[4]**

Frame data.

**8.16.2.8 uint32\_t CANTxFrame::data32[2]**

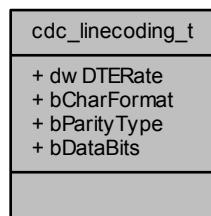
Frame data.

## 8.17 cdc\_linecoding\_t Struct Reference

Type of Line Coding structure.

```
#include <usb_cdc.h>
```

Collaboration diagram for cdc\_linecoding\_t:



### 8.17.1 Detailed Description

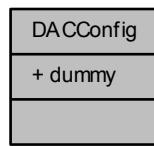
Type of Line Coding structure.

## 8.18 DACConfig Struct Reference

Driver configuration structure.

```
#include <dac_lld.h>
```

Collaboration diagram for DACConfig:



### 8.18.1 Detailed Description

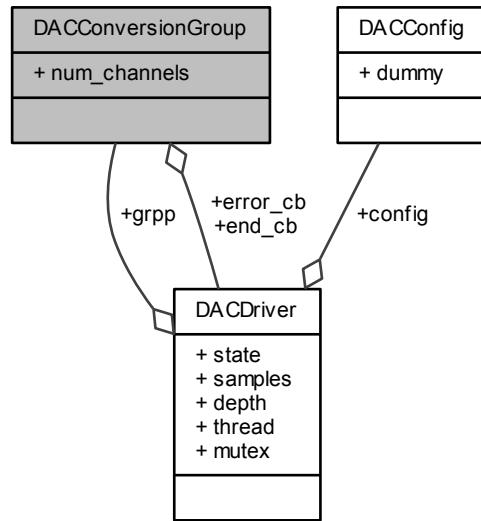
Driver configuration structure.

## 8.19 DACConversionGroup Struct Reference

DAC Conversion group structure.

```
#include <dac_lld.h>
```

Collaboration diagram for DACConversionGroup:



### Data Fields

- `uint32_t num_channels`

*Number of DAC channels.*

- `daccallback_t end_cb`

*Operation complete callback or NULL.*

- `dacerrorcallback_t error_cb`

*Error handling callback or NULL.*

### 8.19.1 Detailed Description

DAC Conversion group structure.

### 8.19.2 Field Documentation

#### 8.19.2.1 `uint32_t DACConversionGroup::num_channels`

Number of DAC channels.

#### 8.19.2.2 `daccallback_t DACConversionGroup::end_cb`

Operation complete callback or NULL.

#### 8.19.2.3 `dacerrorcallback_t DACConversionGroup::error_cb`

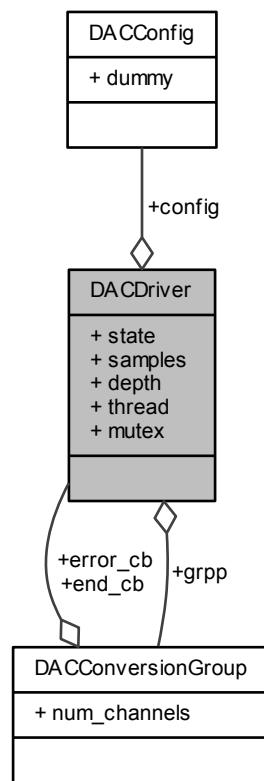
Error handling callback or NULL.

## 8.20 DACDriver Struct Reference

Structure representing a DAC driver.

```
#include <dac_lld.h>
```

Collaboration diagram for DACDriver:



## Data Fields

- **dacstate\_t state**  
*Driver state.*
- **const DACConversionGroup \* grpp**  
*Conversion group.*
- **const dacsample\_t \* samples**  
*Samples buffer pointer.*
- **uint16\_t depth**  
*Samples buffer size.*
- **const DACConfig \* config**  
*Current configuration data.*
- **thread\_reference\_t thread**  
*Waiting thread.*
- **mutex\_t mutex**  
*Mutex protecting the bus.*

### 8.20.1 Detailed Description

Structure representing a DAC driver.

## 8.20.2 Field Documentation

### 8.20.2.1 `dacstate_t` DACDriver::state

Driver state.

### 8.20.2.2 `const DACConversionGroup*` DACDriver::grpp

Conversion group.

### 8.20.2.3 `const dacsample_t*` DACDriver::samples

Samples buffer pointer.

### 8.20.2.4 `uint16_t` DACDriver::depth

Samples buffer size.

### 8.20.2.5 `const DACConfig*` DACDriver::config

Current configuration data.

### 8.20.2.6 `thread_reference_t` DACDriver::thread

Waiting thread.

### 8.20.2.7 `mutex_t` DACDriver::mutex

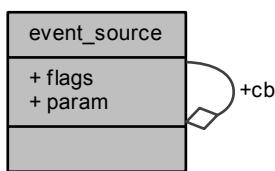
Mutex protecting the bus.

## 8.21 event\_source Struct Reference

Events source object.

```
#include <osal.h>
```

Collaboration diagram for `event_source`:



## Data Fields

- volatile `eventflags_t` `flags`

*Stored event flags.*

- `eventcallback_t` `cb`

*Event source callback.*

- `void *` `param`

*User defined field.*

### 8.21.1 Detailed Description

Events source object.

#### Note

The content of this structure is not part of the API and should not be relied upon. Implementers may define this structure in an entirely different way.

Retrieval and clearing of the flags are not defined in this API and are implementation-dependent.

### 8.21.2 Field Documentation

#### 8.21.2.1 volatile `eventflags_t` `event_source::flags`

Stored event flags.

#### 8.21.2.2 `eventcallback_t` `event_source::cb`

Event source callback.

#### 8.21.2.3 `void*` `event_source::param`

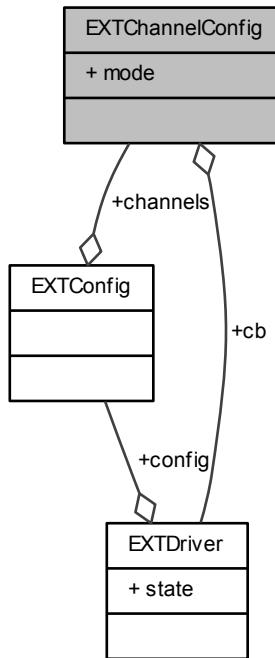
User defined field.

## 8.22 EXTChannelConfig Struct Reference

Channel configuration structure.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTChannelConfig:



## Data Fields

- `uint32_t mode`  
*Channel mode.*
- `extcallback_t cb`  
*Channel callback.*

### 8.22.1 Detailed Description

Channel configuration structure.

### 8.22.2 Field Documentation

#### 8.22.2.1 `uint32_t EXTChannelConfig::mode`

Channel mode.

#### 8.22.2.2 `extcallback_t EXTChannelConfig::cb`

Channel callback.

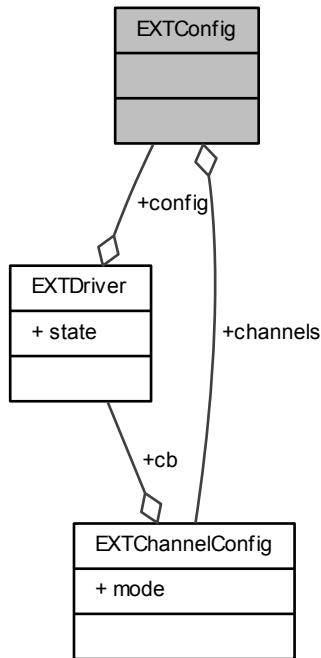
In the STM32 implementation a `NULL` callback pointer is valid and configures the channel as an event sources instead of an interrupt source.

## 8.23 EXTConfig Struct Reference

Driver configuration structure.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTConfig:



### Data Fields

- [EXTChannelConfig channels \[EXT\\_MAX\\_CHANNELS\]](#)

*Channel configurations.*

### 8.23.1 Detailed Description

Driver configuration structure.

#### Note

It could be empty on some architectures.

### 8.23.2 Field Documentation

#### 8.23.2.1 EXTChannelConfig EXTConfig::channels[EXT\_MAX\_CHANNELS]

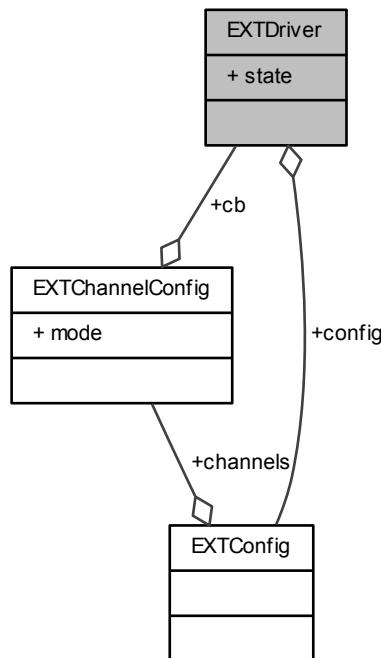
Channel configurations.

## 8.24 EXTDriver Struct Reference

Structure representing an EXT driver.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTDriver:



### Data Fields

- `extstate_t state`  
*Driver state.*
- `const EXTConfig * config`  
*Current configuration data.*

#### 8.24.1 Detailed Description

Structure representing an EXT driver.

#### 8.24.2 Field Documentation

##### 8.24.2.1 `extstate_t EXTDriver::state`

Driver state.

### 8.24.2.2 const EXTConfig\* EXTDriver::config

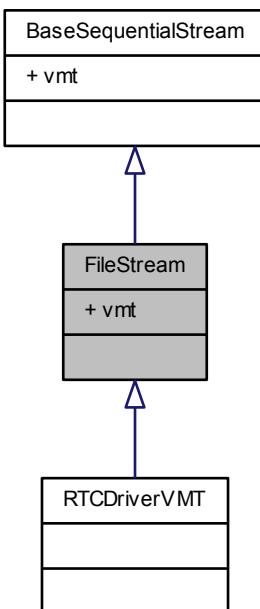
Current configuration data.

## 8.25 FileStream Struct Reference

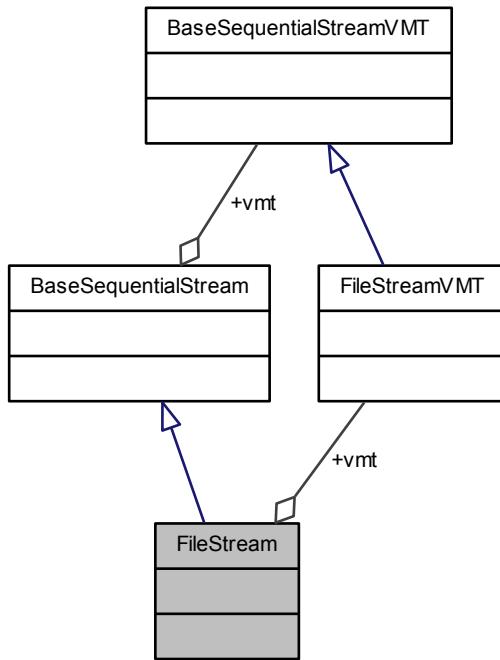
Base file stream class.

```
#include <hal_files.h>
```

Inheritance diagram for FileStream:



Collaboration diagram for FileStream:



## Data Fields

- const struct `FileStreamVMT` \* `vmt`

*Virtual Methods Table.*

### 8.25.1 Detailed Description

Base file stream class.

This class represents a generic file data stream.

### 8.25.2 Field Documentation

#### 8.25.2.1 const struct `FileStreamVMT`\* `FileStream::vmt`

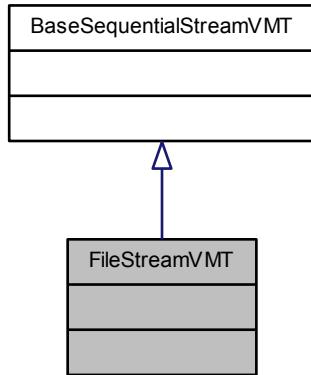
*Virtual Methods Table.*

## 8.26 FileStreamVMT Struct Reference

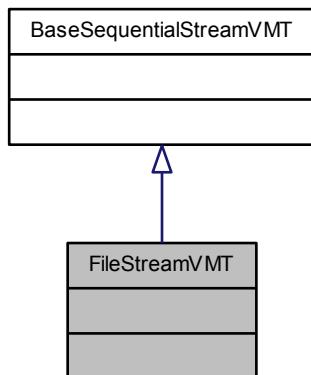
`FileStream` virtual methods table.

```
#include <hal_files.h>
```

Inheritance diagram for FileStreamVMT:



Collaboration diagram for FileStreamVMT:



### 8.26.1 Detailed Description

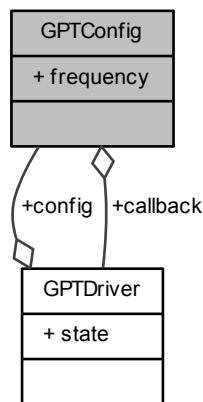
[FileStream](#) virtual methods table.

## 8.27 GPTConfig Struct Reference

Driver configuration structure.

```
#include <gpt_lld.h>
```

Collaboration diagram for GPTConfig:



## Data Fields

- `gptfreq_t frequency`  
*Timer clock in Hz.*
- `gptcallback_t callback`  
*Timer callback pointer.*

### 8.27.1 Detailed Description

Driver configuration structure.

#### Note

It could be empty on some architectures.

### 8.27.2 Field Documentation

#### 8.27.2.1 `gptfreq_t GPTConfig::frequency`

Timer clock in Hz.

#### Note

The low level can use assertions in order to catch invalid frequency specifications.

#### 8.27.2.2 `gptcallback_t GPTConfig::callback`

Timer callback pointer.

#### Note

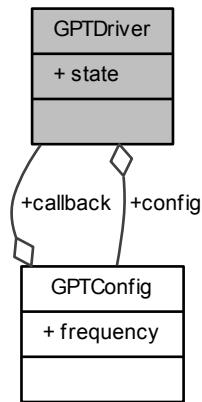
This callback is invoked on GPT counter events.

## 8.28 GPTDriver Struct Reference

Structure representing a GPT driver.

```
#include <gpt_lld.h>
```

Collaboration diagram for GPTDriver:



### Data Fields

- [gptstate\\_t state](#)  
*Driver state.*
- [const GPTConfig \\* config](#)  
*Current configuration data.*

#### 8.28.1 Detailed Description

Structure representing a GPT driver.

#### 8.28.2 Field Documentation

##### 8.28.2.1 gptstate\_t GPTDriver::state

Driver state.

##### 8.28.2.2 const GPTConfig\* GPTDriver::config

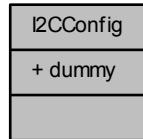
Current configuration data.

## 8.29 I2CConfig Struct Reference

Type of I2C driver configuration structure.

```
#include <i2c_lld.h>
```

Collaboration diagram for I2CConfig:



### 8.29.1 Detailed Description

Type of I2C driver configuration structure.

#### Note

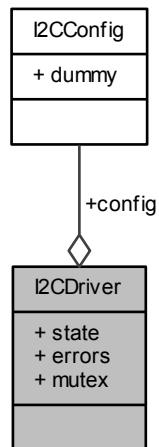
Implementations may extend this structure to contain more, architecture dependent, fields.

## 8.30 I2CDriver Struct Reference

Structure representing an I2C driver.

```
#include <i2c_lld.h>
```

Collaboration diagram for I2CDriver:



## Data Fields

- `i2cstate_t state`

*Driver state.*

- `const I2CConfig * config`

*Current configuration data.*

- `i2cflags_t errors`

*Error flags.*

### 8.30.1 Detailed Description

Structure representing an I2C driver.

### 8.30.2 Field Documentation

#### 8.30.2.1 `i2cstate_t I2CDriver::state`

Driver state.

#### 8.30.2.2 `const I2CConfig* I2CDriver::config`

Current configuration data.

#### 8.30.2.3 `i2cflags_t I2CDriver::errors`

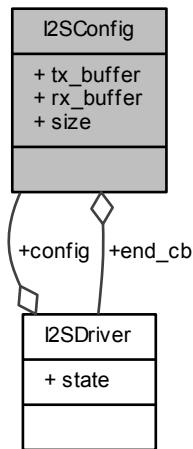
Error flags.

## 8.31 I2SConfig Struct Reference

Driver configuration structure.

```
#include <i2s_ll.h>
```

Collaboration diagram for I2SConfig:



## Data Fields

- `const void * tx_buffer`  
*Transmission buffer pointer.*
- `void * rx_buffer`  
*Receive buffer pointer.*
- `size_t size`  
*TX and RX buffers size as number of samples.*
- `i2scallback_t end_cb`  
*Callback function called during streaming.*

### 8.31.1 Detailed Description

Driver configuration structure.

#### Note

It could be empty on some architectures.

### 8.31.2 Field Documentation

#### 8.31.2.1 `const void* I2SConfig::tx_buffer`

Transmission buffer pointer.

#### Note

Can be `NULL` if TX is not required.

## 8.31.2.2 void\* I2SConfig::rx\_buffer

Receive buffer pointer.

## Note

Can be NULL if RX is not required.

## 8.31.2.3 size\_t I2SConfig::size

TX and RX buffers size as number of samples.

## 8.31.2.4 i2scallback\_t I2SConfig::end\_cb

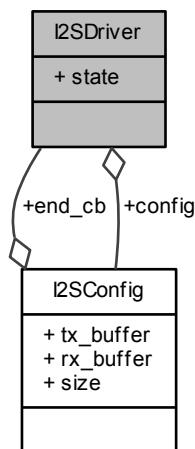
Callback function called during streaming.

## 8.32 I2SDriver Struct Reference

Structure representing an I2S driver.

```
#include <i2s_llld.h>
```

Collaboration diagram for I2SDriver:



## Data Fields

- [i2sstate\\_t state](#)  
*Driver state.*
- const [I2SConfig \\* config](#)  
*Current configuration data.*

### 8.32.1 Detailed Description

Structure representing an I2S driver.

### 8.32.2 Field Documentation

#### 8.32.2.1 i2sstate\_t I2SDriver::state

Driver state.

#### 8.32.2.2 const I2SConfig\* I2SDriver::config

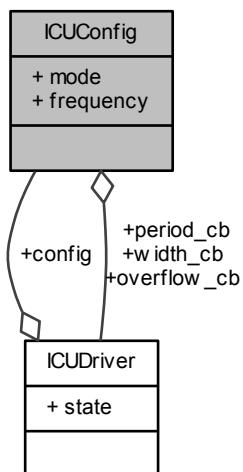
Current configuration data.

## 8.33 ICUConfig Struct Reference

Driver configuration structure.

```
#include <icu_lld.h>
```

Collaboration diagram for ICUConfig:



## Data Fields

- [icemode\\_t mode](#)  
*Driver mode.*
- [icufreq\\_t frequency](#)  
*Timer clock in Hz.*
- [icucallback\\_t width\\_cb](#)  
*Callback for pulse width measurement.*
- [icucallback\\_t period\\_cb](#)

*Callback for cycle period measurement.*

- [icucallback\\_t overflow\\_cb](#)

*Callback for timer overflow.*

### 8.33.1 Detailed Description

Driver configuration structure.

#### Note

It could be empty on some architectures.

### 8.33.2 Field Documentation

#### 8.33.2.1 [icumode\\_t](#) ICUConfig::mode

Driver mode.

#### 8.33.2.2 [icufreq\\_t](#) ICUConfig::frequency

Timer clock in Hz.

#### Note

The low level can use assertions in order to catch invalid frequency specifications.

#### 8.33.2.3 [icucallback\\_t](#) ICUConfig::width\_cb

Callback for pulse width measurement.

#### 8.33.2.4 [icucallback\\_t](#) ICUConfig::period\_cb

Callback for cycle period measurement.

#### 8.33.2.5 [icucallback\\_t](#) ICUConfig::overflow\_cb

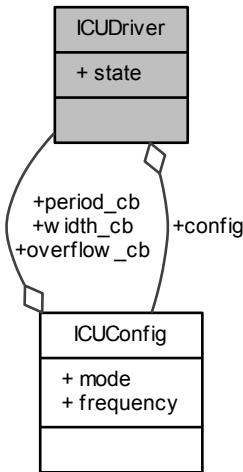
Callback for timer overflow.

## 8.34 ICUDriver Struct Reference

Structure representing an ICU driver.

```
#include <icu_lld.h>
```

Collaboration diagram for ICUDriver:



## Data Fields

- `icustate_t state`  
*Driver state.*
- `const ICUConfig * config`  
*Current configuration data.*

### 8.34.1 Detailed Description

Structure representing an ICU driver.

### 8.34.2 Field Documentation

#### 8.34.2.1 `icustate_t ICUDriver::state`

Driver state.

#### 8.34.2.2 `const ICUConfig* ICUDriver::config`

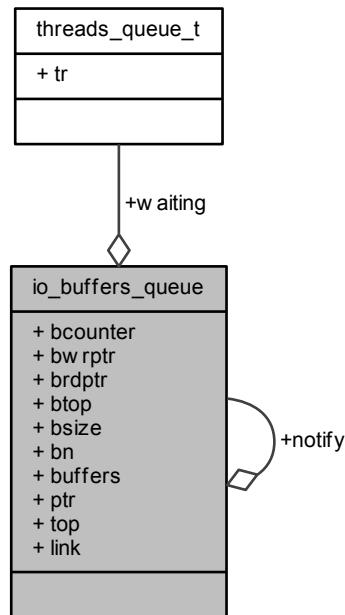
Current configuration data.

## 8.35 io\_buffers\_queue Struct Reference

Structure of a generic buffers queue.

```
#include <hal_buffers.h>
```

Collaboration diagram for io\_buffers\_queue:



## Data Fields

- **threads\_queue\_t waiting**  
*Queue of waiting threads.*
- volatile size\_t **bcounter**  
*Active buffers counter.*
- uint8\_t \* **bwrptr**  
*Buffer write pointer.*
- uint8\_t \* **brdptr**  
*Buffer read pointer.*
- uint8\_t \* **btop**  
*Pointer to the buffers boundary.*
- size\_t **bsize**  
*Size of buffers.*
- size\_t **bn**  
*Number of buffers.*
- uint8\_t \* **buffers**  
*Queue of buffer objects.*
- uint8\_t \* **ptr**  
*Pointer for R/W sequential access.*
- uint8\_t \* **top**  
*Boundary for R/W sequential access.*
- bqnotify\_t **notify**  
*Data notification callback.*
- void \* **link**  
*Application defined field.*

### 8.35.1 Detailed Description

Structure of a generic buffers queue.

### 8.35.2 Field Documentation

#### 8.35.2.1 `threads_queue_t io_buffers_queue::waiting`

Queue of waiting threads.

#### 8.35.2.2 `volatile size_t io_buffers_queue::bcounter`

Active buffers counter.

#### 8.35.2.3 `uint8_t* io_buffers_queue::bwrptr`

Buffer write pointer.

#### 8.35.2.4 `uint8_t* io_buffers_queue::brdptr`

Buffer read pointer.

#### 8.35.2.5 `uint8_t* io_buffers_queue::btop`

Pointer to the buffers boundary.

#### 8.35.2.6 `size_t io_buffers_queue::bsize`

Size of buffers.

#### Note

The buffer size must be not lower than `sizeof(size_t) + 2` because the first bytes are used to store the used size of the buffer.

#### 8.35.2.7 `size_t io_buffers_queue::bn`

Number of buffers.

#### 8.35.2.8 `uint8_t* io_buffers_queue::buffers`

Queue of buffer objects.

#### 8.35.2.9 `uint8_t* io_buffers_queue::ptr`

Pointer for R/W sequential access.

#### Note

It is `NULL` if a new buffer must be fetched from the queue.

8.35.2.10 `uint8_t* io_buffers_queue::top`

Boundary for R/W sequential access.

8.35.2.11 `bqnotify_t io_buffers_queue::notify`

Data notification callback.

8.35.2.12 `void* io_buffers_queue::link`

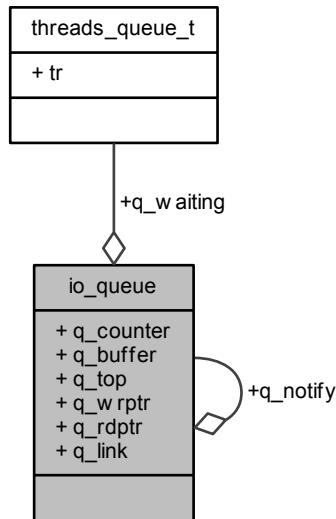
Application defined field.

## 8.36 io\_queue Struct Reference

Generic I/O queue structure.

```
#include <hal_queues.h>
```

Collaboration diagram for `io_queue`:



### Data Fields

- `threads_queue_t q_waiting`

*Queue of waiting threads.*

- volatile size\_t `q_counter`

*Resources counter.*

- `uint8_t * q_buffer`

*Pointer to the queue buffer.*

- `uint8_t * q_top`

- `uint8_t * q_wptr`  
*Write pointer.*
- `uint8_t * q_rdptr`  
*Read pointer.*
- `qnotify_t q_notify`  
*Data notification callback.*
- `void * q_link`  
*Application defined field.*

### 8.36.1 Detailed Description

Generic I/O queue structure.

This structure represents a generic Input or Output asymmetrical queue. The queue is asymmetrical because one end is meant to be accessed from a thread context, and thus can be blocking, the other end is accessible from interrupt handlers or from within a kernel lock zone and is non-blocking.

### 8.36.2 Field Documentation

#### 8.36.2.1 `threads_queue_t io_queue::q_waiting`

Queue of waiting threads.

#### 8.36.2.2 `volatile size_t io_queue::q_counter`

Resources counter.

#### 8.36.2.3 `uint8_t* io_queue::q_buffer`

Pointer to the queue buffer.

#### 8.36.2.4 `uint8_t* io_queue::q_top`

Pointer to the first location after the buffer.

#### 8.36.2.5 `uint8_t* io_queue::q_wptr`

Write pointer.

#### 8.36.2.6 `uint8_t* io_queue::q_rdptr`

Read pointer.

#### 8.36.2.7 `qnotify_t io_queue::q_notify`

Data notification callback.

#### 8.36.2.8 `void* io_queue::q_link`

Application defined field.

## 8.37 IOBus Struct Reference

I/O bus descriptor.

```
#include <pal.h>
```

Collaboration diagram for IOBus:



### Data Fields

- **ioportid\_t portid**  
*Port identifier.*
- **ioportmask\_t mask**  
*Bus mask aligned to port bit 0.*
- **uint\_fast8\_t offset**  
*Offset, within the port, of the least significant bit of the bus.*

### 8.37.1 Detailed Description

I/O bus descriptor.

This structure describes a group of contiguous digital I/O lines that have to be handled as bus.

#### Note

I/O operations on a bus do not affect I/O lines on the same port but not belonging to the bus.

### 8.37.2 Field Documentation

#### 8.37.2.1 ioportid\_t IOBus::portid

Port identifier.

#### 8.37.2.2 ioportmask\_t IOBus::mask

Bus mask aligned to port bit 0.

#### Note

The bus mask implicitly define the bus width. A logic AND is performed on the bus data.

### 8.37.2.3 uint\_fast8\_t IOBus::offset

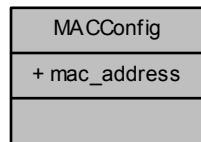
Offset, within the port, of the least significant bit of the bus.

## 8.38 MACConfig Struct Reference

Driver configuration structure.

```
#include <mac_lld.h>
```

Collaboration diagram for MACConfig:



### Data Fields

- `uint8_t * mac_address`

*MAC address.*

### 8.38.1 Detailed Description

Driver configuration structure.

### 8.38.2 Field Documentation

#### 8.38.2.1 `uint8_t* MACConfig::mac_address`

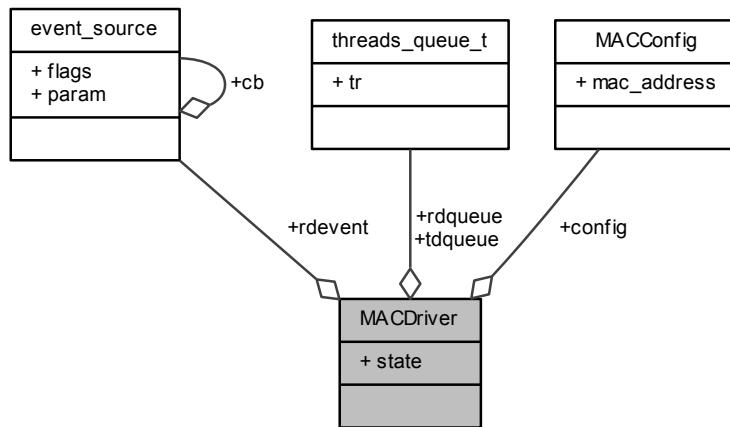
MAC address.

## 8.39 MACDriver Struct Reference

Structure representing a MAC driver.

```
#include <mac_lld.h>
```

Collaboration diagram for MACDriver:



## Data Fields

- **macstate\_t state**  
*Driver state.*
- **const MACConfig \* config**  
*Current configuration data.*
- **threads\_queue\_t tdqueue**  
*Transmit semaphore.*
- **threads\_queue\_t rdqueue**  
*Receive semaphore.*
- **event\_source\_t rdevent**  
*Receive event.*

### 8.39.1 Detailed Description

Structure representing a MAC driver.

### 8.39.2 Field Documentation

#### 8.39.2.1 **macstate\_t MACDriver::state**

Driver state.

#### 8.39.2.2 **const MACConfig\* MACDriver::config**

Current configuration data.

#### 8.39.2.3 **threads\_queue\_t MACDriver::tdqueue**

Transmit semaphore.

#### 8.39.2.4 threads\_queue\_t MACDriver::rdqueue

Receive semaphore.

#### 8.39.2.5 event\_source\_t MACDriver::rdevent

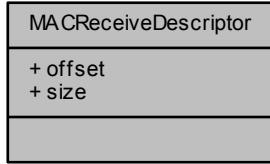
Receive event.

## 8.40 MACReceiveDescriptor Struct Reference

Structure representing a receive descriptor.

```
#include <mac_lld.h>
```

Collaboration diagram for MACReceiveDescriptor:



### Data Fields

- size\_t **offset**

*Current read offset.*

- size\_t **size**

*Available data size.*

### 8.40.1 Detailed Description

Structure representing a receive descriptor.

### 8.40.2 Field Documentation

#### 8.40.2.1 size\_t MACReceiveDescriptor::offset

Current read offset.

#### 8.40.2.2 size\_t MACReceiveDescriptor::size

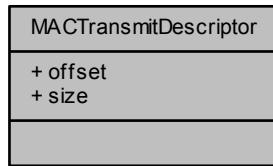
Available data size.

## 8.41 MACTransmitDescriptor Struct Reference

Structure representing a transmit descriptor.

```
#include <mac_lld.h>
```

Collaboration diagram for MACTransmitDescriptor:



### Data Fields

- `size_t offset`  
*Current write offset.*
- `size_t size`  
*Available space size.*

#### 8.41.1 Detailed Description

Structure representing a transmit descriptor.

#### 8.41.2 Field Documentation

##### 8.41.2.1 `size_t MACTransmitDescriptor::offset`

Current write offset.

##### 8.41.2.2 `size_t MACTransmitDescriptor::size`

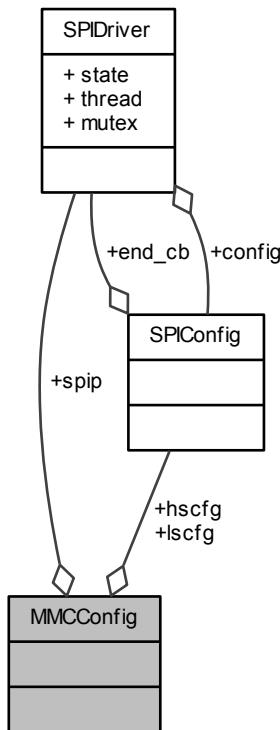
Available space size.

## 8.42 MMCConfig Struct Reference

MMC/SD over SPI driver configuration structure.

```
#include <mmc_spi.h>
```

Collaboration diagram for MMCConfig:



## Data Fields

- **SPIDriver \* spip**  
*SPI driver associated to this MMC driver.*
- const **SPIConfig \* lscfg**  
*SPI low speed configuration used during initialization.*
- const **SPIConfig \* hscfg**  
*SPI high speed configuration used during transfers.*

### 8.42.1 Detailed Description

MMC/SD over SPI driver configuration structure.

### 8.42.2 Field Documentation

#### 8.42.2.1 SPIDriver\* MMCConfig::spip

SPI driver associated to this MMC driver.

#### 8.42.2.2 const SPIConfig\* MMCConfig::lscfg

SPI low speed configuration used during initialization.

#### 8.42.2.3 const SPIConfig\* MMCConfig::hscfg

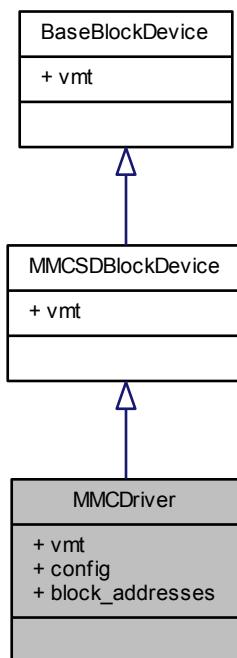
SPI high speed configuration used during transfers.

## 8.43 MMCDriver Struct Reference

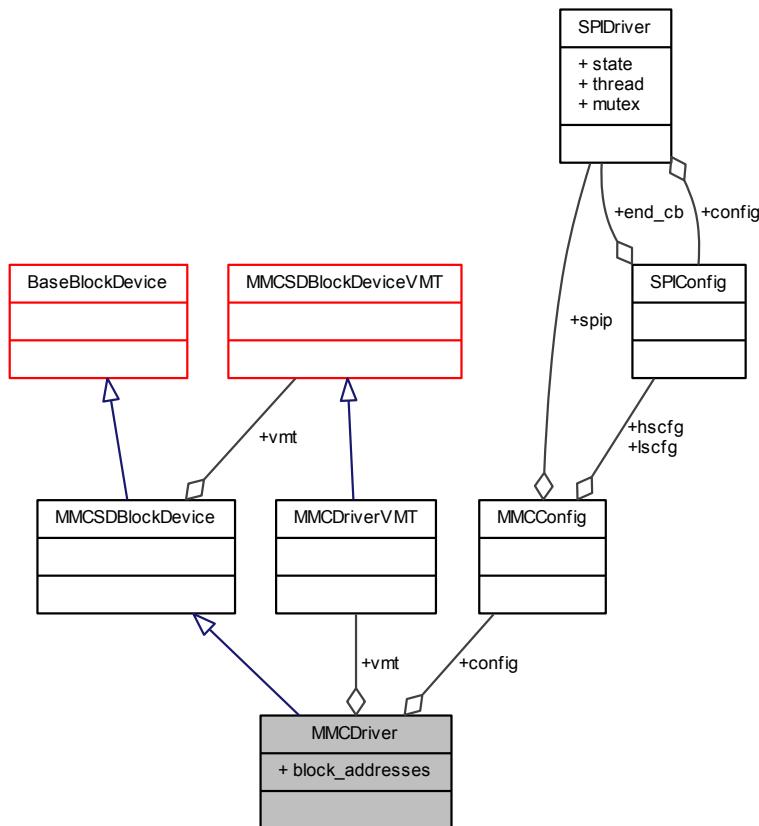
Structure representing a MMC/SD over SPI driver.

```
#include <mmc_spi.h>
```

Inheritance diagram for MMCDriver:



Collaboration diagram for MMCDDriver:



## Data Fields

- const struct `MMCDDriverVMT` \* `vmt`  
*Virtual Methods Table.*
- `_mmcsd_block_device_data` const `MMCCConfig` \* `config`  
*Current configuration data.*

### 8.43.1 Detailed Description

Structure representing a MMC/SD over SPI driver.

### 8.43.2 Field Documentation

#### 8.43.2.1 const struct `MMCDDriverVMT`\* `MMCDDriver::vmt`

Virtual Methods Table.

#### 8.43.2.2 \_mmcsd\_block\_device\_data const MMCConfig\* MMCDriver::config

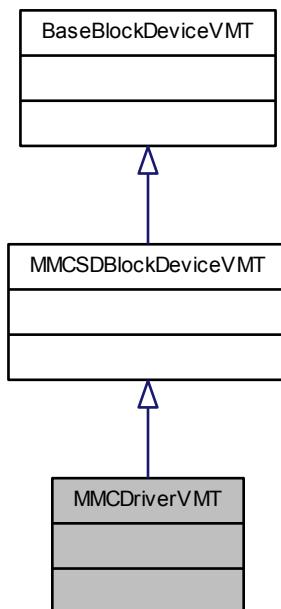
Current configuration data.

## 8.44 MMCDriverVMT Struct Reference

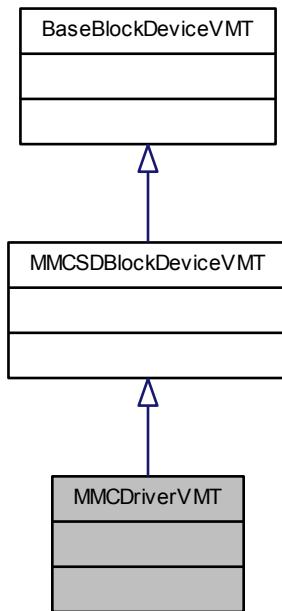
[MMCDriver](#) virtual methods table.

```
#include <mmc_spi.h>
```

Inheritance diagram for MMCDriverVMT:



Collaboration diagram for MMCDriverVMT:



#### 8.44.1 Detailed Description

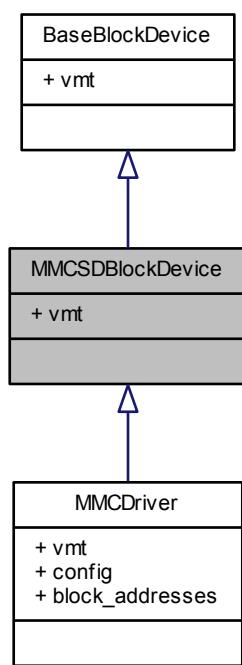
[MMCDriver](#) virtual methods table.

### 8.45 MMCSDBlockDevice Struct Reference

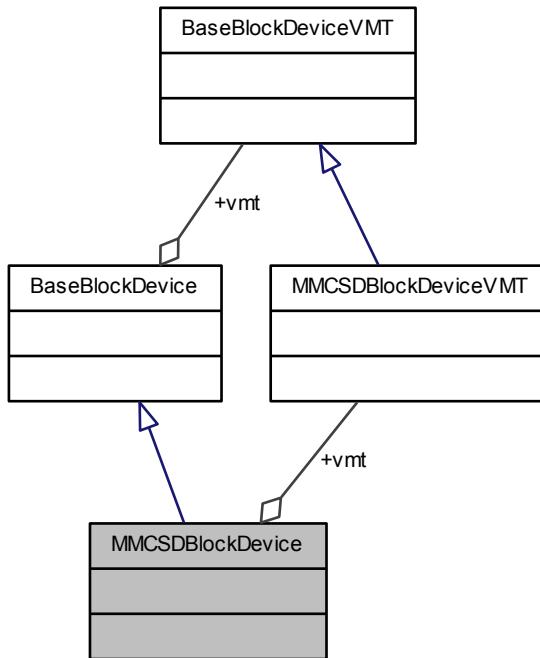
MCC/SD block device class.

```
#include <hal_mmc.h>
```

Inheritance diagram for MMCSDBlockDevice:



Collaboration diagram for MMCSDBlockDevice:



## Data Fields

- const struct `MMCSDBlockDeviceVMT` \* `vmt`

*Virtual Methods Table.*

### 8.45.1 Detailed Description

MCC/SD block device class.

This class represents a, block-accessible, MMC/SD device.

### 8.45.2 Field Documentation

#### 8.45.2.1 const struct `MMCSDBlockDeviceVMT`\* `MMCSDBlockDevice::vmt`

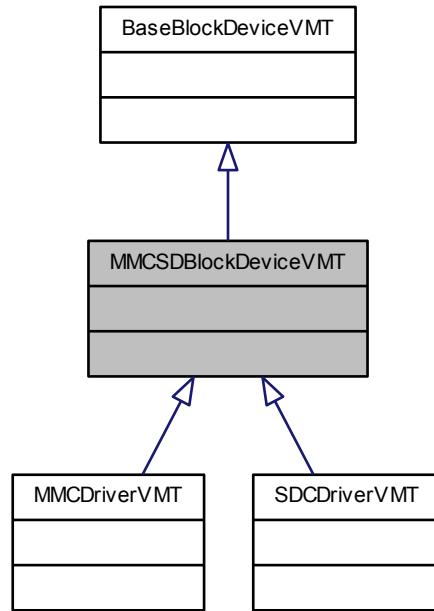
*Virtual Methods Table.*

## 8.46 MMCSDBlockDeviceVMT Struct Reference

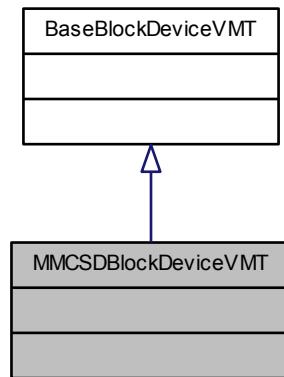
`MMCSDBlockDevice` virtual methods table.

```
#include <hal_mmcblk.h>
```

Inheritance diagram for MMCSDBlockDeviceVMT:



Collaboration diagram for MMCSDBlockDeviceVMT:



#### 8.46.1 Detailed Description

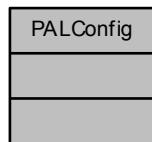
[MMCSDBlockDevice](#) virtual methods table.

## 8.47 PALConfig Struct Reference

Generic I/O ports static initializer.

```
#include <pal_lld.h>
```

Collaboration diagram for PALConfig:



### 8.47.1 Detailed Description

Generic I/O ports static initializer.

An instance of this structure must be passed to [palInit\(\)](#) at system startup time in order to initialize the digital I/O subsystem. This represents only the initial setup, specific pads or whole ports can be reprogrammed at later time.

#### Note

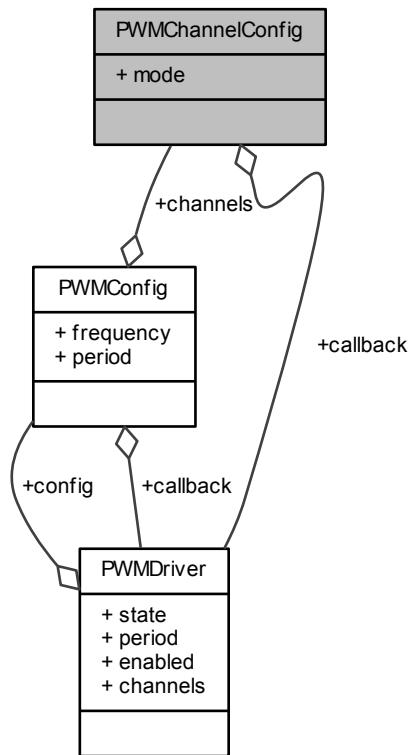
Implementations may extend this structure to contain more, architecture dependent, fields.

## 8.48 PWMChannelConfig Struct Reference

Type of a PWM driver channel configuration structure.

```
#include <pwm_lld.h>
```

Collaboration diagram for PWMChannelConfig:



## Data Fields

- `pwmmode_t mode`  
*Channel active logic level.*
- `pwmcallback_t callback`  
*Channel callback pointer.*

### 8.48.1 Detailed Description

Type of a PWM driver channel configuration structure.

### 8.48.2 Field Documentation

#### 8.48.2.1 `pwmmode_t PWMChannelConfig::mode`

Channel active logic level.

#### 8.48.2.2 `pwmcallback_t PWMChannelConfig::callback`

Channel callback pointer.

**Note**

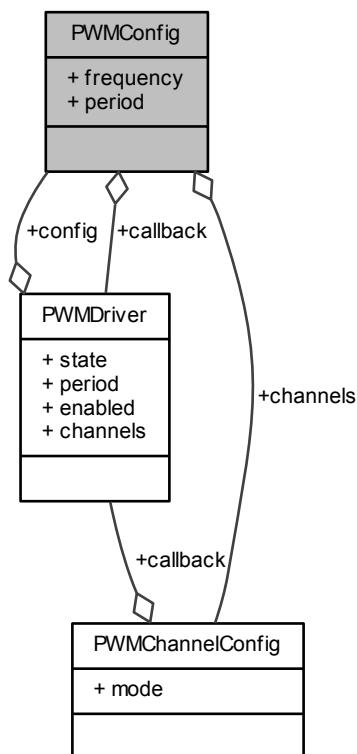
This callback is invoked on the channel compare event. If set to NULL then the callback is disabled.

## 8.49 PWMConfig Struct Reference

Type of a PWM driver configuration structure.

```
#include <pwm_ll.h>
```

Collaboration diagram for PWMConfig:



### Data Fields

- `uint32_t frequency`  
*Timer clock in Hz.*
- `pwmcnt_t period`  
*PWM period in ticks.*
- `pwmcallback_t callback`  
*Periodic callback pointer.*
- `PWMChannelConfig channels [PWM_CHANNELS]`  
*Channels configurations.*

### 8.49.1 Detailed Description

Type of a PWM driver configuration structure.

### 8.49.2 Field Documentation

#### 8.49.2.1 `uint32_t PWMConfig::frequency`

Timer clock in Hz.

##### Note

The low level can use assertions in order to catch invalid frequency specifications.

#### 8.49.2.2 `pwmcnt_t PWMConfig::period`

PWM period in ticks.

##### Note

The low level can use assertions in order to catch invalid period specifications.

#### 8.49.2.3 `pwmcallback_t PWMConfig::callback`

Periodic callback pointer.

##### Note

This callback is invoked on PWM counter reset. If set to `NULL` then the callback is disabled.

#### 8.49.2.4 `PWMChannelConfig PWMConfig::channels[PWM_CHANNELS]`

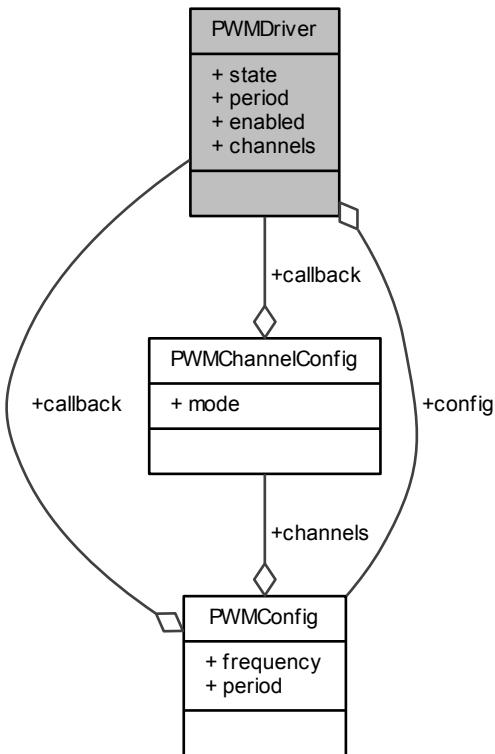
Channels configurations.

## 8.50 PWMDriver Struct Reference

Structure representing a PWM driver.

```
#include <pwm_lld.h>
```

Collaboration diagram for PWMDriver:



## Data Fields

- `pwmstate_t state`  
*Driver state.*
- `const PWMConfig * config`  
*Current driver configuration data.*
- `pwmcnt_t period`  
*Current PWM period in ticks.*
- `pwmchnmsk_t enabled`  
*Mask of the enabled channels.*
- `pwmchannel_t channels`  
*Number of channels in this instance.*

### 8.50.1 Detailed Description

Structure representing a PWM driver.

### 8.50.2 Field Documentation

**8.50.2.1 pwmstate\_t PWMDriver::state**

Driver state.

**8.50.2.2 const PWMConfig\* PWMDriver::config**

Current driver configuration data.

**8.50.2.3 pwmcnt\_t PWMDriver::period**

Current PWM period in ticks.

**8.50.2.4 pwmchnmsk\_t PWMDriver::enabled**

Mask of the enabled channels.

**8.50.2.5 pwmchannel\_t PWMDriver::channels**

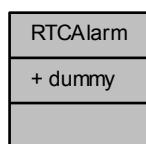
Number of channels in this instance.

## 8.51 RTCAlarm Struct Reference

Type of a structure representing an RTC alarm time stamp.

```
#include <rtc_llld.h>
```

Collaboration diagram for RTCAlarm:



### 8.51.1 Detailed Description

Type of a structure representing an RTC alarm time stamp.

## 8.52 RTCDateTime Struct Reference

Type of a structure representing an RTC date/time stamp.

```
#include <rtc.h>
```

Collaboration diagram for RTCDateTime:



## Data Fields

- `uint32_t year`: 8  
*Years since 1980.*
- `uint32_t month`: 4  
*Months 1..12.*
- `uint32_t dstflag`: 1  
*DST correction flag.*
- `uint32_t dayofweek`: 3  
*Day of week 1..7.*
- `uint32_t day`: 5  
*Day of the month 1..31.*
- `uint32_t millisecond`: 27  
*Milliseconds since midnight.*

### 8.52.1 Detailed Description

Type of a structure representing an RTC date/time stamp.

### 8.52.2 Field Documentation

#### 8.52.2.1 `uint32_t RTCDateTime::year`

*Years since 1980.*

#### 8.52.2.2 `uint32_t RTCDateTime::month`

*Months 1..12.*

#### 8.52.2.3 `uint32_t RTCDateTime::dstflag`

*DST correction flag.*

## 8.52.2.4 uint32\_t RTCDateTime::dayofweek

Day of week 1..7.

## 8.52.2.5 uint32\_t RTCDateTime::day

Day of the month 1..31.

## 8.52.2.6 uint32\_t RTCDateTime::millisecond

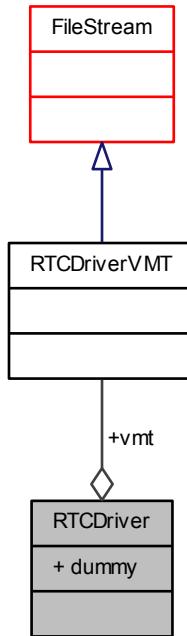
Milliseconds since midnight.

## 8.53 RTCDriver Struct Reference

Structure representing an RTC driver.

```
#include <rtc_llld.h>
```

Collaboration diagram for RTCDriver:



### Data Fields

- const struct [RTCDriverVMT](#) \* **vmt**

*Virtual Methods Table.*

### 8.53.1 Detailed Description

Structure representing an RTC driver.

### 8.53.2 Field Documentation

#### 8.53.2.1 const struct RTCDriverVMT\* RTCDriver::vmt

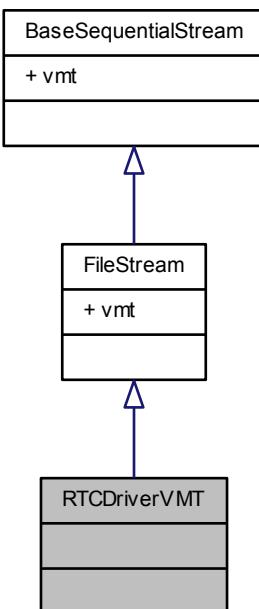
Virtual Methods Table.

## 8.54 RTCDriverVMT Struct Reference

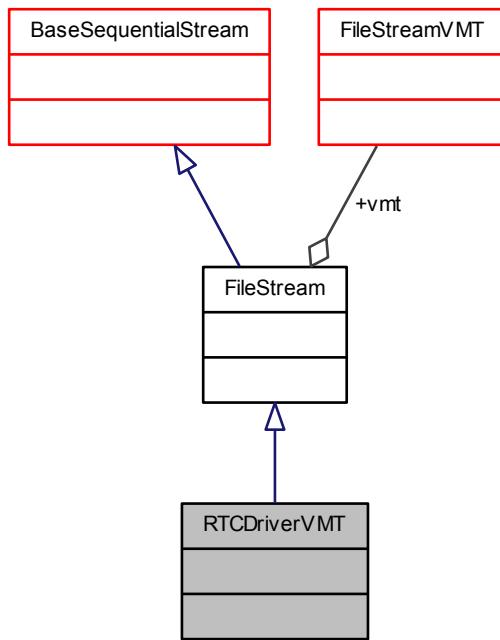
[RTCDriver](#) virtual methods table.

```
#include <rtc_lld.h>
```

Inheritance diagram for RTCDriverVMT:



Collaboration diagram for RTCDriverVMT:



## Additional Inherited Members

### 8.54.1 Detailed Description

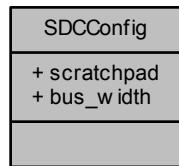
[RTCDriver](#) virtual methods table.

## 8.55 SDCCConfig Struct Reference

Driver configuration structure.

```
#include <sdc_lld.h>
```

Collaboration diagram for SDCCConfig:



## Data Fields

- `uint8_t * scratchpad`  
*Working area for memory consuming operations.*
- `sdcbusmode_t bus_width`  
*Bus width.*

### 8.55.1 Detailed Description

Driver configuration structure.

#### Note

It could be empty on some architectures.

### 8.55.2 Field Documentation

#### 8.55.2.1 `uint8_t* SDCCConfig::scratchpad`

Working area for memory consuming operations.

#### Note

It is mandatory for detecting MMC cards bigger than 2GB else it can be NULL.  
Memory pointed by this buffer is only used by `sdcConnect ()`, afterward it can be reused for other purposes.

#### 8.55.2.2 `sdcbusmode_t SDCCConfig::bus_width`

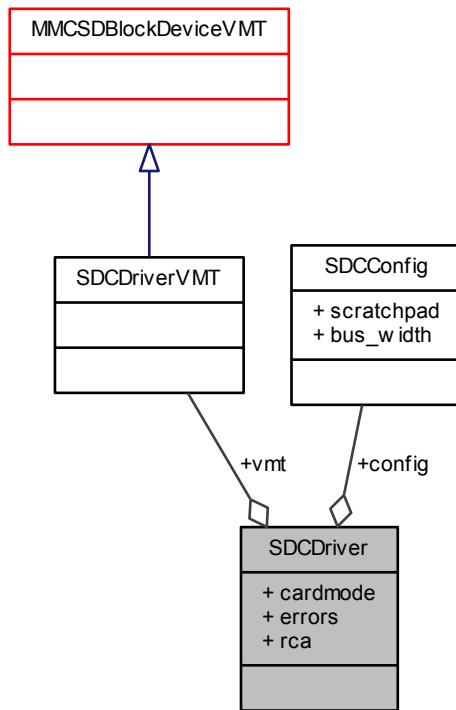
Bus width.

## 8.56 SDCDriver Struct Reference

Structure representing an SDC driver.

```
#include <sdc_lld.h>
```

Collaboration diagram for SDCDriver:



## Data Fields

- const struct [SDCDriverVMT](#) \* vmt  
*Virtual Methods Table.*
- [\\_mmcsd\\_block\\_device\\_data](#) const [SDCCConfig](#) \* config  
*Current configuration data.*
- [sdcmode\\_t](#) cardmode  
*Various flags regarding the mounted card.*
- [sdcflags\\_t](#) errors  
*Errors flags.*
- uint32\_t rca  
*Card RCA.*

### 8.56.1 Detailed Description

Structure representing an SDC driver.

### 8.56.2 Field Documentation

#### 8.56.2.1 const struct [SDCDriverVMT](#)\* SDCDriver::vmt

*Virtual Methods Table.*

#### 8.56.2.2 `_mmcsd_block_device_data const SDCCConfig* SDCDriver::config`

Current configuration data.

#### 8.56.2.3 `sdcmode_t SDCDriver::cardmode`

Various flags regarding the mounted card.

#### 8.56.2.4 `sdcflags_t SDCDriver::errors`

Errors flags.

#### 8.56.2.5 `uint32_t SDCDriver::rca`

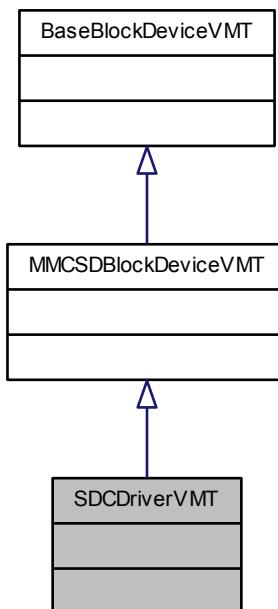
Card RCA.

## 8.57 SDCDriverVMT Struct Reference

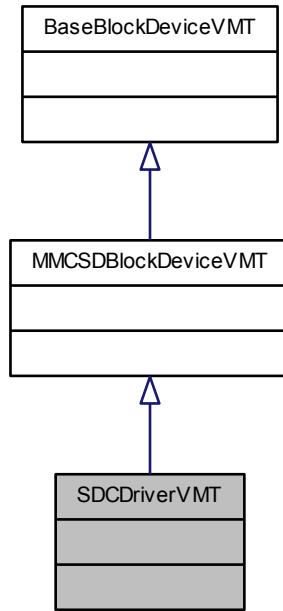
`SDCDriver` virtual methods table.

```
#include <sdc_lld.h>
```

Inheritance diagram for SDCDriverVMT:



Collaboration diagram for SDCDriverVMT:



### 8.57.1 Detailed Description

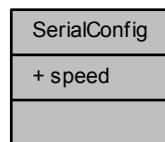
[SDCDriver](#) virtual methods table.

## 8.58 SerialConfig Struct Reference

PLATFORM Serial Driver configuration structure.

```
#include <serial_lld.h>
```

Collaboration diagram for SerialConfig:



## Data Fields

- `uint32_t speed`

*Bit rate.*

### 8.58.1 Detailed Description

PLATFORM Serial Driver configuration structure.

An instance of this structure must be passed to `sdStart()` in order to configure and start a serial driver operations.

#### Note

This structure content is architecture dependent, each driver implementation defines its own version and the custom static initializers.

### 8.58.2 Field Documentation

#### 8.58.2.1 `uint32_t SerialConfig::speed`

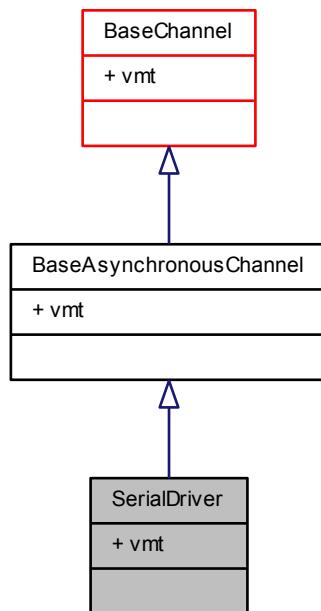
Bit rate.

## 8.59 SerialDriver Struct Reference

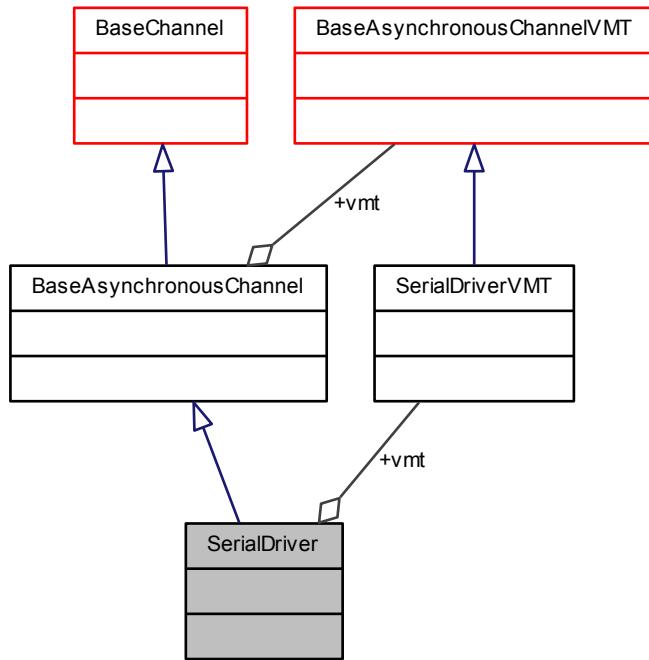
Full duplex serial driver class.

```
#include <serial.h>
```

Inheritance diagram for SerialDriver:



Collaboration diagram for SerialDriver:



## Data Fields

- const struct `SerialDriverVMT` \* `vmt`

*Virtual Methods Table.*

### 8.59.1 Detailed Description

Full duplex serial driver class.

This class extends `BaseAsynchronousChannel` by adding physical I/O queues.

### 8.59.2 Field Documentation

#### 8.59.2.1 const struct `SerialDriverVMT`\* `SerialDriver::vmt`

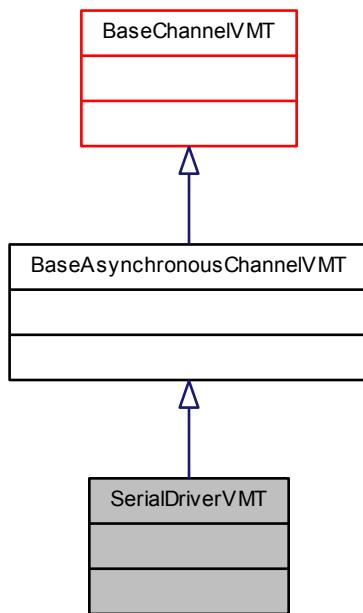
*Virtual Methods Table.*

## 8.60 SerialDriverVMT Struct Reference

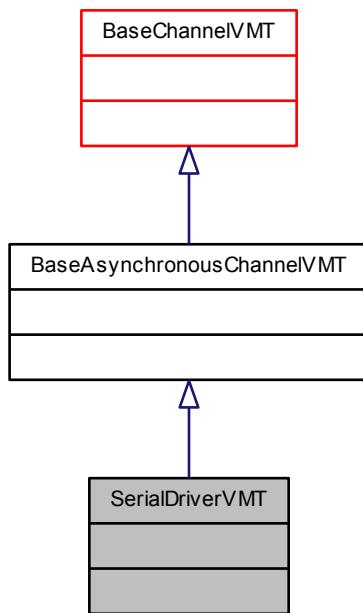
`SerialDriver` virtual methods table.

```
#include <serial.h>
```

Inheritance diagram for SerialDriverVMT:



Collaboration diagram for SerialDriverVMT:



### 8.60.1 Detailed Description

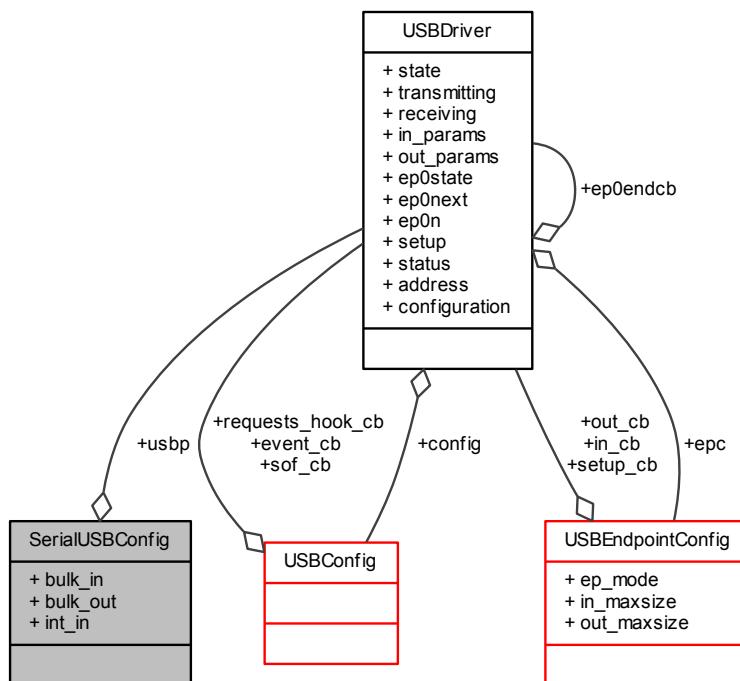
[SerialDriver](#) virtual methods table.

## 8.61 SerialUSBConfig Struct Reference

Serial over USB Driver configuration structure.

```
#include <serial_usb.h>
```

Collaboration diagram for SerialUSBConfig:



### Data Fields

- [USBDriver \\* usbp](#)  
*USB driver to use.*
- [usbep\\_t bulk\\_in](#)  
*Bulk IN endpoint used for outgoing data transfer.*
- [usbep\\_t bulk\\_out](#)  
*Bulk OUT endpoint used for incoming data transfer.*
- [usbep\\_t int\\_in](#)  
*Interrupt IN endpoint used for notifications.*

### 8.61.1 Detailed Description

Serial over USB Driver configuration structure.

An instance of this structure must be passed to `sduStart()` in order to configure and start the driver operations.

## 8.61.2 Field Documentation

### 8.61.2.1 `USBDriver* SerialUSBConfig::usbp`

USB driver to use.

### 8.61.2.2 `usbep_t SerialUSBConfig::bulk_in`

Bulk IN endpoint used for outgoing data transfer.

### 8.61.2.3 `usbep_t SerialUSBConfig::bulk_out`

Bulk OUT endpoint used for incoming data transfer.

### 8.61.2.4 `usbep_t SerialUSBConfig::int_in`

Interrupt IN endpoint used for notifications.

#### Note

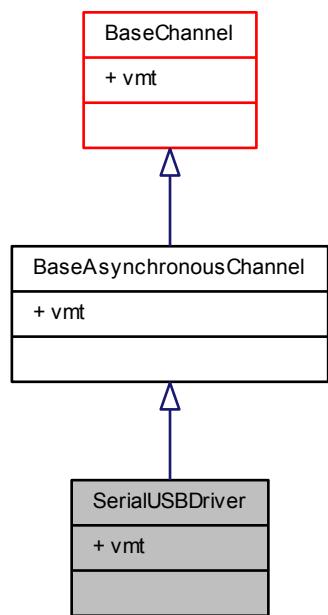
If set to zero then the INT endpoint is assumed to be not present, USB descriptors must be changed accordingly.

## 8.62 SerialUSBDriver Struct Reference

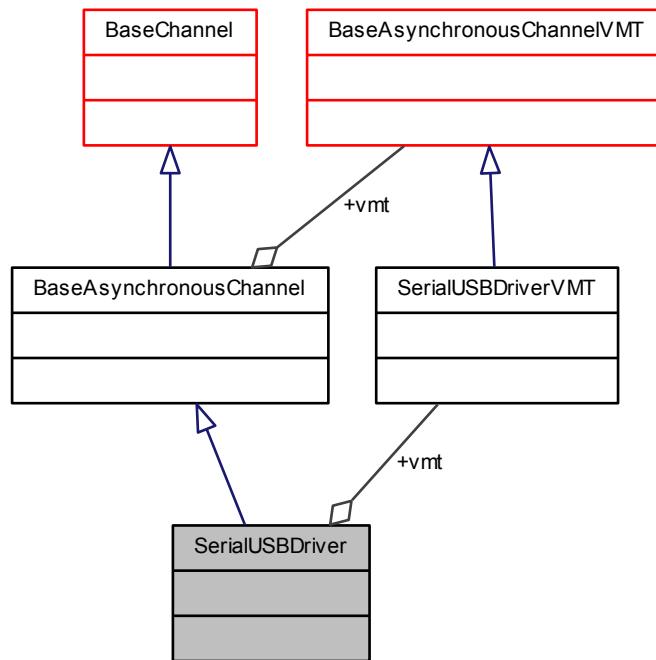
Full duplex serial driver class.

```
#include <serial_usb.h>
```

Inheritance diagram for SerialUSBDriver:



Collaboration diagram for SerialUSBDriver:



## Data Fields

- const struct [SerialUSBDriverVMT](#) \* vmt

*Virtual Methods Table.*

### 8.62.1 Detailed Description

Full duplex serial driver class.

This class extends [BaseAsynchronousChannel](#) by adding physical I/O queues.

### 8.62.2 Field Documentation

#### 8.62.2.1 const struct [SerialUSBDriverVMT](#)\* [SerialUSBDriver::vmt](#)

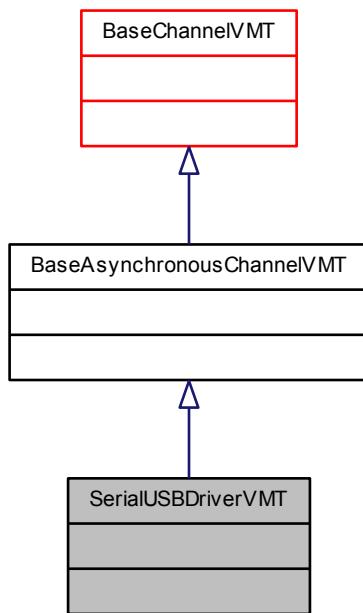
*Virtual Methods Table.*

## 8.63 [SerialUSBDriverVMT](#) Struct Reference

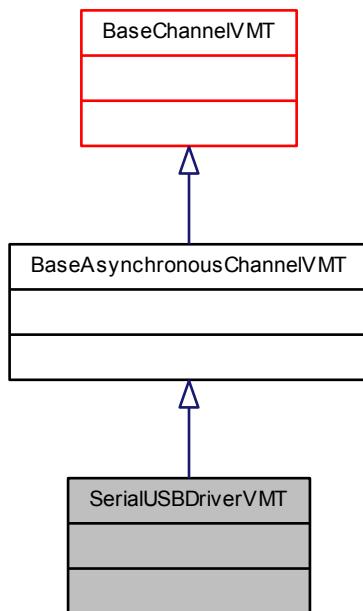
[SerialDriver](#) virtual methods table.

```
#include <serial_usb.h>
```

Inheritance diagram for SerialUSBDriverVMT:



Collaboration diagram for SerialUSBDriverVMT:



### 8.63.1 Detailed Description

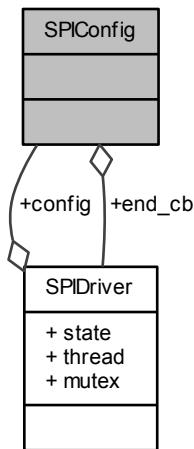
`SerialDriver` virtual methods table.

## 8.64 SPIConfig Struct Reference

Driver configuration structure.

```
#include <spi_lld.h>
```

Collaboration diagram for SPIConfig:



### Data Fields

- `spicallback_t end_cb`

*Operation complete callback or NULL.*

### 8.64.1 Detailed Description

Driver configuration structure.

#### Note

Implementations may extend this structure to contain more, architecture dependent, fields.

### 8.64.2 Field Documentation

#### 8.64.2.1 spicallback\_t SPIConfig::end\_cb

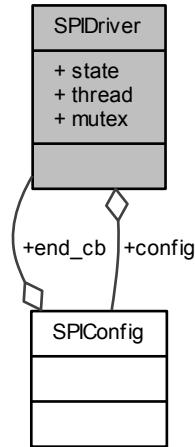
*Operation complete callback or NULL.*

## 8.65 SPIDriver Struct Reference

Structure representing an SPI driver.

```
#include <spi_llld.h>
```

Collaboration diagram for SPIDriver:



### Data Fields

- **spistate\_t state**  
*Driver state.*
- **const SPIConfig \* config**  
*Current configuration data.*
- **thread\_reference\_t thread**  
*Waiting thread.*
- **mutex\_t mutex**  
*Mutex protecting the peripheral.*

### 8.65.1 Detailed Description

Structure representing an SPI driver.

#### Note

Implementations may extend this structure to contain more, architecture dependent, fields.

### 8.65.2 Field Documentation

#### 8.65.2.1 spistate\_t SPIDriver::state

Driver state.

#### 8.65.2.2 const SPIConfig\* SPIDriver::config

Current configuration data.

#### 8.65.2.3 thread\_reference\_t SPIDriver::thread

Waiting thread.

#### 8.65.2.4 mutex\_t SPIDriver::mutex

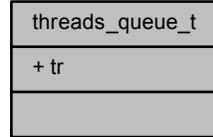
Mutex protecting the peripheral.

## 8.66 threads\_queue\_t Struct Reference

Type of a thread queue.

```
#include <osal.h>
```

Collaboration diagram for threads\_queue\_t:



### 8.66.1 Detailed Description

Type of a thread queue.

A thread queue is a queue of sleeping threads, queued threads can be dequeued one at time or all together.

#### Note

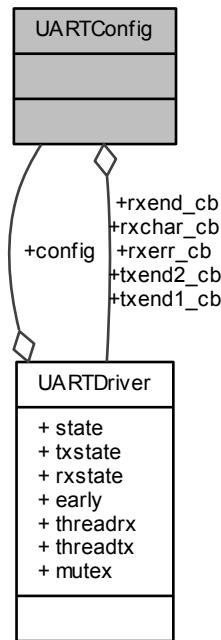
If the OSAL is implemented on a bare metal machine without RTOS then the queue can be implemented as a single thread reference.

## 8.67 UARTConfig Struct Reference

Driver configuration structure.

```
#include <uart_lld.h>
```

Collaboration diagram for UARTConfig:



## Data Fields

- [uartcb\\_t txend1\\_cb](#)  
*End of transmission buffer callback.*
- [uartcb\\_t txend2\\_cb](#)  
*Physical end of transmission callback.*
- [uartcb\\_t rxend\\_cb](#)  
*Receive buffer filled callback.*
- [uartccb\\_t rxchar\\_cb](#)  
*Character received while out if the `UART_RECEIVE` state.*
- [uar tcb\\_t rxerr\\_cb](#)  
*Receive error callback.*

### 8.67.1 Detailed Description

Driver configuration structure.

#### Note

Implementations may extend this structure to contain more, architecture dependent, fields.

### 8.67.2 Field Documentation

#### 8.67.2.1 `uartcb_t` `UARTConfig::txend1_cb`

End of transmission buffer callback.

#### 8.67.2.2 `uartcb_t` `UARTConfig::txend2_cb`

Physical end of transmission callback.

#### 8.67.2.3 `uartcb_t` `UARTConfig::rxend_cb`

Receive buffer filled callback.

#### 8.67.2.4 `uartccb_t` `UARTConfig::rxchar_cb`

Character received while out if the `UART_RECEIVE` state.

#### 8.67.2.5 `uartecb_t` `UARTConfig::rxerr_cb`

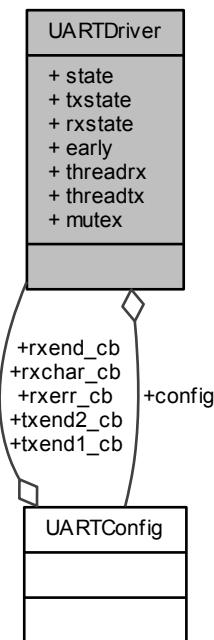
Receive error callback.

## 8.68 `UARTDriver` Struct Reference

Structure representing an UART driver.

```
#include <uart_lld.h>
```

Collaboration diagram for UARTDriver:



## Data Fields

- `uartstate_t state`  
*Driver state.*
- `uartxstate_t txstate`  
*Transmitter state.*
- `uartrxstate_t rxstate`  
*Receiver state.*
- `const UARTConfig * config`  
*Current configuration data.*
- `bool early`  
*Synchronization flag for transmit operations.*
- `thread_reference_t threadrx`  
*Waiting thread on RX.*
- `thread_reference_t threadtx`  
*Waiting thread on TX.*
- `mutex_t mutex`  
*Mutex protecting the peripheral.*

### 8.68.1 Detailed Description

Structure representing an UART driver.

**Note**

Implementations may extend this structure to contain more, architecture dependent, fields.

## 8.68.2 Field Documentation

### 8.68.2.1 `uartstate_t` `UARTDriver::state`

Driver state.

### 8.68.2.2 `uartxstate_t` `UARTDriver::txstate`

Transmitter state.

### 8.68.2.3 `uartrxstate_t` `UARTDriver::rxstate`

Receiver state.

### 8.68.2.4 `const UARTConfig*` `UARTDriver::config`

Current configuration data.

### 8.68.2.5 `bool` `UARTDriver::early`

Synchronization flag for transmit operations.

### 8.68.2.6 `thread_reference_t` `UARTDriver::threadrx`

Waiting thread on RX.

### 8.68.2.7 `thread_reference_t` `UARTDriver::threadtx`

Waiting thread on TX.

### 8.68.2.8 `mutex_t` `UARTDriver::mutex`

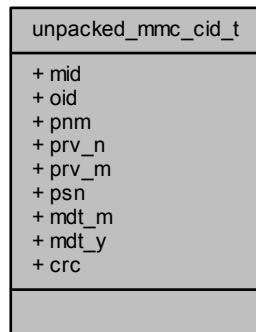
Mutex protecting the peripheral.

## 8.69 `unpacked_mmc_cid_t` Struct Reference

Unpacked CID register from MMC.

```
#include <hal_mmcsd.h>
```

Collaboration diagram for unpacked\_mmc\_cid\_t:



#### 8.69.1 Detailed Description

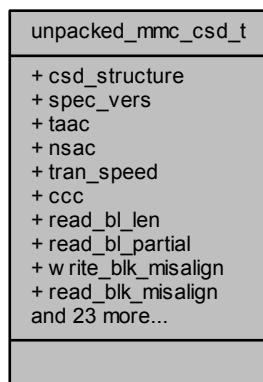
Unpacked CID register from MMC.

### 8.70 unpacked\_mmc\_csd\_t Struct Reference

Unpacked CSD register from MMC.

```
#include <hal_mmc.h>
```

Collaboration diagram for unpacked\_mmc\_csd\_t:



#### 8.70.1 Detailed Description

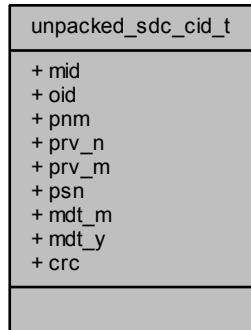
Unpacked CSD register from MMC.

## 8.71 unpacked\_sdc\_cid\_t Struct Reference

Unpacked CID register from SDC.

```
#include <hal_mmc.h>
```

Collaboration diagram for unpacked\_sdc\_cid\_t:



### 8.71.1 Detailed Description

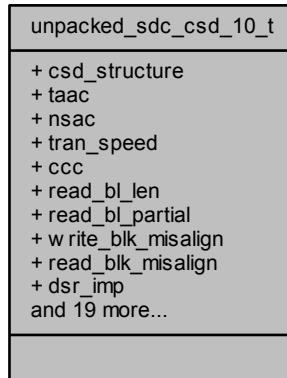
Unpacked CID register from SDC.

## 8.72 unpacked\_sdc\_csd\_10\_t Struct Reference

Unpacked CSD v1.0 register from SDC.

```
#include <hal_mmc.h>
```

Collaboration diagram for unpacked\_sdc\_csd\_10\_t:



### 8.72.1 Detailed Description

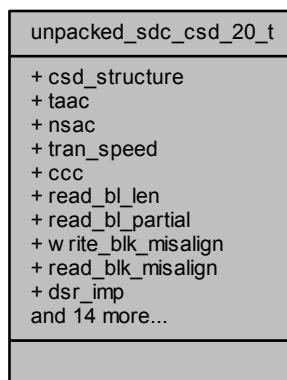
Unpacked CSD v1.0 register from SDC.

## 8.73 unpacked\_sdc\_csd\_20\_t Struct Reference

Unpacked CSD v2.0 register from SDC.

```
#include <hal_mmcsd.h>
```

Collaboration diagram for unpacked\_sdc\_csd\_20\_t:



### 8.73.1 Detailed Description

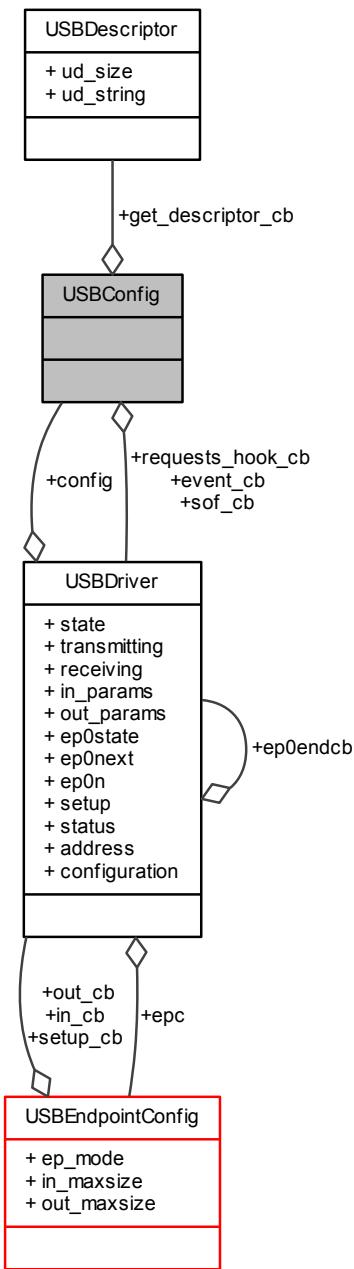
Unpacked CSD v2.0 register from SDC.

## 8.74 USBConfig Struct Reference

Type of an USB driver configuration structure.

```
#include <usb_lld.h>
```

Collaboration diagram for USBConfig:



## Data Fields

- [usbeventcb\\_t event\\_cb](#)  
*USB events callback.*
- [usbgetdescriptor\\_t get\\_descriptor\\_cb](#)  
*Device GET\_DESCRIPTOR request callback.*
- [usbreqhandler\\_t requests\\_hook\\_cb](#)

- Requests hook callback.*
- **usbcallback\_t sof\_cb**  
*Start Of Frame callback.*

### 8.74.1 Detailed Description

Type of an USB driver configuration structure.

### 8.74.2 Field Documentation

#### 8.74.2.1 usbeventcb\_t USBConfig::event\_cb

USB events callback.

This callback is invoked when an USB driver event is registered.

#### 8.74.2.2 usbgetdescriptor\_t USBConfig::get\_descriptor\_cb

Device GET\_DESCRIPTOR request callback.

##### Note

This callback is mandatory and cannot be set to NULL.

#### 8.74.2.3 usbreqhandler\_t USBConfig::requests\_hook\_cb

Requests hook callback.

This hook allows to be notified of standard requests or to handle non standard requests.

#### 8.74.2.4 usbcallback\_t USBConfig::sof\_cb

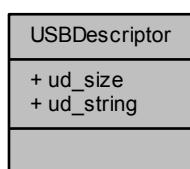
Start Of Frame callback.

## 8.75 USBDescriptor Struct Reference

Type of an USB descriptor.

```
#include <usb.h>
```

Collaboration diagram for USBDescriptor:



## Data Fields

- `size_t ud_size`  
*Descriptor size in unicode characters.*
- `const uint8_t * ud_string`  
*Pointer to the descriptor.*

### 8.75.1 Detailed Description

Type of an USB descriptor.

### 8.75.2 Field Documentation

#### 8.75.2.1 `size_t USBDescriptor::ud_size`

Descriptor size in unicode characters.

#### 8.75.2.2 `const uint8_t* USBDescriptor::ud_string`

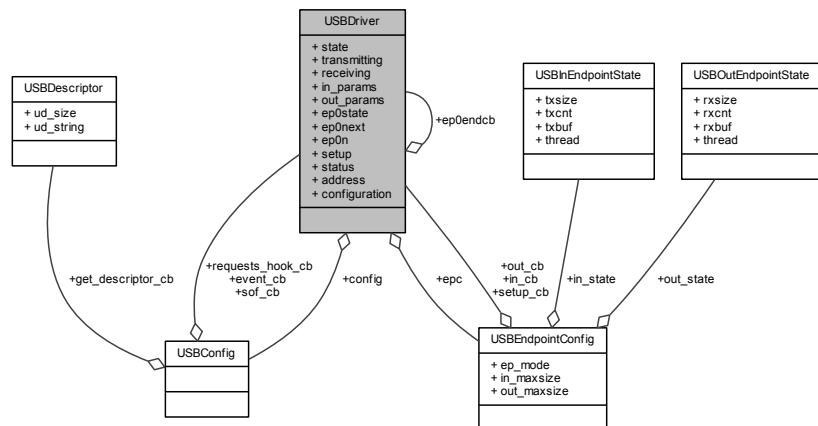
Pointer to the descriptor.

## 8.76 USBDriver Struct Reference

Structure representing an USB driver.

```
#include <usb_ll.h>
```

Collaboration diagram for USBDriver:



## Data Fields

- `usbstate_t state`  
*Driver state.*
- `const USBConfig * config`

- **Current configuration data.**
- **uint16\_t transmitting**  
*Bit map of the transmitting IN endpoints.*
- **uint16\_t receiving**  
*Bit map of the receiving OUT endpoints.*
- **const USBEndpointConfig \* epc [USB\_MAX\_ENDPOINTS+1]**  
*Active endpoints configurations.*
- **void \* in\_params [USB\_MAX\_ENDPOINTS]**  
*Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.*
- **void \* out\_params [USB\_MAX\_ENDPOINTS]**  
*Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.*
- **usbep0state\_t ep0state**  
*Endpoint 0 state.*
- **uint8\_t \* ep0next**  
*Next position in the buffer to be transferred through endpoint 0.*
- **size\_t ep0n**  
*Number of bytes yet to be transferred through endpoint 0.*
- **usbcallback\_t ep0endcb**  
*Endpoint 0 end transaction callback.*
- **uint8\_t setup [8]**  
*Setup packet buffer.*
- **uint16\_t status**  
*Current USB device status.*
- **uint8\_t address**  
*Assigned USB address.*
- **uint8\_t configuration**  
*Current USB device configuration.*

### 8.76.1 Detailed Description

Structure representing an USB driver.

### 8.76.2 Field Documentation

#### 8.76.2.1 **usbstate\_t** `USBDriver::state`

Driver state.

#### 8.76.2.2 **const USBConfig\*** `USBDriver::config`

Current configuration data.

#### 8.76.2.3 **uint16\_t** `USBDriver::transmitting`

Bit map of the transmitting IN endpoints.

#### 8.76.2.4 **uint16\_t** `USBDriver::receiving`

Bit map of the receiving OUT endpoints.

8.76.2.5 `const USBEndpointConfig* USBDriver::epc[USB_MAX_ENDPOINTS+1]`

Active endpoints configurations.

8.76.2.6 `void* USBDriver::in_params[USB_MAX_ENDPOINTS]`

Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.

**Note**

The base index is one, the endpoint zero does not have a reserved element in this array.

8.76.2.7 `void* USBDriver::out_params[USB_MAX_ENDPOINTS]`

Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.

**Note**

The base index is one, the endpoint zero does not have a reserved element in this array.

8.76.2.8 `usbep0state_t USBDriver::ep0state`

Endpoint 0 state.

8.76.2.9 `uint8_t* USBDriver::ep0next`

Next position in the buffer to be transferred through endpoint 0.

8.76.2.10 `size_t USBDriver::ep0n`

Number of bytes yet to be transferred through endpoint 0.

8.76.2.11 `usbcallback_t USBDriver::ep0endcb`

Endpoint 0 end transaction callback.

8.76.2.12 `uint8_t USBDriver::setup[8]`

Setup packet buffer.

8.76.2.13 `uint16_t USBDriver::status`

Current USB device status.

8.76.2.14 `uint8_t USBDriver::address`

Assigned USB address.

8.76.2.15 `uint8_t USBDriver::configuration`

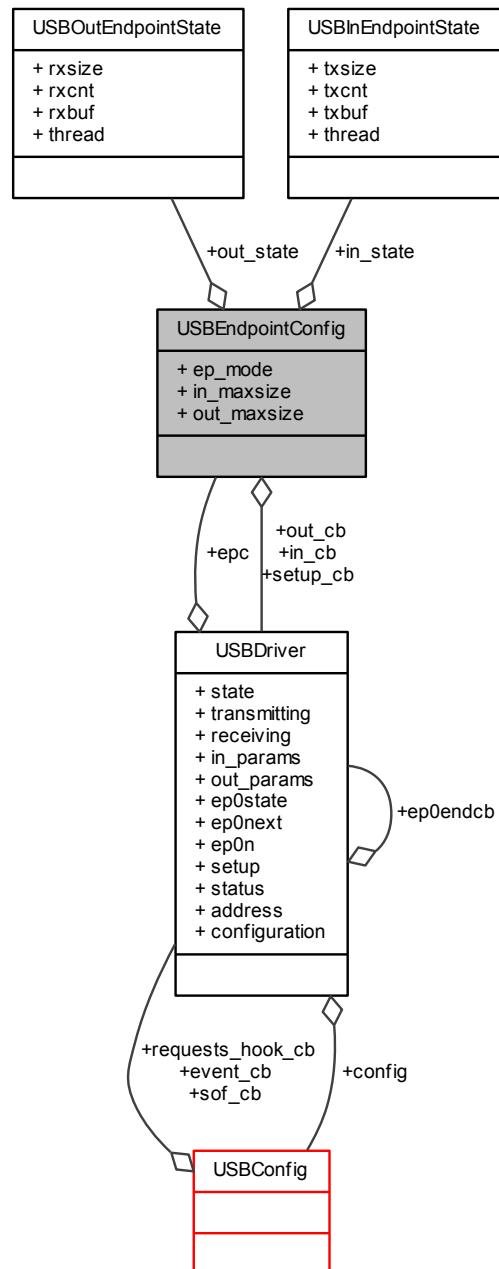
Current USB device configuration.

## 8.77 USBEndpointConfig Struct Reference

Type of an USB endpoint configuration structure.

```
#include <usb_lld.h>
```

Collaboration diagram for USBEndpointConfig:



## Data Fields

- `uint32_t ep_mode`  
*Type and mode of the endpoint.*
- `usbepcallback_t setup_cb`  
*Setup packet notification callback.*
- `usbepcallback_t in_cb`  
*IN endpoint notification callback.*
- `usbepcallback_t out_cb`  
*OUT endpoint notification callback.*
- `uint16_t in_maxsize`  
*IN endpoint maximum packet size.*
- `uint16_t out_maxsize`  
*OUT endpoint maximum packet size.*
- `USBInEndpointState * in_state`  
*USBInEndpointState associated to the IN endpoint.*
- `USBOutEndpointState * out_state`  
*USBOutEndpointState associated to the OUT endpoint.*

### 8.77.1 Detailed Description

Type of an USB endpoint configuration structure.

#### Note

Platform specific restrictions may apply to endpoints.

### 8.77.2 Field Documentation

#### 8.77.2.1 `uint32_t USBEndpointConfig::ep_mode`

Type and mode of the endpoint.

#### 8.77.2.2 `usbepcallback_t USBEndpointConfig::setup_cb`

Setup packet notification callback.

This callback is invoked when a setup packet has been received.

#### Postcondition

The application must immediately call `usbReadPacket ()` in order to access the received packet.

#### Note

This field is only valid for `USB_EP_MODE_TYPE_CTRL` endpoints, it should be set to `NULL` for other endpoint types.

#### 8.77.2.3 `usbepcallback_t USBEndpointConfig::in_cb`

IN endpoint notification callback.

This field must be set to `NULL` if the IN endpoint is not used.

#### 8.77.2.4 `usbepcallback_t USBEndpointConfig::out_cb`

OUT endpoint notification callback.

This field must be set to `NULL` if the OUT endpoint is not used.

#### 8.77.2.5 `uint16_t USBEndpointConfig::in_maxsize`

IN endpoint maximum packet size.

This field must be set to zero if the IN endpoint is not used.

#### 8.77.2.6 `uint16_t USBEndpointConfig::out_maxsize`

OUT endpoint maximum packet size.

This field must be set to zero if the OUT endpoint is not used.

#### 8.77.2.7 `USBInEndpointState* USBEndpointConfig::in_state`

`USBInEndpointState` associated to the IN endpoint.

This structure maintains the state of the IN endpoint.

#### 8.77.2.8 `USBOutEndpointState* USBEndpointConfig::out_state`

`USBOutEndpointState` associated to the OUT endpoint.

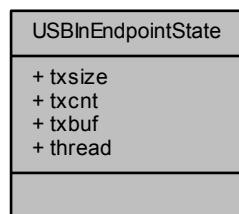
This structure maintains the state of the OUT endpoint.

## 8.78 `USBInEndpointState` Struct Reference

Type of an IN endpoint state structure.

```
#include <usb_lld.h>
```

Collaboration diagram for `USBInEndpointState`:



### Data Fields

- `size_t txsize`

- `size_t txcnt`  
*Requested transmit transfer size.*
- `const uint8_t * txbuf`  
*Transmitted bytes so far.*
- `thread_reference_t thread`  
*Pointer to the transmission linear buffer.*
- `thread_reference_t thread`  
*Waiting thread.*

### 8.78.1 Detailed Description

Type of an IN endpoint state structure.

### 8.78.2 Field Documentation

#### 8.78.2.1 `size_t USBInEndpointState::txsize`

Requested transmit transfer size.

#### 8.78.2.2 `size_t USBInEndpointState::txcnt`

Transmitted bytes so far.

#### 8.78.2.3 `const uint8_t* USBInEndpointState::txbuf`

Pointer to the transmission linear buffer.

#### 8.78.2.4 `thread_reference_t USBInEndpointState::thread`

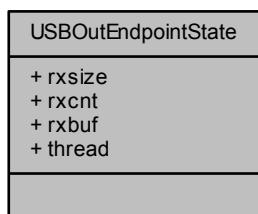
Waiting thread.

## 8.79 USBOutEndpointState Struct Reference

Type of an OUT endpoint state structure.

```
#include <usb_lld.h>
```

Collaboration diagram for USBOutEndpointState:



## Data Fields

- `size_t rxsize`  
*Requested receive transfer size.*
- `size_t rxcnt`  
*Received bytes so far.*
- `uint8_t * rdbuf`  
*Pointer to the receive linear buffer.*
- `thread_reference_t thread`  
*Waiting thread.*

### 8.79.1 Detailed Description

Type of an OUT endpoint state structure.

### 8.79.2 Field Documentation

#### 8.79.2.1 `size_t USBOutEndpointState::rxsize`

Requested receive transfer size.

#### 8.79.2.2 `size_t USBOutEndpointState::rxcnt`

Received bytes so far.

#### 8.79.2.3 `uint8_t* USBOutEndpointState::rdbuf`

Pointer to the receive linear buffer.

#### 8.79.2.4 `thread_reference_t USBOutEndpointState::thread`

Waiting thread.

## 8.80 WDGConfig Struct Reference

Driver configuration structure.

```
#include <wdg_lld.h>
```

Collaboration diagram for WDGConfig:



### 8.80.1 Detailed Description

Driver configuration structure.

#### Note

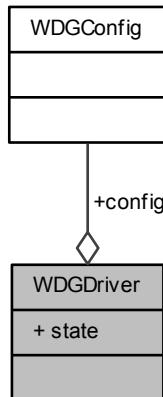
It could be empty on some architectures.

## 8.81 WDGDriver Struct Reference

Structure representing an WDG driver.

```
#include <wdg_lld.h>
```

Collaboration diagram for WDGDriver:



### Data Fields

- `wdgstate_t state`  
*Driver state.*
- `const WDGConfig * config`  
*Current configuration data.*

### 8.81.1 Detailed Description

Structure representing an WDG driver.

### 8.81.2 Field Documentation

#### 8.81.2.1 `wdgstate_t WDGDriver::state`

Driver state.

**8.81.2.2 const WDGConfig\* WDGDriver::config**

Current configuration data.

# Chapter 9

## File Documentation

### 9.1 adc.c File Reference

ADC Driver code.

```
#include "hal.h"
```

#### Functions

- void `adcInit` (void)  
*ADC Driver initialization.*
- void `adcObjectInit` (`ADCDriver` \*adcp)  
*Initializes the standard part of a `ADCDriver` structure.*
- void `adcStart` (`ADCDriver` \*adcp, const `ADCCConfig` \*config)  
*Configures and activates the ADC peripheral.*
- void `adcStop` (`ADCDriver` \*adcp)  
*Deactivates the ADC peripheral.*
- void `adcStartConversion` (`ADCDriver` \*adcp, const `ADCCConversionGroup` \*grpp, `adcsample_t` \*samples, `size_t` depth)  
*Starts an ADC conversion.*
- void `adcStartConversionI` (`ADCDriver` \*adcp, const `ADCCConversionGroup` \*grpp, `adcsample_t` \*samples, `size_t` depth)  
*Starts an ADC conversion.*
- void `adcStopConversion` (`ADCDriver` \*adcp)  
*Stops an ongoing conversion.*
- void `adcStopConversionI` (`ADCDriver` \*adcp)  
*Stops an ongoing conversion.*
- `msg_t adcConvert` (`ADCDriver` \*adcp, const `ADCCConversionGroup` \*grpp, `adcsample_t` \*samples, `size_t` depth)  
*Performs an ADC conversion.*
- void `adcAcquireBus` (`ADCDriver` \*adcp)  
*Gains exclusive access to the ADC peripheral.*
- void `adcReleaseBus` (`ADCDriver` \*adcp)  
*Releases exclusive access to the ADC peripheral.*

#### 9.1.1 Detailed Description

ADC Driver code.

## 9.2 adc.h File Reference

ADC Driver macros and structures.

```
#include "adc_lld.h"
```

### Macros

#### ADC configuration options

- #define `ADC_USE_WAIT` TRUE  
*Enables synchronous APIs.*
- #define `ADC_USE_MUTUAL_EXCLUSION` TRUE  
*Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.*

#### Low level driver helper macros

- #define `_adc_reset_i`(adcp) `osalThreadResumel(&(adcp)->thread, MSG_RESET)`  
*Resumes a thread waiting for a conversion completion.*
- #define `_adc_reset_s`(adcp) `osalThreadResumeS(&(adcp)->thread, MSG_RESET)`  
*Resumes a thread waiting for a conversion completion.*
- #define `_adc_wakeup_isr`(adcp)  
*Wakes up the waiting thread.*
- #define `_adc_timeout_isr`(adcp)  
*Wakes up the waiting thread with a timeout message.*
- #define `_adc_isr_half_code`(adcp)  
*Common ISR code, half buffer event.*
- #define `_adc_isr_full_code`(adcp)  
*Common ISR code, full buffer event.*
- #define `_adc_isr_error_code`(adcp, err)  
*Common ISR code, error event.*

### Enumerations

- enum `adcstate_t` {  
  `ADC_UNINIT` = 0, `ADC_STOP` = 1, `ADC_READY` = 2, `ADC_ACTIVE` = 3,  
  `ADC_COMPLETE` = 4, `ADC_ERROR` = 5 }  
*Driver state machine possible states.*

### Functions

- void `adclInit` (void)  
*ADC Driver initialization.*
- void `adcObjectInit` (`ADCDriver` \*adcp)  
*Initializes the standard part of a `ADCDriver` structure.*
- void `adcStart` (`ADCDriver` \*adcp, const `ADCCConfig` \*config)  
*Configures and activates the ADC peripheral.*
- void `adcStop` (`ADCDriver` \*adcp)  
*Deactivates the ADC peripheral.*
- void `adcStartConversion` (`ADCDriver` \*adcp, const `ADCCConversionGroup` \*grpp, `adcsample_t` \*samples, `size_t` depth)  
*Starts an ADC conversion.*
- void `adcStartConversionI` (`ADCDriver` \*adcp, const `ADCCConversionGroup` \*grpp, `adcsample_t` \*samples, `size_t` depth)

- void `adcStopConversion` (ADCDriver \*adcp)
 

*Starts an ADC conversion.*
- void `adcStopConversion` (ADCDriver \*adcp)
 

*Stops an ongoing conversion.*
- void `adcStopConversionI` (ADCDriver \*adcp)
 

*Stops an ongoing conversion.*
- msg\_t `adcConvert` (ADCDriver \*adcp, const ADCConversionGroup \*grpp, adcsample\_t \*samples, size\_t depth)
 

*Performs an ADC conversion.*
- void `adcAcquireBus` (ADCDriver \*adcp)
 

*Gains exclusive access to the ADC peripheral.*
- void `adcReleaseBus` (ADCDriver \*adcp)
 

*Releases exclusive access to the ADC peripheral.*

### 9.2.1 Detailed Description

ADC Driver macros and structures.

## 9.3 adc\_lld.c File Reference

PLATFORM ADC subsystem low level driver source.

```
#include "hal.h"
```

### Functions

- void `adc_lld_init` (void)
 

*Low level ADC driver initialization.*
- void `adc_lld_start` (ADCDriver \*adcp)
 

*Configures and activates the ADC peripheral.*
- void `adc_lld_stop` (ADCDriver \*adcp)
 

*Deactivates the ADC peripheral.*
- void `adc_lld_start_conversion` (ADCDriver \*adcp)
 

*Starts an ADC conversion.*
- void `adc_lld_stop_conversion` (ADCDriver \*adcp)
 

*Stops an ongoing conversion.*

### Variables

- ADCDriver ADCD1
 

*ADC1 driver identifier.*

### 9.3.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

## 9.4 adc\_lld.h File Reference

PLATFORM ADC subsystem low level driver header.

## Data Structures

- struct [ADCConversionGroup](#)  
*Conversion group configuration structure.*
- struct [ADCCConfig](#)  
*Driver configuration structure.*
- struct [ADCDriver](#)  
*Structure representing an ADC driver.*

## Macros

### PLATFORM configuration options

- #define [PLATFORM\\_ADC\\_USE\\_ADC1](#) FALSE  
*ADC1 driver enable switch.*

## Typedefs

- typedef uint16\_t [adcsample\\_t](#)  
*ADC sample data type.*
- typedef uint16\_t [adc\\_channels\\_num\\_t](#)  
*Channels number in a conversion group.*
- typedef struct [ADCDriver](#) [ADCDriver](#)  
*Type of a structure representing an ADC driver.*
- typedef void(\* [adccallback\\_t](#)) ([ADCDriver](#) \*adcp, [adcsample\\_t](#) \*buffer, size\_t n)  
*ADC notification callback type.*
- typedef void(\* [adcerrorcallback\\_t](#)) ([ADCDriver](#) \*adcp, [adcerror\\_t](#) err)  
*ADC error callback type.*

## Enumerations

- enum [adcerror\\_t](#) { [ADC\\_ERR\\_DMAFAILURE](#) = 0, [ADC\\_ERR\\_OVERFLOW](#) = 1, [ADC\\_ERR\\_AWD](#) = 2 }  
*Possible ADC failure causes.*

## Functions

- void [adc\\_lld\\_init](#) (void)  
*Low level ADC driver initialization.*
- void [adc\\_lld\\_start](#) ([ADCDriver](#) \*adcp)  
*Configures and activates the ADC peripheral.*
- void [adc\\_lld\\_stop](#) ([ADCDriver](#) \*adcp)  
*Deactivates the ADC peripheral.*
- void [adc\\_lld\\_start\\_conversion](#) ([ADCDriver](#) \*adcp)  
*Starts an ADC conversion.*
- void [adc\\_lld\\_stop\\_conversion](#) ([ADCDriver](#) \*adcp)  
*Stops an ongoing conversion.*

### 9.4.1 Detailed Description

PLATFORM ADC subsystem low level driver header.

## 9.5 can.c File Reference

CAN Driver code.

```
#include "hal.h"
```

### Functions

- void **canInit** (void)  
*CAN Driver initialization.*
- void **canObjectInit** (**CANDriver** \*canp)  
*Initializes the standard part of a **CANDriver** structure.*
- void **canStart** (**CANDriver** \*canp, const **CANConfig** \*config)  
*Configures and activates the CAN peripheral.*
- void **canStop** (**CANDriver** \*canp)  
*Deactivates the CAN peripheral.*
- bool **canTryTransmitl** (**CANDriver** \*canp, **canmbx\_t** mailbox, const **CANTxFrame** \*ctfp)  
*Can frame transmission attempt.*
- bool **canTryReceivev** (**CANDriver** \*canp, **canmbx\_t** mailbox, **CANRxFrame** \*crfp)  
*Can frame receive attempt.*
- **msg\_t** **canTransmit** (**CANDriver** \*canp, **canmbx\_t** mailbox, const **CANTxFrame** \*ctfp, **systime\_t** timeout)  
*Can frame transmission.*
- **msg\_t** **canReceive** (**CANDriver** \*canp, **canmbx\_t** mailbox, **CANRxFrame** \*crfp, **systime\_t** timeout)  
*Can frame receive.*
- void **canSleep** (**CANDriver** \*canp)  
*Enters the sleep mode.*
- void **canWakeup** (**CANDriver** \*canp)  
*Forces leaving the sleep mode.*

### 9.5.1 Detailed Description

CAN Driver code.

## 9.6 can.h File Reference

CAN Driver macros and structures.

```
#include "can_lld.h"
```

### Macros

- #define **CAN\_ANY\_MAILBOX** 0  
*Special mailbox identifier.*

### CAN status flags

- #define **CAN\_LIMIT\_WARNING** 1U  
*Errors rate warning.*
- #define **CAN\_LIMIT\_ERROR** 2U

- `#define CAN_BUS_OFF_ERROR 4U`  
*Bus off condition reached.*
- `#define CAN_FRAMING_ERROR 8U`  
*Framing error of some kind on the CAN bus.*
- `#define CAN_OVERFLOW_ERROR 16U`  
*Overflow in receive queue.*

## CAN configuration options

- `#define CAN_USE_SLEEP_MODE TRUE`  
*Sleep mode related APIs inclusion switch.*

## Macro Functions

- `#define CAN_MAILBOX_TO_MASK(mbx) (1U << ((mbx) - 1U))`  
*Converts a mailbox index to a bit mask.*

## Enumerations

- enum `canstate_t` {
   
  `CAN_UNINIT` = 0, `CAN_STOP` = 1, `CAN_STARTING` = 2, `CAN_READY` = 3,
   
  `CAN_SLEEP` = 4
 }
   
*Driver state machine possible states.*

## Functions

- void `canInit` (void)  
*CAN Driver initialization.*
- void `canObjectInit` (`CANDriver` \*`canp`)  
*Initializes the standard part of a `CANDriver` structure.*
- void `canStart` (`CANDriver` \*`canp`, const `CANConfig` \*`config`)  
*Configures and activates the CAN peripheral.*
- void `canStop` (`CANDriver` \*`canp`)  
*Deactivates the CAN peripheral.*
- bool `canTryTransmit` (`CANDriver` \*`canp`, `canmbx_t` `mailbox`, const `CANTxFrame` \*`ctfp`)  
*Can frame transmission attempt.*
- bool `canTryReceive` (`CANDriver` \*`canp`, `canmbx_t` `mailbox`, `CANRxFrame` \*`crfp`)  
*Can frame receive attempt.*
- `msg_t` `canTransmit` (`CANDriver` \*`canp`, `canmbx_t` `mailbox`, const `CANTxFrame` \*`ctfp`, `systime_t` `timeout`)  
*Can frame transmission.*
- `msg_t` `canReceive` (`CANDriver` \*`canp`, `canmbx_t` `mailbox`, `CANRxFrame` \*`crfp`, `systime_t` `timeout`)  
*Can frame receive.*

### 9.6.1 Detailed Description

CAN Driver macros and structures.

## 9.7 can\_lld.c File Reference

PLATFORM CAN subsystem low level driver source.

```
#include "hal.h"
```

## Functions

- void `can_lld_init` (void)  
*Low level CAN driver initialization.*
- void `can_lld_start` (`CANDriver` \*canp)  
*Configures and activates the CAN peripheral.*
- void `can_lld_stop` (`CANDriver` \*canp)  
*Deactivates the CAN peripheral.*
- bool `can_lld_is_tx_empty` (`CANDriver` \*canp, `canmbx_t` mailbox)  
*Determines whether a frame can be transmitted.*
- void `can_lld_transmit` (`CANDriver` \*canp, `canmbx_t` mailbox, const `CANTxFrame` \*ctfp)  
*Inserts a frame into the transmit queue.*
- bool `can_lld_is_rx_nonempty` (`CANDriver` \*canp, `canmbx_t` mailbox)  
*Determines whether a frame has been received.*
- void `can_lld_receive` (`CANDriver` \*canp, `canmbx_t` mailbox, `CANRxFrame` \*crfp)  
*Receives a frame from the input queue.*
- void `can_lld_sleep` (`CANDriver` \*canp)  
*Enters the sleep mode.*
- void `can_lld_wakeup` (`CANDriver` \*canp)  
*Enforces leaving the sleep mode.*

## Variables

- `CANDriver CAND1`  
*CAN1 driver identifier.*

### 9.7.1 Detailed Description

PLATFORM CAN subsystem low level driver source.

## 9.8 can\_lld.h File Reference

PLATFORM CAN subsystem low level driver header.

## Data Structures

- struct `CANTxFrame`  
*CAN transmission frame.*
- struct `CANRxFrame`  
*CAN received frame.*
- struct `CANConfig`  
*Driver configuration structure.*
- struct `CANDriver`  
*Structure representing an CAN driver.*

## Macros

- `#define CAN_TX_MAILBOXES 1`  
*Number of transmit mailboxes.*
- `#define CAN_RX_MAILBOXES 1`  
*Number of receive mailboxes.*

## PLATFORM configuration options

- `#define PLATFORM_CAN_USE_CAN1 FALSE`  
*CAN1 driver enable switch.*

## Typedefs

- `typedef uint32_t canmbx_t`  
*Type of a transmission mailbox index.*

## Functions

- `void can_lld_init (void)`  
*Low level CAN driver initialization.*
- `void can_lld_start (CANDriver *canp)`  
*Configures and activates the CAN peripheral.*
- `void can_lld_stop (CANDriver *canp)`  
*Deactivates the CAN peripheral.*
- `bool can_lld_is_tx_empty (CANDriver *canp, canmbx_t mailbox)`  
*Determines whether a frame can be transmitted.*
- `void can_lld_transmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)`  
*Inserts a frame into the transmit queue.*
- `bool can_lld_is_rx_nonempty (CANDriver *canp, canmbx_t mailbox)`  
*Determines whether a frame has been received.*
- `void can_lld_receive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)`  
*Receives a frame from the input queue.*
- `void can_lld_sleep (CANDriver *canp)`  
*Enters the sleep mode.*
- `void can_lld_wakeup (CANDriver *canp)`  
*Enforces leaving the sleep mode.*

### 9.8.1 Detailed Description

PLATFORM CAN subsystem low level driver header.

## 9.9 dac.c File Reference

DAC Driver code.

```
#include "hal.h"
```

## Functions

- void `dacInit` (void)  
*DAC Driver initialization.*
- void `dacObjectInit` (`DACDriver` \*`dacp`)  
*Initializes the standard part of a `DACDriver` structure.*
- void `dacStart` (`DACDriver` \*`dacp`, const `DACConfig` \*`config`)  
*Configures and activates the DAC peripheral.*
- void `dacStop` (`DACDriver` \*`dacp`)  
*Deactivates the DAC peripheral.*
- void `dacPutChannelX` (`DACDriver` \*`dacp`, `dacchannel_t` `channel`, `dacsample_t` `sample`)  
*Outputs a value directly on a DAC channel.*
- void `dacStartConversion` (`DACDriver` \*`dacp`, const `DACConversionGroup` \*`grpp`, const `dacsample_t` \*`samples`, `size_t` `depth`)  
*Starts a DAC conversion.*
- void `dacStartConversionI` (`DACDriver` \*`dacp`, const `DACConversionGroup` \*`grpp`, const `dacsample_t` \*`samples`, `size_t` `depth`)  
*Starts a DAC conversion.*
- void `dacStopConversion` (`DACDriver` \*`dacp`)  
*Stops an ongoing conversion.*
- void `dacStopConversionI` (`DACDriver` \*`dacp`)  
*Stops an ongoing conversion.*
- msg\_t `dacConvert` (`DACDriver` \*`dacp`, const `DACConversionGroup` \*`grpp`, const `dacsample_t` \*`samples`, `size_t` `depth`)  
*Performs a DAC conversion.*
- void `dacAcquireBus` (`DACDriver` \*`dacp`)  
*Gains exclusive access to the DAC bus.*
- void `dacReleaseBus` (`DACDriver` \*`dacp`)  
*Releases exclusive access to the DAC bus.*

### 9.9.1 Detailed Description

DAC Driver code.

## 9.10 dac.h File Reference

DAC Driver macros and structures.

```
#include "dac_lld.h"
```

## Macros

### DAC configuration options

- #define `DAC_USE_WAIT` TRUE  
*Enables synchronous APIs.*
- #define `DAC_USE_MUTUAL_EXCLUSION` TRUE  
*Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.*

### Low level driver helper macros

- `#define _dac_wait_s(dacp) osalThreadSuspendS(&(dacp)->thread)`  
*Waits for operation completion.*
- `#define _dac_reset_i(dacp) osalThreadResumeI(&(dacp)->thread, MSG_RESET)`  
*Resumes a thread waiting for a conversion completion.*
- `#define _dac_reset_s(dacp) osalThreadResumeS(&(dacp)->thread, MSG_RESET)`  
*Resumes a thread waiting for a conversion completion.*
- `#define _dac_wakeup_isr(dacp)`  
*Wakes up the waiting thread.*
- `#define _dac_timeout_isr(dacp)`  
*Wakes up the waiting thread with a timeout message.*
- `#define _dac_isr_half_code(dacp)`  
*Common ISR code, half buffer event.*
- `#define _dac_isr_full_code(dacp)`  
*Common ISR code, full buffer event.*
- `#define _dac_isr_error_code(dacp, err)`  
*Common ISR code, error event.*

## Enumerations

- enum `dacstate_t` {  
`DAC_UNINIT` = 0, `DAC_STOP` = 1, `DAC_READY` = 2, `DAC_ACTIVE` = 3,  
`DAC_COMPLETE` = 4, `DAC_ERROR` = 5 }

*Driver state machine possible states.*

## Functions

- `void dacInit (void)`  
*DAC Driver initialization.*
- `void dacObjectInit (DACDriver *dacp)`  
*Initializes the standard part of a `DACDriver` structure.*
- `void dacStart (DACDriver *dacp, const DACConfig *config)`  
*Configures and activates the DAC peripheral.*
- `void dacStop (DACDriver *dacp)`  
*Deactivates the DAC peripheral.*
- `void dacPutChannelX (DACDriver *dacp, dacchannel_t channel, dacsample_t sample)`  
*Outputs a value directly on a DAC channel.*
- `void dacStartConversion (DACDriver *dacp, const DACConversionGroup *grpp, const dacsample_t *samples, size_t depth)`  
*Starts a DAC conversion.*
- `void dacStartConversionI (DACDriver *dacp, const DACConversionGroup *grpp, const dacsample_t *samples, size_t depth)`  
*Starts a DAC conversion.*
- `void dacStopConversion (DACDriver *dacp)`  
*Stops an ongoing conversion.*
- `void dacStopConversionI (DACDriver *dacp)`  
*Stops an ongoing conversion.*

### 9.10.1 Detailed Description

DAC Driver macros and structures.

## 9.11 dac\_lld.c File Reference

PLATFORM DAC subsystem low level driver source.

```
#include "hal.h"
```

### Functions

- `void dac_lld_init (void)`  
*Low level DAC driver initialization.*
- `void dac_lld_start (DACDriver *dacp)`  
*Configures and activates the DAC peripheral.*
- `void dac_lld_stop (DACDriver *dacp)`  
*Deactivates the DAC peripheral.*
- `void dac_lld_put_channel (DACDriver *dacp, dacchannel_t channel, dacsample_t sample)`  
*Outputs a value directly on a DAC channel.*
- `void dac_lld_start_conversion (DACDriver *dacp)`  
*Starts a DAC conversion.*
- `void dac_lld_stop_conversion (DACDriver *dacp)`  
*Stops an ongoing conversion.*

### Variables

- `DACDriver DACD1`  
*DAC1 driver identifier.*

#### 9.11.1 Detailed Description

PLATFORM DAC subsystem low level driver source.

## 9.12 dac\_lld.h File Reference

PLATFORM DAC subsystem low level driver header.

### Data Structures

- `struct DACConversionGroup`  
*DAC Conversion group structure.*
- `struct DACConfig`  
*Driver configuration structure.*
- `struct DACDriver`  
*Structure representing a DAC driver.*

## Macros

- `#define DAC_MAX_CHANNELS 2`  
*Maximum number of DAC channels per unit.*

## Configuration options

- `#define PLATFORM_DAC_USE_DAC1 FALSE`  
*DAC1 CH1 driver enable switch.*

## Typedefs

- `typedef uint32_t dacchannel_t`  
*Type of a DAC channel index.*
- `typedef struct DACDriver DACDriver`  
*Type of a structure representing an DAC driver.*
- `typedef uint16_t dacsample_t`  
*Type representing a DAC sample.*
- `typedef void(* daccallback_t) (DACDriver *dACP, const dacsample_t *buffer, size_t n)`  
*DAC notification callback type.*
- `typedef void(* dacerrorcallback_t) (DACDriver *dACP, dacerror_t err)`  
*ADC error callback type.*

## Enumerations

- `enum dacerror_t { DAC_ERR_DMAFAILURE = 0, DAC_ERR_UNDERFLOW = 1 }`  
*Possible DAC failure causes.*

## Functions

- `void dac_llD_init (void)`  
*Low level DAC driver initialization.*
- `void dac_llD_start (DACDriver *dACP)`  
*Configures and activates the DAC peripheral.*
- `void dac_llD_stop (DACDriver *dACP)`  
*Deactivates the DAC peripheral.*
- `void dac_llD_put_channel (DACDriver *dACP, dacchannel_t channel, dacsample_t sample)`  
*Outputs a value directly on a DAC channel.*
- `void dac_llD_start_conversion (DACDriver *dACP)`  
*Starts a DAC conversion.*
- `void dac_llD_stop_conversion (DACDriver *dACP)`  
*Stops an ongoing conversion.*

### 9.12.1 Detailed Description

PLATFORM DAC subsystem low level driver header.

## 9.13 ext.c File Reference

EXT Driver code.

```
#include "hal.h"
```

### Functions

- void `extInit` (void)  
*EXT Driver initialization.*
- void `extObjectInit` (EXTDriver \*extp)  
*Initializes the standard part of a `EXTDriver` structure.*
- void `extStart` (EXTDriver \*extp, const EXTConfig \*config)  
*Configures and activates the EXT peripheral.*
- void `extStop` (EXTDriver \*extp)  
*Deactivates the EXT peripheral.*
- void `extChannelEnable` (EXTDriver \*extp, expchannel\_t channel)  
*Enables an EXT channel.*
- void `extChannelDisable` (EXTDriver \*extp, expchannel\_t channel)  
*Disables an EXT channel.*
- void `extSetChannelModel` (EXTDriver \*extp, expchannel\_t channel, const EXTChannelConfig \*extcp)  
*Changes the operation mode of a channel.*

### 9.13.1 Detailed Description

EXT Driver code.

## 9.14 ext.h File Reference

EXT Driver macros and structures.

```
#include "ext_lld.h"
```

### Macros

#### EXT channel modes

- #define `EXT_CH_MODE_EDGES_MASK` 3U  
*Mask of edges field.*
- #define `EXT_CH_MODE_DISABLED` 0U  
*Channel disabled.*
- #define `EXT_CH_MODE_RISING_EDGE` 1U  
*Rising edge callback.*
- #define `EXT_CH_MODE_FALLING_EDGE` 2U  
*Falling edge callback.*
- #define `EXT_CH_MODE_BOTH_EDGES` 3U  
*Both edges callback.*
- #define `EXT_CH_MODE_AUTOSTART` 4U  
*Channel started automatically on driver start.*

#### Macro Functions

- #define `extChannelEnable`(extp, channel) `ext_lld_channel_enable(extp, channel)`  
*Enables an EXT channel.*
- #define `extChannelDisable`(extp, channel) `ext_lld_channel_disable(extp, channel)`  
*Disables an EXT channel.*
- #define `extSetChannelMode`(extp, channel, extcp)  
*Changes the operation mode of a channel.*

## Typedefs

- `typedef struct EXTDriver EXTDriver`  
*Type of a structure representing a EXT driver.*

## Enumerations

- enum `extstate_t` { `EXT_UNINIT` = 0, `EXT_STOP` = 1, `EXT_ACTIVE` = 2 }
- Driver state machine possible states.*

## Functions

- `void extInit (void)`  
*EXT Driver initialization.*
- `void extObjectInit (EXTDriver *extp)`  
*Initializes the standard part of a `EXTDriver` structure.*
- `void extStart (EXTDriver *extp, const EXTConfig *config)`  
*Configures and activates the EXT peripheral.*
- `void extStop (EXTDriver *extp)`  
*Deactivates the EXT peripheral.*
- `void extChannelEnable (EXTDriver *extp, expchannel_t channel)`  
*Enables an EXT channel.*
- `void extChannelDisable (EXTDriver *extp, expchannel_t channel)`  
*Disables an EXT channel.*
- `void extSetChannelModel (EXTDriver *extp, expchannel_t channel, const EXTChannelConfig *extcp)`  
*Changes the operation mode of a channel.*

### 9.14.1 Detailed Description

EXT Driver macros and structures.

## 9.15 ext\_lld.c File Reference

PLATFORM EXT subsystem low level driver source.

```
#include "hal.h"
```

## Functions

- void `ext_ll_init` (void)  
*Low level EXT driver initialization.*
- void `ext_ll_start` (EXTDriver \*extp)  
*Configures and activates the EXT peripheral.*
- void `ext_ll_stop` (EXTDriver \*extp)  
*Deactivates the EXT peripheral.*
- void `ext_ll_channel_enable` (EXTDriver \*extp, expchannel\_t channel)  
*Enables an EXT channel.*
- void `ext_ll_channel_disable` (EXTDriver \*extp, expchannel\_t channel)  
*Disables an EXT channel.*

## Variables

- EXTDriver EXTD1  
*EXT1 driver identifier.*

### 9.15.1 Detailed Description

PLATFORM EXT subsystem low level driver source.

## 9.16 ext\_ll.h File Reference

PLATFORM EXT subsystem low level driver header.

### Data Structures

- struct `EXTChannelConfig`  
*Channel configuration structure.*
- struct `EXTConfig`  
*Driver configuration structure.*
- struct `EXTDriver`  
*Structure representing an EXT driver.*

### Macros

- #define `EXT_MAX_CHANNELS` 20  
*Available number of EXT channels.*

### PLATFORM configuration options

- #define `PLATFORM_EXT_USE_EXT1` FALSE  
*EXT driver enable switch.*

### Typedefs

- typedef uint32\_t `expchannel_t`  
*EXT channel identifier.*
- typedef void(\* `extcallback_t`) (EXTDriver \*extp, expchannel\_t channel)  
*Type of an EXT generic notification callback.*

## Functions

- void `ext_lld_init` (void)  
*Low level EXT driver initialization.*
- void `ext_lld_start` (EXTDriver \*extp)  
*Configures and activates the EXT peripheral.*
- void `ext_lld_stop` (EXTDriver \*extp)  
*Deactivates the EXT peripheral.*
- void `ext_lld_channel_enable` (EXTDriver \*extp, expchannel\_t channel)  
*Enables an EXT channel.*
- void `ext_lld_channel_disable` (EXTDriver \*extp, expchannel\_t channel)  
*Disables an EXT channel.*

### 9.16.1 Detailed Description

PLATFORM EXT subsystem low level driver header.

## 9.17 gpt.c File Reference

GPT Driver code.

```
#include "hal.h"
```

## Functions

- void `gptInit` (void)  
*GPT Driver initialization.*
- void `gptObjectInit` (GPTDriver \*gptp)  
*Initializes the standard part of a `GPTDriver` structure.*
- void `gptStart` (GPTDriver \*gptp, const GPTConfig \*config)  
*Configures and activates the GPT peripheral.*
- void `gptStop` (GPTDriver \*gptp)  
*Deactivates the GPT peripheral.*
- void `gptChangeInterval` (GPTDriver \*gptp, gptcnt\_t interval)  
*Changes the interval of GPT peripheral.*
- void `gptStartContinuous` (GPTDriver \*gptp, gptcnt\_t interval)  
*Starts the timer in continuous mode.*
- void `gptStartContinuousl` (GPTDriver \*gptp, gptcnt\_t interval)  
*Starts the timer in continuous mode.*
- void `gptStartOneShot` (GPTDriver \*gptp, gptcnt\_t interval)  
*Starts the timer in one shot mode.*
- void `gptStartOneShotl` (GPTDriver \*gptp, gptcnt\_t interval)  
*Starts the timer in one shot mode.*
- void `gptStopTimer` (GPTDriver \*gptp)  
*Stops the timer.*
- void `gptStopTimerl` (GPTDriver \*gptp)  
*Stops the timer.*
- void `gptPolledDelay` (GPTDriver \*gptp, gptcnt\_t interval)  
*Starts the timer in one shot mode and waits for completion.*

### 9.17.1 Detailed Description

GPT Driver code.

## 9.18 gpt.h File Reference

GPT Driver macros and structures.

```
#include "gpt_lld.h"
```

### Macros

- `#define gptChangeInterval(gptp, interval)`  
*Changes the interval of GPT peripheral.*
- `#define gptGetIntervalX(gptp) gpt_lld_get_interval(gptp)`  
*Returns the interval of GPT peripheral.*
- `#define gptGetCounterX(gptp) gpt_lld_get_counter(gptp)`  
*Returns the counter value of GPT peripheral.*

### Typedefs

- `typedef struct GPTDriver GPTDriver`  
*Type of a structure representing a GPT driver.*
- `typedef void(* gptcallback_t) (GPTDriver *gptp)`  
*GPT notification callback type.*

### Enumerations

- `enum gptstate_t {`  
 `GPT_UNINIT = 0, GPT_STOP = 1, GPT_READY = 2, GPT_CONTINUOUS = 3,`  
 `GPT_ONESHOT = 4 }`  
*Driver state machine possible states.*

### Functions

- `void gptInit (void)`  
*GPT Driver initialization.*
- `void gptObjectInit (GPTDriver *gptp)`  
*Initializes the standard part of a `GPTDriver` structure.*
- `void gptStart (GPTDriver *gptp, const GPTConfig *config)`  
*Configures and activates the GPT peripheral.*
- `void gptStop (GPTDriver *gptp)`  
*Deactivates the GPT peripheral.*
- `void gptStartContinuous (GPTDriver *gptp, gptcnt_t interval)`  
*Starts the timer in continuous mode.*
- `void gptStartContinuousI (GPTDriver *gptp, gptcnt_t interval)`  
*Starts the timer in continuous mode.*
- `void gptChangeInterval (GPTDriver *gptp, gptcnt_t interval)`  
*Changes the interval of GPT peripheral.*

- void `gptStartOneShot (GPTDriver *gptp, gptcnt_t interval)`  
*Starts the timer in one shot mode.*
- void `gptStartOneShotl (GPTDriver *gptp, gptcnt_t interval)`  
*Starts the timer in one shot mode.*
- void `gptStopTimer (GPTDriver *gptp)`  
*Stops the timer.*
- void `gptStopTimerl (GPTDriver *gptp)`  
*Stops the timer.*
- void `gptPolledDelay (GPTDriver *gptp, gptcnt_t interval)`  
*Starts the timer in one shot mode and waits for completion.*

### 9.18.1 Detailed Description

GPT Driver macros and structures.

## 9.19 gpt\_lld.c File Reference

PLATFORM GPT subsystem low level driver source.

```
#include "hal.h"
```

### Functions

- void `gpt_lld_init (void)`  
*Low level GPT driver initialization.*
- void `gpt_lld_start (GPTDriver *gptp)`  
*Configures and activates the GPT peripheral.*
- void `gpt_lld_stop (GPTDriver *gptp)`  
*Deactivates the GPT peripheral.*
- void `gpt_lld_start_timer (GPTDriver *gptp, gptcnt_t interval)`  
*Starts the timer in continuous mode.*
- void `gpt_lld_stop_timer (GPTDriver *gptp)`  
*Stops the timer.*
- void `gpt_lld_polled_delay (GPTDriver *gptp, gptcnt_t interval)`  
*Starts the timer in one shot mode and waits for completion.*

### Variables

- `GPTDriver GPTD1`  
*GPTD1 driver identifier.*

### 9.19.1 Detailed Description

PLATFORM GPT subsystem low level driver source.

## 9.20 gpt\_lld.h File Reference

PLATFORM GPT subsystem low level driver header.

## Data Structures

- struct **GPTConfig**  
*Driver configuration structure.*
- struct **GPTDriver**  
*Structure representing a GPT driver.*

## Macros

- #define **gpt\_lld\_change\_interval**(gptp, interval)  
*Changes the interval of GPT peripheral.*

### PLATFORM configuration options

- #define **PLATFORM\_GPT\_USE\_GPT1** FALSE  
*GPTD1 driver enable switch.*

## Typedefs

- typedef uint32\_t **gptfreq\_t**  
*GPT frequency type.*
- typedef uint16\_t **gptcnt\_t**  
*GPT counter type.*

## Functions

- void **gpt\_lld\_init** (void)  
*Low level GPT driver initialization.*
- void **gpt\_lld\_start** (GPTDriver \*gptp)  
*Configures and activates the GPT peripheral.*
- void **gpt\_lld\_stop** (GPTDriver \*gptp)  
*Deactivates the GPT peripheral.*
- void **gpt\_lld\_start\_timer** (GPTDriver \*gptp, gptcnt\_t interval)  
*Starts the timer in continuous mode.*
- void **gpt\_lld\_stop\_timer** (GPTDriver \*gptp)  
*Stops the timer.*
- void **gpt\_lld\_polled\_delay** (GPTDriver \*gptp, gptcnt\_t interval)  
*Starts the timer in one shot mode and waits for completion.*

### 9.20.1 Detailed Description

PLATFORM GPT subsystem low level driver header.

## 9.21 hal.c File Reference

HAL subsystem code.

```
#include "hal.h"
```

## Functions

- void **halInit** (void)

*HAL initialization.*

### 9.21.1 Detailed Description

HAL subsystem code.

## 9.22 hal.h File Reference

HAL subsystem header.

```
#include "osal.h"
#include "board.h"
#include "halconf.h"
#include "hal_lld.h"
#include "hal_streams.h"
#include "hal_channels.h"
#include "hal_files.h"
#include "hal_ioblock.h"
#include "hal_mmc.h"
#include "hal_buffers.h"
#include "hal_queues.h"
#include "pal.h"
#include "adc.h"
#include "can.h"
#include "dac.h"
#include "ext.h"
#include "gpt.h"
#include "i2c.h"
#include "i2s.h"
#include "icu.h"
#include "mac.h"
#include "mii.h"
#include "pwm.h"
#include "rtc.h"
#include "serial.h"
#include "sdc.h"
#include "spi.h"
#include "uart.h"
#include "usb.h"
#include "wdg.h"
#include "mmc_spi.h"
#include "serial_usb.h"
#include "hal_community.h"
```

## Macros

- #define **\_CHIBIOS\_HAL\_**  
*ChibiOS/HAL identification macro.*
- #define **CH\_HAL\_STABLE** 1  
*Stable release flag.*

### ChibiOS/HAL version identification

- `#define HAL_VERSION "4.0.3"`  
*HAL version string.*
- `#define CH_HAL_MAJOR 4`  
*HAL version major number.*
- `#define CH_HAL_MINOR 0`  
*HAL version minor number.*
- `#define CH_HAL_PATCH 3`  
*HAL version patch number.*

### Return codes

- `#define HAL_SUCCESS false`
- `#define HAL_FAILED true`

## Functions

- `void halInit (void)`  
*HAL initialization.*

### 9.22.1 Detailed Description

HAL subsystem header.

## 9.23 hal\_buffers.c File Reference

I/O Buffers code.

```
#include <string.h>
#include "hal.h"
```

## Functions

- `void ibqObjectInit (input_buffers_queue_t *ibqp, uint8_t *bp, size_t size, size_t n, bqnotify_t infy, void *link)`  
*Initializes an input buffers queue object.*
- `void ibqReset (input_buffers_queue_t *ibqp)`  
*Resets an input buffers queue.*
- `uint8_t * ibqGetEmptyBuffer (input_buffers_queue_t *ibqp)`  
*Gets the next empty buffer from the queue.*
- `void ibqPostFullBuffer (input_buffers_queue_t *ibqp, size_t size)`  
*Posts a new filled buffer to the queue.*
- `msg_t ibqGetFullBufferTimeout (input_buffers_queue_t *ibqp, systime_t timeout)`  
*Gets the next filled buffer from the queue.*
- `msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t *ibqp, systime_t timeout)`  
*Gets the next filled buffer from the queue.*
- `void ibqReleaseEmptyBuffer (input_buffers_queue_t *ibqp)`  
*Releases the buffer back in the queue.*
- `void ibqReleaseEmptyBufferS (input_buffers_queue_t *ibqp)`  
*Releases the buffer back in the queue.*
- `msg_t ibqGetTimeout (input_buffers_queue_t *ibqp, systime_t timeout)`

- `size_t ibqReadTimeout (input_buffers_queue_t *ibqp, uint8_t *bp, size_t n, systime_t timeout)`

*Input queue read with timeout.*
- `void obqObjectInit (output_buffers_queue_t *obqp, uint8_t *bp, size_t size, size_t n, bqnotify_t onfy, void *link)`

*Initializes an output buffers queue object.*
- `void obqResetI (output_buffers_queue_t *obqp)`

*Resets an output buffers queue.*
- `uint8_t * obqGetFullBufferI (output_buffers_queue_t *obqp, size_t *sizep)`

*Gets the next filled buffer from the queue.*
- `void obqReleaseEmptyBufferI (output_buffers_queue_t *obqp)`

*Releases the next filled buffer back in the queue.*
- `msg_t obqGetEmptyBufferTimeout (output_buffers_queue_t *obqp, systime_t timeout)`

*Gets the next empty buffer from the queue.*
- `msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t *obqp, systime_t timeout)`

*Gets the next empty buffer from the queue.*
- `void obqPostFullBuffer (output_buffers_queue_t *obqp, size_t size)`

*Posts a new filled buffer to the queue.*
- `void obqPostFullBufferS (output_buffers_queue_t *obqp, size_t size)`

*Posts a new filled buffer to the queue.*
- `msg_t obqPutTimeout (output_buffers_queue_t *obqp, uint8_t b, systime_t timeout)`

*Output queue write with timeout.*
- `size_t obqWriteTimeout (output_buffers_queue_t *obqp, const uint8_t *bp, size_t n, systime_t timeout)`

*Output queue write with timeout.*
- `bool obqTryFlushI (output_buffers_queue_t *obqp)`

*Flushes the current, partially filled, buffer to the queue.*
- `void obqFlush (output_buffers_queue_t *obqp)`

*Flushes the current, partially filled, buffer to the queue.*

### 9.23.1 Detailed Description

I/O Buffers code.

## 9.24 hal\_buffers.h File Reference

I/O Buffers macros and structures.

### Data Structures

- struct `io_buffers_queue`

*Structure of a generic buffers queue.*

### Macros

- `#define BQ_BUFFER_SIZE(n, size) (((size_t)(size) + sizeof (size_t)) * (size_t)(n))`

*Computes the size of a buffers queue buffer size.*

### Macro Functions

- `#define bqSizeX(bqp) ((bqp)->bn)`  
*Returns the queue's number of buffers.*
- `#define bqSpaceI(bqp) ((bqp)->bcounter)`  
*Return the ready buffers number.*
- `#define bqGetLinkX(bqp) ((bqp)->link)`  
*Returns the queue application-defined link.*
- `#define ibqIsEmptyI(ibqp) ((bool)(bqSpaceI(ibqp) == 0U))`  
*Evaluates to TRUE if the specified input buffers queue is empty.*
- `#define ibqIsFullI(ibqp)`  
*Evaluates to TRUE if the specified input buffers queue is full.*
- `#define obqIsEmptyI(obqp)`  
*Evaluates to true if the specified output buffers queue is empty.*
- `#define obqIsFullI(obqp) ((bool)(bqSpaceI(obqp) == 0U))`  
*Evaluates to true if the specified output buffers queue is full.*

## Typedefs

- `typedef struct io_buffers_queue io_buffers_queue_t`  
*Type of a generic queue of buffers.*
- `typedef void(* bqnotify_t) (io_buffers_queue_t *bqp)`  
*Double buffer notification callback type.*
- `typedef io_buffers_queue_t input_buffers_queue_t`  
*Type of an input buffers queue.*
- `typedef io_buffers_queue_t output_buffers_queue_t`  
*Type of an output buffers queue.*

## Functions

- `void ibqObjectInit (input_buffers_queue_t *ibqp, uint8_t *bp, size_t size, size_t n, bqnotify_t infy, void *link)`  
*Initializes an input buffers queue object.*
- `void ibqResetI (input_buffers_queue_t *ibqp)`  
*Resets an input buffers queue.*
- `uint8_t * ibqGetEmptyBufferI (input_buffers_queue_t *ibqp)`  
*Gets the next empty buffer from the queue.*
- `void ibqPostFullBufferI (input_buffers_queue_t *ibqp, size_t size)`  
*Posts a new filled buffer to the queue.*
- `msg_t ibqGetFullBufferTimeout (input_buffers_queue_t *ibqp, systime_t timeout)`  
*Gets the next filled buffer from the queue.*
- `msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t *ibqp, systime_t timeout)`  
*Gets the next filled buffer from the queue.*
- `void ibqReleaseEmptyBuffer (input_buffers_queue_t *ibqp)`  
*Releases the buffer back in the queue.*
- `void ibqReleaseEmptyBufferS (input_buffers_queue_t *ibqp)`  
*Releases the buffer back in the queue.*
- `msg_t ibqGetTimeout (input_buffers_queue_t *ibqp, systime_t timeout)`  
*Input queue read with timeout.*
- `size_t ibqReadTimeout (input_buffers_queue_t *ibqp, uint8_t *bp, size_t n, systime_t timeout)`  
*Input queue read with timeout.*
- `void obqObjectInit (output_buffers_queue_t *obqp, uint8_t *bp, size_t size, size_t n, bqnotify_t onfy, void *link)`  
*Initializes an output buffers queue object.*
- `void obqResetI (output_buffers_queue_t *obqp)`

- Resets an output buffers queue.
- `uint8_t * obqGetFullBufferI (output_buffers_queue_t *obqp, size_t *sizep)`  
Gets the next filled buffer from the queue.
- `void obqReleaseEmptyBufferI (output_buffers_queue_t *obqp)`  
Releases the next filled buffer back in the queue.
- `msg_t obqGetEmptyBufferTimeout (output_buffers_queue_t *obqp, systime_t timeout)`  
Gets the next empty buffer from the queue.
- `msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t *obqp, systime_t timeout)`  
Gets the next empty buffer from the queue.
- `void obqPostFullBuffer (output_buffers_queue_t *obqp, size_t size)`  
Posts a new filled buffer to the queue.
- `void obqPostFullBufferS (output_buffers_queue_t *obqp, size_t size)`  
Posts a new filled buffer to the queue.
- `msg_t obqPutTimeout (output_buffers_queue_t *obqp, uint8_t b, systime_t timeout)`  
Output queue write with timeout.
- `size_t obqWriteTimeout (output_buffers_queue_t *obqp, const uint8_t *bp, size_t n, systime_t timeout)`  
Output queue write with timeout.
- `bool obqTryFlushI (output_buffers_queue_t *obqp)`  
Flushes the current, partially filled, buffer to the queue.
- `void obqFlush (output_buffers_queue_t *obqp)`  
Flushes the current, partially filled, buffer to the queue.

#### 9.24.1 Detailed Description

I/O Buffers macros and structures.

### 9.25 hal\_channels.h File Reference

I/O channels access.

#### Data Structures

- struct `BaseChannelVMT`  
*BaseChannel virtual methods table.*
- struct `BaseChannel`  
*Base channel class.*
- struct `BaseAsynchronousChannelVMT`  
*BaseAsynchronousChannel virtual methods table.*
- struct `BaseAsynchronousChannel`  
*Base asynchronous channel class.*

#### Macros

- `#define _base_channel_methods`  
*BaseChannel specific methods.*
- `#define _base_channel_data _base_sequential_stream_data`  
*BaseChannel specific data.*
- `#define _base_asynchronous_channel_methods _base_channel_methods \`  
*BaseAsynchronousChannel specific methods.*

- `#define _base_asynchronous_channel_data`  
`BaseAsynchronousChannel` specific data.

### Macro Functions (BaseChannel)

- `#define chnPutTimeout(ip, b, time) ((ip)->vmt->putt(ip, b, time))`  
*Channel blocking byte write with timeout.*
- `#define chnGetTimeout(ip, time) ((ip)->vmt->gett(ip, time))`  
*Channel blocking byte read with timeout.*
- `#define chnWrite(ip, bp, n) streamWrite(ip, bp, n)`  
*Channel blocking write.*
- `#define chnWriteTimeout(ip, bp, n, time) ((ip)->vmt->writet(ip, bp, n, time))`  
*Channel blocking write with timeout.*
- `#define chnRead(ip, bp, n) streamRead(ip, bp, n)`  
*Channel blocking read.*
- `#define chnReadTimeout(ip, bp, n, time) ((ip)->vmt->readt(ip, bp, n, time))`  
*Channel blocking read with timeout.*

### I/O status flags added to the event listener

- `#define CHN_NO_ERROR (eventflags_t)0`  
*No pending conditions.*
- `#define CHN_CONNECTED (eventflags_t)1`  
*Connection happened.*
- `#define CHN_DISCONNECTED (eventflags_t)2`  
*Disconnection happened.*
- `#define CHN_INPUT_AVAILABLE (eventflags_t)4`  
*Data available in the input queue.*
- `#define CHN_OUTPUT_EMPTY (eventflags_t)8`  
*Output queue empty.*
- `#define CHN_TRANSMISSION_END (eventflags_t)16`  
*Transmission end.*

### Macro Functions (BaseAsynchronousChannel)

- `#define chnGetEventSource(ip) (&((ip)->event))`  
*Returns the I/O condition event source.*
- `#define chnAddFlagsI(ip, flags)`  
*Adds status flags to the listeners's flags mask.*

## 9.25.1 Detailed Description

I/O channels access.

This header defines an abstract interface useful to access generic I/O serial devices in a standardized way.

## 9.26 hal\_files.h File Reference

Data files.

### Data Structures

- struct `FileStreamVMT`  
`FileStream` virtual methods table.
- struct `FileStream`  
`Base file stream class.`

## Macros

- `#define _file_stream_methods`  
*FileStream specific methods.*
- `#define _file_stream_data_base_sequential_stream_data`  
*FileStream specific data.*

## Files return codes

- `#define FILE_OK STM_OK`  
*No error return code.*
- `#define FILE_ERROR STM_TIMEOUT`  
*Error code from the file stream methods.*
- `#define FILE_EOF STM_RESET`  
*End-of-file condition for file get/put methods.*

## Macro Functions (FileStream)

- `#define fileStreamWrite(ip, bp, n) streamWrite(ip, bp, n)`  
*File stream write.*
- `#define fileStreamRead(ip, bp, n) streamRead(ip, bp, n)`  
*File stream read.*
- `#define fileStreamPut(ip, b) streamPut(ip, b)`  
*File stream blocking byte write.*
- `#define fileStreamGet(ip) streamGet(ip)`  
*File stream blocking byte read.*
- `#define fileStreamClose(ip) ((ip)->vmt->close(ip))`  
*File Stream close.*
- `#define fileStreamGetError(ip) ((ip)->vmt->geterror(ip))`  
*Returns an implementation dependent error code.*
- `#define fileStreamGetSize(ip) ((ip)->vmt->getsize(ip))`  
*Returns the current file size.*
- `#define fileStreamGetPosition(ip) ((ip)->vmt->getposition(ip))`  
*Returns the current file pointer position.*
- `#define fileStreamSeek(ip, offset) ((ip)->vmt->lseek(ip, offset))`  
*Moves the file current pointer to an absolute position.*

## Typedefs

- `typedef uint32_t fileoffset_t`  
*File offset type.*

### 9.26.1 Detailed Description

Data files.

This header defines abstract interfaces useful to access generic data files in a standardized way.

## 9.27 hal\_ioblock.h File Reference

I/O block devices access.

## Data Structures

- struct `BlockDeviceInfo`  
*Block device info.*
- struct `BaseBlockDeviceVMT`  
*BaseBlockDevice virtual methods table.*
- struct `BaseBlockDevice`  
*Base block device class.*

## Macros

- `#define _base_block_device_methods`  
*BaseBlockDevice specific methods.*
- `#define _base_block_device_data`  
*BaseBlockDevice specific data.*

## Macro Functions (BaseBlockDevice)

- `#define blkGetDriverState(ip) ((ip)->state)`  
*Returns the driver state.*
- `#define blkIsTransferring(ip)`  
*Determines if the device is transferring data.*
- `#define blkIsInserted(ip) ((ip)->vmt->is_inserted(ip))`  
*Returns the media insertion status.*
- `#define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))`  
*Returns the media write protection status.*
- `#define blkConnect(ip) ((ip)->vmt->connect(ip))`  
*Performs the initialization procedure on the block device.*
- `#define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))`  
*Terminates operations on the block device.*
- `#define blkRead(ip, startblk, buf, n) ((ip)->vmt->read(ip, startblk, buf, n))`  
*Reads one or more blocks.*
- `#define blkWrite(ip, startblk, buf, n) ((ip)->vmt->write(ip, startblk, buf, n))`  
*Writes one or more blocks.*
- `#define blkSync(ip) ((ip)->vmt->sync(ip))`  
*Ensures write synchronization.*
- `#define blkGetInfo(ip, bdip) ((ip)->vmt->get_info(ip, bdip))`  
*Returns a media information structure.*

## Enumerations

- enum `blkstate_t` {
   
`BLK_UNINIT = 0, BLK_STOP = 1, BLK_ACTIVE = 2, BLK_CONNECTING = 3,`
  
`BLK_DISCONNECTING = 4, BLK_READY = 5, BLK_READING = 6, BLK_WRITING = 7,`
  
`BLK_SYNCING = 8 }`
  
*Driver state machine possible states.*

### 9.27.1 Detailed Description

I/O block devices access.

This header defines an abstract interface useful to access generic I/O block devices in a standardized way.

## 9.28 hal\_lld.c File Reference

PLATFORM HAL subsystem low level driver source.

```
#include "hal.h"
```

### Functions

- void [hal\\_lld\\_init](#) (void)  
*Low level HAL driver initialization.*

#### 9.28.1 Detailed Description

PLATFORM HAL subsystem low level driver source.

## 9.29 hal\_lld.h File Reference

PLATFORM HAL subsystem low level driver header.

### Macros

#### Platform identification macros

- #define **PLATFORM\_NAME** "templates"

### Functions

- void [hal\\_lld\\_init](#) (void)  
*Low level HAL driver initialization.*

#### 9.29.1 Detailed Description

PLATFORM HAL subsystem low level driver header.

## 9.30 hal\_mmc.c File Reference

MMC/SD cards common code.

```
#include "hal.h"
```

### Functions

- uint32\_t [\\_mmc\\_get\\_slice](#) (const uint32\_t \*data, uint32\_t end, uint32\_t start)  
*Gets a bit field from a words array.*
- uint32\_t [\\_mmc\\_get\\_capacity](#) (const uint32\_t \*csd)  
*Extract card capacity from a CSD.*
- uint32\_t [\\_mmc\\_get\\_capacity\\_ext](#) (const uint8\_t \*ext\_csd)

- Extract MMC card capacity from EXT\_CSD.
- void `_mmcsd_unpack_sdc_cid` (const `MMCSDBlockDevice` \*`sdc`, `unpacked_sdc_cid_t` \*`cidsdc`)  
*Unpacks SDC CID array in structure.*
- void `_mmcsd_unpack_mmc_cid` (const `MMCSDBlockDevice` \*`sdc`, `unpacked_mmc_cid_t` \*`cidmmc`)  
*Unpacks MMC CID array in structure.*
- void `_mmcsd_unpack_csd_mmc` (const `MMCSDBlockDevice` \*`sdc`, `unpacked_mmc_csd_t` \*`csdmmc`)  
*Unpacks MMC CSD array in structure.*
- void `_mmcsd_unpack_csd_v10` (const `MMCSDBlockDevice` \*`sdc`, `unpacked_sdc_csd_10_t` \*`csd10`)  
*Unpacks SDC CSD v1.0 array in structure.*
- void `_mmcsd_unpack_csd_v20` (const `MMCSDBlockDevice` \*`sdc`, `unpacked_sdc_csd_20_t` \*`csd20`)  
*Unpacks SDC CSD v2.0 array in structure.*

### 9.30.1 Detailed Description

MMC/SD cards common code.

## 9.31 hal\_mmc.h File Reference

MMC/SD cards common header.

### Data Structures

- struct `MMCSDBlockDeviceVMT`  
*MMCSDBlockDevice virtual methods table.*
- struct `MMCSDBlockDevice`  
*MCC/SD block device class.*
- struct `unpacked_sdc_cid_t`  
*Unpacked CID register from SDC.*
- struct `unpacked_mmc_cid_t`  
*Unpacked CID register from MMC.*
- struct `unpacked_sdc_csd_10_t`  
*Unpacked CSD v1.0 register from SDC.*
- struct `unpacked_sdc_csd_20_t`  
*Unpacked CSD v2.0 register from SDC.*
- struct `unpacked_mmc_csd_t`  
*Unpacked CSD register from MMC.*

### Macros

- #define `MMCSD_BLOCK_SIZE` 512U  
*Fixed block size for MMC/SD block devices.*
- #define `MMCSD_R1_ERROR_MASK` 0xFDFFE008U  
*Mask of error bits in R1 responses.*
- #define `MMCSD_CMD8_PATTERN` 0x000001AAU  
*Fixed pattern for CMD8.*
- #define `_mmcsd_block_device_methods` `_base_block_device_methods`  
*MMCSDBlockDevice specific methods.*
- #define `_mmcsd_block_device_data`  
*MMCSDBlockDevice specific data.*

### SD/MMC status conditions

- #define **MMCSD\_STS\_IDLE** 0U
- #define **MMCSD\_STS\_READY** 1U
- #define **MMCSD\_STS\_IDENT** 2U
- #define **MMCSD\_STS\_STBY** 3U
- #define **MMCSD\_STS\_TRAN** 4U
- #define **MMCSD\_STS\_DATA** 5U
- #define **MMCSD\_STS\_RCV** 6U
- #define **MMCSD\_STS\_PRG** 7U
- #define **MMCSD\_STS\_DIS** 8U

### SD/MMC commands

- #define **MMCSD\_CMD\_GO\_IDLE\_STATE** 0U
- #define **MMCSD\_CMD\_INIT** 1U
- #define **MMCSD\_CMD\_ALL\_SEND\_CID** 2U
- #define **MMCSD\_CMD\_SEND\_RELATIVE\_ADDR** 3U
- #define **MMCSD\_CMD\_SET\_BUS\_WIDTH** 6U
- #define **MMCSD\_CMD\_SWITCH** MMCSD\_CMD\_SET\_BUS\_WIDTH
- #define **MMCSD\_CMD\_SEL\_DESEL\_CARD** 7U
- #define **MMCSD\_CMD\_SEND\_IF\_COND** 8U
- #define **MMCSD\_CMD\_SEND\_EXT\_CSD** MMCSD\_CMD\_SEND\_IF\_COND
- #define **MMCSD\_CMD\_SEND\_CSD** 9U
- #define **MMCSD\_CMD\_SEND\_CID** 10U
- #define **MMCSD\_CMD\_STOP\_TRANSMISSION** 12U
- #define **MMCSD\_CMD\_SEND\_STATUS** 13U
- #define **MMCSD\_CMD\_SET\_BLOCKLEN** 16U
- #define **MMCSD\_CMD\_READ\_SINGLE\_BLOCK** 17U
- #define **MMCSD\_CMD\_READ\_MULTIPLE\_BLOCK** 18U
- #define **MMCSD\_CMD\_SET\_BLOCK\_COUNT** 23U
- #define **MMCSD\_CMD\_WRITE\_BLOCK** 24U
- #define **MMCSD\_CMD\_WRITE\_MULTIPLE\_BLOCK** 25U
- #define **MMCSD\_CMD\_ERASE\_RW\_BLK\_START** 32U
- #define **MMCSD\_CMD\_ERASE\_RW\_BLK\_END** 33U
- #define **MMCSD\_CMD\_ERASE** 38U
- #define **MMCSD\_CMD\_APP\_OP\_COND** 41U
- #define **MMCSD\_CMD\_LOCK\_UNLOCK** 42U
- #define **MMCSD\_CMD\_APP\_CMD** 55U
- #define **MMCSD\_CMD\_READ\_OCR** 58U

### CSD record offsets

- #define **MMCSD\_CSD\_MMC\_CSD\_STRUCTURE\_SLICE** 127U,126U  
*Slice position of values in CSD register.*
- #define **MMCSD\_CSD\_MMC\_SPEC\_VERS\_SLICE** 125U,122U
- #define **MMCSD\_CSD\_MMC\_TAAC\_SLICE** 119U,112U
- #define **MMCSD\_CSD\_MMC\_NSAC\_SLICE** 111U,104U
- #define **MMCSD\_CSD\_MMC\_TRAN\_SPEED\_SLICE** 103U,96U
- #define **MMCSD\_CSD\_MMC\_CCC\_SLICE** 95U,84U
- #define **MMCSD\_CSD\_MMC\_READ\_BL\_LEN\_SLICE** 83U,80U
- #define **MMCSD\_CSD\_MMC\_READ\_BL\_PARTIAL\_SLICE** 79U,79U
- #define **MMCSD\_CSD\_MMC\_WRITE\_BLK\_MISALIGN\_SLICE** 78U,78U
- #define **MMCSD\_CSD\_MMC\_READ\_BLK\_MISALIGN\_SLICE** 77U,77U
- #define **MMCSD\_CSD\_MMC\_DSR\_IMP\_SLICE** 76U,76U
- #define **MMCSD\_CSD\_MMC\_C\_SIZE\_SLICE** 73U,62U
- #define **MMCSD\_CSD\_MMC\_VDD\_R\_CURR\_MIN\_SLICE** 61U,59U
- #define **MMCSD\_CSD\_MMC\_VDD\_R\_CURR\_MAX\_SLICE** 58U,56U
- #define **MMCSD\_CSD\_MMC\_VDD\_W\_CURR\_MIN\_SLICE** 55U,53U
- #define **MMCSD\_CSD\_MMC\_VDD\_W\_CURR\_MAX\_SLICE** 52U,50U
- #define **MMCSD\_CSD\_MMC\_C\_SIZE\_MULT\_SLICE** 49U,47U
- #define **MMCSD\_CSD\_MMC\_ERASE\_GRP\_SIZE\_SLICE** 46U,42U
- #define **MMCSD\_CSD\_MMC\_ERASE\_GRP\_MULT\_SLICE** 41U,37U

- #define **MMCSD\_CSD\_MMC\_WP\_GRP\_SIZE\_SLICE** 36U,32U
- #define **MMCSD\_CSD\_MMC\_WP\_GRP\_ENABLE\_SLICE** 31U,31U
- #define **MMCSD\_CSD\_MMC\_DEFAULT\_ECC\_SLICE** 30U,29U
- #define **MMCSD\_CSD\_MMC\_R2W\_FACTOR\_SLICE** 28U,26U
- #define **MMCSD\_CSD\_MMC\_WRITE\_BL\_LEN\_SLICE** 25U,22U
- #define **MMCSD\_CSD\_MMC\_WRITE\_BL\_PARTIAL\_SLICE** 21U,21U
- #define **MMCSD\_CSD\_MMC\_CONTENT\_PROT\_APP\_SLICE** 16U,16U
- #define **MMCSD\_CSD\_MMC\_FILE\_FORMAT\_GRP\_SLICE** 15U,15U
- #define **MMCSD\_CSD\_MMC\_COPY\_SLICE** 14U,14U
- #define **MMCSD\_CSD\_MMC\_PERM\_WRITE\_PROTECT\_SLICE** 13U,13U
- #define **MMCSD\_CSD\_MMC\_TMP\_WRITE\_PROTECT\_SLICE** 12U,12U
- #define **MMCSD\_CSD\_MMC\_FILE\_FORMAT\_SLICE** 11U,10U
- #define **MMCSD\_CSD\_MMC\_ECC\_SLICE** 9U,8U
- #define **MMCSD\_CSD\_MMC\_CRC\_SLICE** 7U,1U
- #define **MMCSD\_CSD\_20\_CRC\_SLICE** 7U,1U
- #define **MMCSD\_CSD\_20\_FILE\_FORMAT\_SLICE** 11U,10U
- #define **MMCSD\_CSD\_20\_TMP\_WRITE\_PROTECT\_SLICE** 12U,12U
- #define **MMCSD\_CSD\_20\_PERM\_WRITE\_PROTECT\_SLICE** 13U,13U
- #define **MMCSD\_CSD\_20\_COPY\_SLICE** 14U,14U
- #define **MMCSD\_CSD\_20\_FILE\_FORMAT\_GRP\_SLICE** 15U,15U
- #define **MMCSD\_CSD\_20\_WRITE\_BL\_PARTIAL\_SLICE** 21U,21U
- #define **MMCSD\_CSD\_20\_WRITE\_BL\_LEN\_SLICE** 25U,12U
- #define **MMCSD\_CSD\_20\_R2W\_FACTOR\_SLICE** 28U,26U
- #define **MMCSD\_CSD\_20\_WP\_GRP\_ENABLE\_SLICE** 31U,31U
- #define **MMCSD\_CSD\_20\_WP\_GRP\_SIZE\_SLICE** 38U,32U
- #define **MMCSD\_CSD\_20\_ERASE\_SECTOR\_SIZE\_SLICE** 45U,39U
- #define **MMCSD\_CSD\_20\_ERASE\_BLK\_EN\_SLICE** 46U,46U
- #define **MMCSD\_CSD\_20\_C\_SIZE\_SLICE** 69U,48U
- #define **MMCSD\_CSD\_20\_DSR\_IMP\_SLICE** 76U,76U
- #define **MMCSD\_CSD\_20\_READ\_BLK\_MISALIGN\_SLICE** 77U,77U
- #define **MMCSD\_CSD\_20\_WRITE\_BLK\_MISALIGN\_SLICE** 78U,78U
- #define **MMCSD\_CSD\_20\_READ\_BL\_PARTIAL\_SLICE** 79U,79U
- #define **MMCSD\_CSD\_20\_READ\_BL\_LEN\_SLICE** 83U,80U
- #define **MMCSD\_CSD\_20\_CCC\_SLICE** 95U,84U
- #define **MMCSD\_CSD\_20\_TRANS\_SPEED\_SLICE** 103U,96U
- #define **MMCSD\_CSD\_20\_NSAC\_SLICE** 111U,104U
- #define **MMCSD\_CSD\_20\_TAAC\_SLICE** 119U,112U
- #define **MMCSD\_CSD\_20\_CSD\_STRUCTURE\_SLICE** 127U,126U
- #define **MMCSD\_CSD\_10\_CRC\_SLICE** MMCSD\_CSD\_20\_CRC\_SLICE
- #define **MMCSD\_CSD\_10\_FILE\_FORMAT\_SLICE** MMCSD\_CSD\_20\_FILE\_FORMAT\_SLICE
- #define **MMCSD\_CSD\_10\_TMP\_WRITE\_PROTECT\_SLICE** MMCSD\_CSD\_20\_TMP\_WRITE\_PROTECT\_SLICE
- #define **MMCSD\_CSD\_10\_PERM\_WRITE\_PROTECT\_SLICE** MMCSD\_CSD\_20\_PERM\_WRITE\_PROTECT\_SLICE
- #define **MMCSD\_CSD\_10\_COPY\_SLICE** MMCSD\_CSD\_20\_COPY\_SLICE
- #define **MMCSD\_CSD\_10\_FILE\_FORMAT\_GRP\_SLICE** MMCSD\_CSD\_20\_FILE\_FORMAT\_GRP\_SLICE
- #define **MMCSD\_CSD\_10\_WRITE\_BL\_PARTIAL\_SLICE** MMCSD\_CSD\_20\_WRITE\_BL\_PARTIAL\_SLICE
- #define **MMCSD\_CSD\_10\_WRITE\_BL\_LEN\_SLICE** MMCSD\_CSD\_20\_WRITE\_BL\_LEN\_SLICE
- #define **MMCSD\_CSD\_10\_R2W\_FACTOR\_SLICE** MMCSD\_CSD\_20\_R2W\_FACTOR\_SLICE
- #define **MMCSD\_CSD\_10\_WP\_GRP\_ENABLE\_SLICE** MMCSD\_CSD\_20\_WP\_GRP\_ENABLE\_SLICE
- #define **MMCSD\_CSD\_10\_WP\_GRP\_SIZE\_SLICE** MMCSD\_CSD\_20\_WP\_GRP\_SIZE\_SLICE
- #define **MMCSD\_CSD\_10\_ERASE\_SECTOR\_SIZE\_SLICE** MMCSD\_CSD\_20\_ERASE\_SECTOR\_SIZE\_SLICE
- #define **MMCSD\_CSD\_10\_ERASE\_BLK\_EN\_SLICE** MMCSD\_CSD\_20\_ERASE\_BLK\_EN\_SLICE
- #define **MMCSD\_CSD\_10\_C\_SIZE\_MULT\_SLICE** 49U,47U
- #define **MMCSD\_CSD\_10\_VDD\_W\_CURR\_MAX\_SLICE** 52U,50U
- #define **MMCSD\_CSD\_10\_VDD\_W\_CURR\_MIN\_SLICE** 55U,53U
- #define **MMCSD\_CSD\_10\_VDD\_R\_CURR\_MAX\_SLICE** 58U,56U
- #define **MMCSD\_CSD\_10\_VDD\_R\_CURR\_MIX\_SLICE** 61U,59U
- #define **MMCSD\_CSD\_10\_C\_SIZE\_SLICE** 73U,62U
- #define **MMCSD\_CSD\_10\_DSR\_IMP\_SLICE** MMCSD\_CSD\_20\_DSR\_IMP\_SLICE

- #define **MMCSD\_CSD\_10\_READ\_BLK\_MISALIGN\_SLICE** MMCSD\_CSD\_20\_READ\_BLK\_MISALIG↔  
N\_SLICE
- #define **MMCSD\_CSD\_10\_WRITE\_BLK\_MISALIGN\_SLICE** MMCSD\_CSD\_20\_WRITE\_BLK\_MISALI↔  
GN\_SLICE
- #define **MMCSD\_CSD\_10\_READ\_BL\_PARTIAL\_SLICE** MMCSD\_CSD\_20\_READ\_BL\_PARTIAL\_SLI↔  
CE
- #define **MMCSD\_CSD\_10\_READ\_BL\_LEN\_SLICE** 83U,80U
- #define **MMCSD\_CSD\_10\_CCC\_SLICE** MMCSD\_CSD\_20\_CCC\_SLICE
- #define **MMCSD\_CSD\_10\_TRANS\_SPEED\_SLICE** MMCSD\_CSD\_20\_TRANS\_SPEED\_SLICE
- #define **MMCSD\_CSD\_10\_NSAC\_SLICE** MMCSD\_CSD\_20\_NSAC\_SLICE
- #define **MMCSD\_CSD\_10\_TAAC\_SLICE** MMCSD\_CSD\_20\_TAAC\_SLICE
- #define **MMCSD\_CSD\_10\_CSD\_STRUCTURE\_SLICE** MMCSD\_CSD\_20\_CSD\_STRUCTURE\_SLICE

### CID record offsets

- #define **MMCSD\_CID\_SDC\_CRC\_SLICE** 7U,1U  
*Slice position of values in CID register.*
- #define **MMCSD\_CID\_SDC\_MDT\_M\_SLICE** 11U,8U
- #define **MMCSD\_CID\_SDC\_MDT\_Y\_SLICE** 19U,12U
- #define **MMCSD\_CID\_SDC\_PSN\_SLICE** 55U,24U
- #define **MMCSD\_CID\_SDC\_PRV\_M\_SLICE** 59U,56U
- #define **MMCSD\_CID\_SDC\_PRV\_N\_SLICE** 63U,60U
- #define **MMCSD\_CID\_SDC\_PNM0\_SLICE** 71U,64U
- #define **MMCSD\_CID\_SDC\_PNM1\_SLICE** 79U,72U
- #define **MMCSD\_CID\_SDC\_PNM2\_SLICE** 87U,80U
- #define **MMCSD\_CID\_SDC\_PNM3\_SLICE** 95U,88U
- #define **MMCSD\_CID\_SDC\_PNM4\_SLICE** 103U,96U
- #define **MMCSD\_CID\_SDC\_OID\_SLICE** 119U,104U
- #define **MMCSD\_CID\_SDC\_MID\_SLICE** 127U,120U
- #define **MMCSD\_CID\_MM\_CRC\_SLICE** 7U,1U
- #define **MMCSD\_CID\_MM\_MDT\_Y\_SLICE** 11U,8U
- #define **MMCSD\_CID\_MM\_MDT\_M\_SLICE** 15U,12U
- #define **MMCSD\_CID\_MM\_PSN\_SLICE** 47U,16U
- #define **MMCSD\_CID\_MM\_PRV\_M\_SLICE** 51U,48U
- #define **MMCSD\_CID\_MM\_PRV\_N\_SLICE** 55U,52U
- #define **MMCSD\_CID\_MM\_PNM0\_SLICE** 63U,56U
- #define **MMCSD\_CID\_MM\_PNM1\_SLICE** 71U,64U
- #define **MMCSD\_CID\_MM\_PNM2\_SLICE** 79U,72U
- #define **MMCSD\_CID\_MM\_PNM3\_SLICE** 87U,80U
- #define **MMCSD\_CID\_MM\_PNM4\_SLICE** 95U,88U
- #define **MMCSD\_CID\_MM\_PNM5\_SLICE** 103U,96U
- #define **MMCSD\_CID\_MM\_OID\_SLICE** 119U,104U
- #define **MMCSD\_CID\_MM\_MID\_SLICE** 127U,120U

### R1 response utilities

- #define **MMCSD\_R1\_ERROR**(r1) (((r1) & **MMCSD\_R1\_ERROR\_MASK**) != 0U)  
*Evaluates to TRUE if the R1 response contains error flags.*
- #define **MMCSD\_R1\_STS**(r1) (((r1) >> 9U) & 15U)  
*Returns the status field of an R1 response.*
- #define **MMCSD\_R1\_IS\_CARD\_LOCKED**(r1) (((((r1) >> 21U) & 1U) != 0U)  
*Evaluates to TRUE if the R1 response indicates a locked card.*

### Macro Functions

- #define **mmcSdGetCardCapacity**(ip) ((ip)->capacity)  
*Returns the card capacity in blocks.*

## Functions

- `uint32_t _mmcsd_get_slice (const uint32_t *data, uint32_t end, uint32_t start)`  
*Gets a bit field from a words array.*
- `uint32_t _mmcsd_get_capacity (const uint32_t *csd)`  
*Extract card capacity from a CSD.*
- `uint32_t _mmcsd_get_capacity_ext (const uint8_t *ext_csd)`  
*Extract MMC card capacity from EXT\_CSD.*
- `void _mmcsd_unpack_sdc_cid (const MMCSDBlockDevice *sdcp, unpacked_sdc_cid_t *cidsdc)`  
*Unpacks SDC CID array in structure.*
- `void _mmcsd_unpack_mmc_cid (const MMCSDBlockDevice *sdcp, unpacked_mmc_cid_t *cidmmc)`  
*Unpacks MMC CID array in structure.*
- `void _mmcsd_unpack_csd_mmc (const MMCSDBlockDevice *sdcp, unpacked_mmc_csd_t *csdmmc)`  
*Unpacks MMC CSD array in structure.*
- `void _mmcsd_unpack_csd_v10 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_10_t *csd10)`  
*Unpacks SDC CSD v1.0 array in structure.*
- `void _mmcsd_unpack_csd_v20 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_20_t *csd20)`  
*Unpacks SDC CSD v2.0 array in structure.*

### 9.31.1 Detailed Description

MMC/SD cards common header.

This header defines an abstract interface useful to access MMC/SD I/O block devices in a standardized way.

## 9.32 hal\_queues.c File Reference

I/O Queues code.

```
#include "hal.h"
```

## Functions

- `void iqObjectInit (input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)`  
*Initializes an input queue.*
- `void iqResetl (input_queue_t *iqp)`  
*Resets an input queue.*
- `msg_t iqPutl (input_queue_t *iqp, uint8_t b)`  
*Input queue write.*
- `msg_t iqGetTimeout (input_queue_t *iqp, systime_t timeout)`  
*Input queue read with timeout.*
- `size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, systime_t timeout)`  
*Input queue read with timeout.*
- `void oqObjectInit (output_queue_t *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)`  
*Initializes an output queue.*
- `void oqResetl (output_queue_t *oqp)`  
*Resets an output queue.*
- `msg_t oqPutTimeout (output_queue_t *oqp, uint8_t b, systime_t timeout)`  
*Output queue write with timeout.*
- `msg_t oqGetl (output_queue_t *oqp)`

- `size_t oqWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, systime_t timeout)`  
*Output queue write with timeout.*

### 9.32.1 Detailed Description

I/O Queues code.

## 9.33 hal\_queues.h File Reference

I/O Queues macros and structures.

### Data Structures

- struct `io_queue`  
*Generic I/O queue structure.*

### Macros

#### Queue functions returned status value

- `#define Q_OK MSG_OK`  
*Operation successful.*
- `#define Q_TIMEOUT MSG_TIMEOUT`  
*Timeout condition.*
- `#define Q_RESET MSG_RESET`  
*Queue has been reset.*
- `#define Q_EMPTY (msg_t)-3`  
*Queue empty.*
- `#define Q_FULL (msg_t)-4`  
*Queue full.*

#### Macro Functions

- `#define qSizeX(qp)`  
*Returns the queue's buffer size.*
- `#define qSpaceI(qp) ((qp)->q_counter)`  
*Queue space.*
- `#define qGetLink(qp) ((qp)->q_link)`  
*Returns the queue application-defined link.*
- `#define iqGetFullI(iqp) qSpaceI(iqp)`  
*Returns the filled space into an input queue.*
- `#define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpaceI(iqp))`  
*Returns the empty space into an input queue.*
- `#define iqIsEmptyI(iqp) ((bool)(qSpaceI(iqp) == 0U))`  
*Evaluates to true if the specified input queue is empty.*
- `#define iqIsFullI(iqp)`  
*Evaluates to true if the specified input queue is full.*
- `#define iqGet(iqp) iqGetTimeout(iqp, TIME_INFINITE)`  
*Input queue read.*
- `#define oqGetFullI(oqp) (qSizeX(oqp) - qSpaceI(oqp))`  
*Returns the filled space into an output queue.*
- `#define oqGetEmptyI(oqp) qSpaceI(oqp)`  
*Returns the empty space into an output queue.*

- `#define oqlsEmptyl(oqp)`  
*Evaluates to true if the specified output queue is empty.*
- `#define oqlsFulll(oqp) ((bool)(qSpaceI(oqp) == 0U))`  
*Evaluates to true if the specified output queue is full.*
- `#define oqPut(oqp, b) oqPutTimeout(oqp, b, TIME_INFINITE)`  
*Output queue write.*

## Typedefs

- `typedef struct io_queue io_queue_t`  
*Type of a generic I/O queue structure.*
- `typedef void(* qnotify_t) (io_queue_t *qp)`  
*Queue notification callback type.*
- `typedef io_queue_t input_queue_t`  
*Type of an input queue structure.*
- `typedef io_queue_t output_queue_t`  
*Type of an output queue structure.*

## Functions

- `void iqObjectInit (input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)`  
*Initializes an input queue.*
- `void iqResetl (input_queue_t *iqp)`  
*Resets an input queue.*
- `msg_t iqPutl (input_queue_t *iqp, uint8_t b)`  
*Input queue write.*
- `msg_t iqGetTimeout (input_queue_t *iqp, systime_t timeout)`  
*Input queue read with timeout.*
- `size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, systime_t timeout)`  
*Input queue read with timeout.*
- `void oqObjectInit (output_queue_t *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)`  
*Initializes an output queue.*
- `void oqResetl (output_queue_t *oqp)`  
*Resets an output queue.*
- `msg_t oqPutTimeout (output_queue_t *oqp, uint8_t b, systime_t timeout)`  
*Output queue write with timeout.*
- `msg_t oqGetl (output_queue_t *oqp)`  
*Output queue read.*
- `size_t oqWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, systime_t timeout)`  
*Output queue write with timeout.*

### 9.33.1 Detailed Description

I/O Queues macros and structures.

## 9.34 hal\_streams.h File Reference

Data streams.

## Data Structures

- struct **BaseSequentialStreamVMT**  
*BaseSequentialStream* virtual methods table.
- struct **BaseSequentialStream**  
*Base stream class.*

## Macros

- #define **\_base\_sequential\_stream\_methods**  
*BaseSequentialStream* specific methods.
- #define **\_base\_sequential\_stream\_data**  
*BaseSequentialStream* specific data.

## Streams return codes

- #define **STM\_OK** MSG\_OK
- #define **STM\_TIMEOUT** MSG\_TIMEOUT
- #define **STM\_RESET** MSG\_RESET

## Macro Functions (BaseSequentialStream)

- #define **streamWrite**(ip, bp, n) ((ip)->vmt->write(ip, bp, n))  
*Sequential Stream write.*
- #define **streamRead**(ip, bp, n) ((ip)->vmt->read(ip, bp, n))  
*Sequential Stream read.*
- #define **streamPut**(ip, b) ((ip)->vmt->put(ip, b))  
*Sequential Stream blocking byte write.*
- #define **streamGet**(ip) ((ip)->vmt->get(ip))  
*Sequential Stream blocking byte read.*

### 9.34.1 Detailed Description

Data streams.

This header defines abstract interfaces useful to access generic data streams in a standardized way.

## 9.35 halconf.h File Reference

HAL configuration header.

```
#include "mcuconf.h"
```

## Macros

### Drivers enable switches

- #define **HAL\_USE\_PAL** TRUE  
*Enables the PAL subsystem.*
- #define **HAL\_USE\_ADC** TRUE  
*Enables the ADC subsystem.*
- #define **HAL\_USE\_CAN** TRUE  
*Enables the CAN subsystem.*

- #define `HAL_USE_DAC` FALSE  
*Enables the DAC subsystem.*
- #define `HAL_USE_EXT` TRUE  
*Enables the EXT subsystem.*
- #define `HAL_USE_GPT` TRUE  
*Enables the GPT subsystem.*
- #define `HAL_USE_I2C` TRUE  
*Enables the I2C subsystem.*
- #define `HAL_USE_I2S` TRUE  
*Enables the I2S subsystem.*
- #define `HAL_USE_ICU` TRUE  
*Enables the ICU subsystem.*
- #define `HAL_USE_MAC` TRUE  
*Enables the MAC subsystem.*
- #define `HAL_USE_MMCSPI` TRUE  
*Enables the MMC\_SPI subsystem.*
- #define `HAL_USE_PWM` TRUE  
*Enables the PWM subsystem.*
- #define `HAL_USE_RTC` TRUE  
*Enables the RTC subsystem.*
- #define `HAL_USE_SDC` TRUE  
*Enables the SDC subsystem.*
- #define `HAL_USE_SERIAL` TRUE  
*Enables the SERIAL subsystem.*
- #define `HAL_USE_SERIALUSB` TRUE  
*Enables the SERIAL over USB subsystem.*
- #define `HAL_USE_SPI` TRUE  
*Enables the SPI subsystem.*
- #define `HAL_USE_UART` TRUE  
*Enables the UART subsystem.*
- #define `HAL_USE_USB` TRUE  
*Enables the USB subsystem.*
- #define `HAL_USE_WDG` TRUE  
*Enables the WDG subsystem.*

#### ADC driver related setting

- #define `ADC_USE_WAIT` TRUE  
*Enables synchronous APIs.*
- #define `ADC_USE_MUTUAL_EXCLUSION` TRUE  
*Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.*

#### CAN driver related setting

- #define `CAN_USE_SLEEP_MODE` TRUE  
*Sleep mode related APIs inclusion switch.*

#### I2C driver related setting

- #define `I2C_USE_MUTUAL_EXCLUSION` TRUE  
*Enables the mutual exclusion APIs on the I2C bus.*

#### MAC driver related setting

- #define `MAC_USE_ZERO_COPY` TRUE  
*Enables an event sources for incoming packets.*
- #define `MAC_USE_EVENTS` TRUE  
*Enables an event sources for incoming packets.*

### MMC\_SPI driver related setting

- #define `MMC_NICE_WAITING` TRUE  
*Delays insertions.*

### SDC driver related setting

- #define `SDC_INIT_RETRY` 100  
*Number of initialization attempts before rejecting the card.*
- #define `SDC_MMC_SUPPORT` TRUE  
*Include support for MMC cards.*
- #define `SDC_NICE_WAITING` TRUE  
*Delays insertions.*

### SERIAL driver related setting

- #define `SERIAL_DEFAULT_BITRATE` 38400  
*Default bit rate.*
- #define `SERIAL_BUFFERS_SIZE` 16  
*Serial buffers size.*

### SERIAL\_USB driver related setting

- #define `SERIAL_USB_BUFFERS_SIZE` 256  
*Serial over USB buffers size.*
- #define `SERIAL_USB_BUFFERS_NUMBER` 2  
*Serial over USB number of buffers.*

### SPI driver related setting

- #define `SPI_USE_WAIT` TRUE  
*Enables synchronous APIs.*
- #define `SPI_USE_MUTUAL_EXCLUSION` TRUE  
*Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.*

### UART driver related setting

- #define `UART_USE_WAIT` TRUE  
*Enables synchronous APIs.*
- #define `UART_USE_MUTUAL_EXCLUSION` TRUE  
*Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.*

### USB driver related setting

- #define `USB_USE_WAIT` TRUE  
*Enables synchronous APIs.*

#### 9.35.1 Detailed Description

HAL configuration header.

HAL configuration file, this file allows to enable or disable the various device drivers from your application. You may also use this file in order to override the device drivers default settings.

## 9.36 i2c.c File Reference

I2C Driver code.

```
#include "hal.h"
```

### Functions

- void `i2cInit` (void)
 

*I2C Driver initialization.*
- void `i2cObjectInit` (`I2CDriver` \*`i2cp`)
 

*Initializes the standard part of a `I2CDriver` structure.*
- void `i2cStart` (`I2CDriver` \*`i2cp`, const `I2CConfig` \*`config`)
 

*Configures and activates the I2C peripheral.*
- void `i2cStop` (`I2CDriver` \*`i2cp`)
 

*Deactivates the I2C peripheral.*
- `i2cflags_t i2cGetErrors` (`I2CDriver` \*`i2cp`)
 

*Returns the errors mask associated to the previous operation.*
- `msg_t i2cMasterTransmitTimeout` (`I2CDriver` \*`i2cp`, `i2caddr_t` `addr`, const `uint8_t` \*`txbuf`, `size_t` `txbytes`, `uint8_t` \*`rxbuf`, `size_t` `rxbytes`, `systime_t` `timeout`)
 

*Sends data via the I2C bus.*
- `msg_t i2cMasterReceiveTimeout` (`I2CDriver` \*`i2cp`, `i2caddr_t` `addr`, `uint8_t` \*`rxbuf`, `size_t` `rxbytes`, `systime_t` `timeout`)
 

*Receives data from the I2C bus.*
- void `i2cAcquireBus` (`I2CDriver` \*`i2cp`)
 

*Gains exclusive access to the I2C bus.*
- void `i2cReleaseBus` (`I2CDriver` \*`i2cp`)
 

*Releases exclusive access to the I2C bus.*

### 9.36.1 Detailed Description

I2C Driver code.

## 9.37 i2c.h File Reference

I2C Driver macros and structures.

```
#include "i2c_lld.h"
```

### Macros

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`

*Enables the mutual exclusion APIs on the I2C bus.*
- `#define _i2c_wakeup_isr(i2cp)`

*Wakes up the waiting thread notifying no errors.*
- `#define _i2c_wakeup_error_isr(i2cp)`

*Wakes up the waiting thread notifying errors.*
- `#define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)`

*Wrap `i2cMasterTransmitTimeout` function with `TIME_INFINITE` timeout.*

- #define `i2cMasterReceive`(`i2cp`, `addr`, `rxbuf`, `rxbytes`) (`i2cMasterReceiveTimeout`(`i2cp`, `addr`, `rxbuf`, `rxbytes`, `TIME_INFINITE`))

*Wrap i2cMasterReceiveTimeout function with TIME\_INFINITE timeout.*

### I2C bus error conditions

- #define `I2C_NO_ERROR` 0x00  
*No error.*
- #define `I2C_BUS_ERROR` 0x01  
*Bus Error.*
- #define `I2C_ARBITRATION_LOST` 0x02  
*Arbitration Lost.*
- #define `I2C_ACK_FAILURE` 0x04  
*Acknowledge Failure.*
- #define `I2C_OVERRUN` 0x08  
*Overrun/Underrun.*
- #define `I2C_PEC_ERROR` 0x10  
*PEC Error in reception.*
- #define `I2C_TIMEOUT` 0x20  
*Hardware timeout.*
- #define `I2C_SMB_ALERT` 0x40  
*SMBus Alert.*

### Enumerations

- enum `i2cstate_t` {  
  `I2C_UNINIT` = 0, `I2C_STOP` = 1, `I2C_READY` = 2, `I2C_ACTIVE_TX` = 3,  
  `I2C_ACTIVE_RX` = 4 }

*Driver state machine possible states.*

### Functions

- void `i2cInit` (void)  
*I2C Driver initialization.*
- void `i2cObjectInit` (`I2CDriver` \*`i2cp`)  
*Initializes the standard part of a I2CDriver structure.*
- void `i2cStart` (`I2CDriver` \*`i2cp`, const `I2CConfig` \*`config`)  
*Configures and activates the I2C peripheral.*
- void `i2cStop` (`I2CDriver` \*`i2cp`)  
*Deactivates the I2C peripheral.*
- `i2cflags_t` `i2cGetErrors` (`I2CDriver` \*`i2cp`)  
*Returns the errors mask associated to the previous operation.*
- `msg_t` `i2cMasterTransmitTimeout` (`I2CDriver` \*`i2cp`, `i2caddr_t` `addr`, const `uint8_t` \*`txbuf`, `size_t` `txbytes`, `uint8_t` \*`rxbuf`, `size_t` `rxbytes`, `systime_t` `timeout`)  
*Sends data via the I2C bus.*
- `msg_t` `i2cMasterReceiveTimeout` (`I2CDriver` \*`i2cp`, `i2caddr_t` `addr`, `uint8_t` \*`rxbuf`, `size_t` `rxbytes`, `systime_t` `timeout`)  
*Receives data from the I2C bus.*
- void `i2cAcquireBus` (`I2CDriver` \*`i2cp`)  
*Gains exclusive access to the I2C bus.*
- void `i2cReleaseBus` (`I2CDriver` \*`i2cp`)  
*Releases exclusive access to the I2C bus.*

### 9.37.1 Detailed Description

I2C Driver macros and structures.

## 9.38 i2c\_lld.c File Reference

PLATFORM I2C subsystem low level driver source.

```
#include "hal.h"
```

### Functions

- void [i2c\\_lld\\_init](#) (void)  
*Low level I2C driver initialization.*
- void [i2c\\_lld\\_start](#) (I2CDriver \*i2cp)  
*Configures and activates the I2C peripheral.*
- void [i2c\\_lld\\_stop](#) (I2CDriver \*i2cp)  
*Deactivates the I2C peripheral.*
- msg\_t [i2c\\_lld\\_master\\_receive\\_timeout](#) (I2CDriver \*i2cp, i2caddr\_t addr, uint8\_t \*rdbuf, size\_t rxbytes, systime\_t timeout)  
*Receives data via the I2C bus as master.*
- msg\_t [i2c\\_lld\\_master\\_transmit\\_timeout](#) (I2CDriver \*i2cp, i2caddr\_t addr, const uint8\_t \*txbuf, size\_t txbytes, uint8\_t \*rdbuf, size\_t rxbytes, systime\_t timeout)  
*Transmits data via the I2C bus as master.*

### Variables

- I2CDriver I2CD1  
*I2C1 driver identifier.*

### 9.38.1 Detailed Description

PLATFORM I2C subsystem low level driver source.

## 9.39 i2c\_lld.h File Reference

PLATFORM I2C subsystem low level driver header.

### Data Structures

- struct [I2CConfig](#)  
*Type of I2C driver configuration structure.*
- struct [I2CDriver](#)  
*Structure representing an I2C driver.*

## Macros

- `#define i2c_lld_get_errors(i2cp) ((i2cp)->errors)`  
*Get errors from I2C driver.*

## PLATFORM configuration options

- `#define PLATFORM_I2C_USE_I2C1 FALSE`  
*I2C1 driver enable switch.*

## Typedefs

- `typedef uint16_t i2caddr_t`  
*Type representing an I2C address.*
- `typedef uint32_t i2cflags_t`  
*Type of I2C Driver condition flags.*
- `typedef struct I2CDriver I2CDriver`  
*Type of a structure representing an I2C driver.*

## Functions

- `void i2c_lld_init (void)`  
*Low level I2C driver initialization.*
- `void i2c_lld_start (I2CDriver *i2cp)`  
*Configures and activates the I2C peripheral.*
- `void i2c_lld_stop (I2CDriver *i2cp)`  
*Deactivates the I2C peripheral.*
- `msg_t i2c_lld_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`  
*Transmits data via the I2C bus as master.*
- `msg_t i2c_lld_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`  
*Receives data via the I2C bus as master.*

### 9.39.1 Detailed Description

PLATFORM I2C subsystem low level driver header.

## 9.40 i2s.c File Reference

I2S Driver code.

```
#include "hal.h"
```

## Functions

- `void i2sInit (void)`  
*I2S Driver initialization.*
- `void i2sObjectInit (I2SDriver *i2sp)`  
*Initializes the standard part of a `I2SDriver` structure.*

- void `i2sStart (I2SDriver *i2sp, const I2SConfig *config)`  
*Configures and activates the I2S peripheral.*
- void `i2sStop (I2SDriver *i2sp)`  
*Deactivates the I2S peripheral.*
- void `i2sStartExchange (I2SDriver *i2sp)`  
*Starts a I2S data exchange.*
- void `i2sStopExchange (I2SDriver *i2sp)`  
*Stops the ongoing data exchange.*

#### 9.40.1 Detailed Description

I2S Driver code.

### 9.41 i2s.h File Reference

I2S Driver macros and structures.

```
#include "i2s_llld.h"
```

#### Macros

##### I2S modes

- #define `I2S_MODE_SLAVE` 0
- #define `I2S_MODE_MASTER` 1

##### Macro Functions

- #define `i2sStartExchange(i2sp)`  
*Starts a I2S data exchange.*
- #define `i2sStopExchange(i2sp)`  
*Stops the ongoing data exchange.*
- #define `_i2s_isr_half_code(i2sp)`  
*Common ISR code, half buffer event.*
- #define `_i2s_isr_full_code(i2sp)`  
*Common ISR code.*

#### Enumerations

- enum `i2sstate_t` {
 `I2S_UNINIT` = 0, `I2S_STOP` = 1, `I2S_READY` = 2, `I2S_ACTIVE` = 3,  
`I2S_COMPLETE` = 4 }

*Driver state machine possible states.*

#### Functions

- void `i2sInit (void)`  
*I2S Driver initialization.*
- void `i2sObjectInit (I2SDriver *i2sp)`  
*Initializes the standard part of a `I2SDriver` structure.*
- void `i2sStart (I2SDriver *i2sp, const I2SConfig *config)`

- void [i2sStop \(I2SDriver \\*i2sp\)](#)  
*Deactivates the I2S peripheral.*
- void [i2sStartExchange \(I2SDriver \\*i2sp\)](#)  
*Starts a I2S data exchange.*
- void [i2sStopExchange \(I2SDriver \\*i2sp\)](#)  
*Stops the ongoing data exchange.*

#### 9.41.1 Detailed Description

I2S Driver macros and structures.

### 9.42 i2s\_lld.c File Reference

PLATFORM I2S subsystem low level driver source.

```
#include "hal.h"
```

#### Functions

- void [i2s\\_lld\\_init \(void\)](#)  
*Low level I2S driver initialization.*
- void [i2s\\_lld\\_start \(I2SDriver \\*i2sp\)](#)  
*Configures and activates the I2S peripheral.*

#### Variables

- [I2SDriver I2SD1](#)  
*I2S2 driver identifier.*

#### 9.42.1 Detailed Description

PLATFORM I2S subsystem low level driver source.

### 9.43 i2s\_lld.h File Reference

PLATFORM I2S subsystem low level driver header.

#### Data Structures

- struct [I2SConfig](#)  
*Driver configuration structure.*
- struct [I2SDriver](#)  
*Structure representing an I2S driver.*

## Macros

### PLATFORM configuration options

- #define PLATFORM\_I2S\_USE\_I2S1 FALSE  
*I2SD1 driver enable switch.*

## Typedefs

- typedef struct I2SDriver I2SDriver  
*Type of a structure representing an I2S driver.*
- typedef void(\* i2scallback\_t) (I2SDriver \*i2sp, size\_t offset, size\_t n)  
*I2S notification callback type.*

## Functions

- void i2s\_lld\_init (void)  
*Low level I2S driver initialization.*
- void i2s\_lld\_start (I2SDriver \*i2sp)  
*Configures and activates the I2S peripheral.*

### 9.43.1 Detailed Description

PLATFORM I2S subsystem low level driver header.

## 9.44 icu.c File Reference

ICU Driver code.

```
#include "hal.h"
```

## Functions

- void icuInit (void)  
*ICU Driver initialization.*
- void icuObjectInit (ICUDriver \*icup)  
*Initializes the standard part of a ICUDriver structure.*
- void icuStart (ICUDriver \*icup, const ICUConfig \*config)  
*Configures and activates the ICU peripheral.*
- void icuStop (ICUDriver \*icup)  
*Deactivates the ICU peripheral.*
- void icuStartCapture (ICUDriver \*icup)  
*Starts the input capture.*
- bool icuWaitCapture (ICUDriver \*icup)  
*Waits for a completed capture.*
- void icuStopCapture (ICUDriver \*icup)  
*Stops the input capture.*
- void icuEnableNotifications (ICUDriver \*icup)  
*Enables notifications.*
- void icuDisableNotifications (ICUDriver \*icup)  
*Disables notifications.*

### 9.44.1 Detailed Description

ICU Driver code.

## 9.45 icu.h File Reference

ICU Driver macros and structures.

```
#include "icu_lld.h"
```

### Macros

#### Macro Functions

- `#define icuStartCapture(icup)`  
*Starts the input capture.*
- `#define icuStopCapture(icup)`  
*Stops the input capture.*
- `#define icuEnableNotifications(icup) icu_lld_enable_notifications(icup)`  
*Enables notifications.*
- `#define icuDisableNotifications(icup) icu_lld_disable_notifications(icup)`  
*Disables notifications.*
- `#define icuAreNotificationsEnabledX(icup) icu_lld_are_notifications_enabled(icup)`  
*Check on notifications status.*
- `#define icuGetWidthX(icup) icu_lld_get_width(icup)`  
*Returns the width of the latest pulse.*
- `#define icuGetPeriodX(icup) icu_lld_get_period(icup)`  
*Returns the width of the latest cycle.*

#### Low level driver helper macros

- `#define _icu_isr_invoke_width_cb(icup)`  
*Common ISR code, ICU width event.*
- `#define _icu_isr_invoke_period_cb(icup)`  
*Common ISR code, ICU period event.*
- `#define _icu_isr_invoke_overflow_cb(icup)`  
*Common ISR code, ICU timer overflow event.*

### Typedefs

- `typedef struct ICUDriver ICUDriver`  
*Type of a structure representing an ICU driver.*
- `typedef void(* icucallback_t) (ICUDriver *icup)`  
*ICU notification callback type.*

### Enumerations

- `enum icustate_t {`  
`ICU_UNINIT = 0, ICU_STOP = 1, ICU_READY = 2, ICU_WAITING = 3,`  
`ICU_ACTIVE = 4 }`  
*Driver state machine possible states.*

## Functions

- void **icuInit** (void)  
*ICU Driver initialization.*
- void **icuObjectInit** (ICUDriver \*icup)  
*Initializes the standard part of a `ICUDriver` structure.*
- void **icuStart** (ICUDriver \*icup, const ICUConfig \*config)  
*Configures and activates the ICU peripheral.*
- void **icuStop** (ICUDriver \*icup)  
*Deactivates the ICU peripheral.*
- void **icuStartCapture** (ICUDriver \*icup)  
*Starts the input capture.*
- bool **icuWaitCapture** (ICUDriver \*icup)  
*Waits for a completed capture.*
- void **icuStopCapture** (ICUDriver \*icup)  
*Stops the input capture.*
- void **icuEnableNotifications** (ICUDriver \*icup)  
*Enables notifications.*
- void **icuDisableNotifications** (ICUDriver \*icup)  
*Disables notifications.*

### 9.45.1 Detailed Description

ICU Driver macros and structures.

## 9.46 icu\_lld.c File Reference

PLATFORM ADC subsystem low level driver source.

```
#include "hal.h"
```

## Functions

- void **icu\_lld\_init** (void)  
*Low level ICU driver initialization.*
- void **icu\_lld\_start** (ICUDriver \*icup)  
*Configures and activates the ICU peripheral.*
- void **icu\_lld\_stop** (ICUDriver \*icup)  
*Deactivates the ICU peripheral.*
- void **icu\_lld\_start\_capture** (ICUDriver \*icup)  
*Starts the input capture.*
- bool **icu\_lld\_wait\_capture** (ICUDriver \*icup)  
*Waits for a completed capture.*
- void **icu\_lld\_stop\_capture** (ICUDriver \*icup)  
*Stops the input capture.*
- void **icu\_lld\_enable\_notifications** (ICUDriver \*icup)  
*Enables notifications.*
- void **icu\_lld\_disable\_notifications** (ICUDriver \*icup)  
*Disables notifications.*

## Variables

- **ICUDriver ICUD1**

*ICUD1 driver identifier.*

### 9.46.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

## 9.47 icu\_ll.h File Reference

PLATFORM ICU subsystem low level driver header.

### Data Structures

- struct **ICUConfig**

*Driver configuration structure.*

- struct **ICUDriver**

*Structure representing an ICU driver.*

### Macros

- #define **icu\_ll\_get\_width**(icup) 0

*Returns the width of the latest pulse.*

- #define **icu\_ll\_get\_period**(icup) 0

*Returns the width of the latest cycle.*

- #define **icu\_ll\_are\_notifications\_enabled**(icup) false

*Check on notifications status.*

### PLATFORM configuration options

- #define **PLATFORM\_ICU\_USE\_ICU1** FALSE

*ICUD1 driver enable switch.*

### Typedefs

- typedef uint32\_t **icufreq\_t**

*ICU frequency type.*

- typedef uint32\_t **icucnt\_t**

*ICU counter type.*

### Enumerations

- enum **icumode\_t** { **ICU\_INPUT\_ACTIVE\_HIGH** = 0, **ICU\_INPUT\_ACTIVE\_LOW** = 1 }

*ICU driver mode.*

## Functions

- void **icu\_lld\_init** (void)  
*Low level ICU driver initialization.*
- void **icu\_lld\_start** (ICUDriver \*icup)  
*Configures and activates the ICU peripheral.*
- void **icu\_lld\_stop** (ICUDriver \*icup)  
*Deactivates the ICU peripheral.*
- void **icu\_lld\_start\_capture** (ICUDriver \*icup)  
*Starts the input capture.*
- bool **icu\_lld\_wait\_capture** (ICUDriver \*icup)  
*Waits for a completed capture.*
- void **icu\_lld\_stop\_capture** (ICUDriver \*icup)  
*Stops the input capture.*
- void **icu\_lld\_enable\_notifications** (ICUDriver \*icup)  
*Enables notifications.*
- void **icu\_lld\_disable\_notifications** (ICUDriver \*icup)  
*Disables notifications.*

### 9.47.1 Detailed Description

PLATFORM ICU subsystem low level driver header.

## 9.48 mac.c File Reference

MAC Driver code.

```
#include "hal.h"
```

## Functions

- void **macInit** (void)  
*MAC Driver initialization.*
- void **macObjectInit** (MACDriver \*macp)  
*Initialize the standard part of a `MACDriver` structure.*
- void **macStart** (MACDriver \*macp, const MACConfig \*config)  
*Configures and activates the MAC peripheral.*
- void **macStop** (MACDriver \*macp)  
*Deactivates the MAC peripheral.*
- msg\_t **macWaitTransmitDescriptor** (MACDriver \*macp, MACTransmitDescriptor \*tdp, systime\_t timeout)  
*Allocates a transmission descriptor.*
- void **macReleaseTransmitDescriptor** (MACTransmitDescriptor \*tdp)  
*Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.*
- msg\_t **macWaitReceiveDescriptor** (MACDriver \*macp, MACReceiveDescriptor \*rdp, systime\_t timeout)  
*Waits for a received frame.*
- void **macReleaseReceiveDescriptor** (MACReceiveDescriptor \*rdp)  
*Releases a receive descriptor.*
- bool **macPollLinkStatus** (MACDriver \*macp)  
*Updates and returns the link status.*

### 9.48.1 Detailed Description

MAC Driver code.

## 9.49 mac.h File Reference

MAC Driver macros and structures.

```
#include "mac_lld.h"
```

### Macros

#### MAC configuration options

- #define `MAC_USE_ZERO_COPY` FALSE  
*Enables an event sources for incoming packets.*
- #define `MAC_USE_EVENTS` TRUE  
*Enables an event sources for incoming packets.*

#### Macro Functions

- #define `macGetReceiveEventSource`(macp) (&(macp)->rdevent)  
*Returns the received frames event source.*
- #define `macWriteTransmitDescriptor`(tdp, buf, size) `mac_lld_write_transmit_descriptor`(tdp, buf, size)  
*Writes to a transmit descriptor's stream.*
- #define `macReadReceiveDescriptor`(rdp, buf, size) `mac_lld_read_receive_descriptor`(rdp, buf, size)  
*Reads from a receive descriptor's stream.*
- #define `macGetNextTransmitBuffer`(tdp, size, sizep) `mac_lld_get_next_transmit_buffer`(tdp, size, sizep)  
*Returns a pointer to the next transmit buffer in the descriptor chain.*
- #define `macGetNextReceiveBuffer`(rdp, sizep) `mac_lld_get_next_receive_buffer`(rdp, sizep)  
*Returns a pointer to the next receive buffer in the descriptor chain.*

### Typedefs

- typedef struct `MACDriver` `MACDriver`  
*Type of a structure representing a MAC driver.*

### Enumerations

- enum `macstate_t` { `MAC_UNINIT` = 0, `MAC_STOP` = 1, `MAC_ACTIVE` = 2 }  
*Driver state machine possible states.*

### Functions

- void `macInit` (void)  
*MAC Driver initialization.*
- void `macObjectInit` (`MACDriver` \*macp)  
*Initialize the standard part of a `MACDriver` structure.*
- void `macStart` (`MACDriver` \*macp, const `MACConfig` \*config)  
*Configures and activates the MAC peripheral.*
- void `macStop` (`MACDriver` \*macp)

- **Deactivates the MAC peripheral.**
- **msg\_t macWaitTransmitDescriptor (MACDriver \*macp, MACTransmitDescriptor \*tdp, systime\_t timeout)**  
*Allocates a transmission descriptor.*
- **void macReleaseTransmitDescriptor (MACTransmitDescriptor \*tdp)**  
*Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.*
- **msg\_t macWaitReceiveDescriptor (MACDriver \*macp, MACReceiveDescriptor \*rdp, systime\_t timeout)**  
*Waits for a received frame.*
- **void macReleaseReceiveDescriptor (MACReceiveDescriptor \*rdp)**  
*Releases a receive descriptor.*
- **bool macPollLinkStatus (MACDriver \*macp)**  
*Updates and returns the link status.*

#### 9.49.1 Detailed Description

MAC Driver macros and structures.

## 9.50 mac\_lld.c File Reference

PLATFORM MAC subsystem low level driver source.

```
#include <string.h>
#include "hal.h"
#include "mii.h"
```

### Functions

- **void mac\_lld\_init (void)**  
*Low level MAC initialization.*
- **void mac\_lld\_start (MACDriver \*macp)**  
*Configures and activates the MAC peripheral.*
- **void mac\_lld\_stop (MACDriver \*macp)**  
*Deactivates the MAC peripheral.*
- **msg\_t mac\_lld\_get\_transmit\_descriptor (MACDriver \*macp, MACTransmitDescriptor \*tdp)**  
*Returns a transmission descriptor.*
- **void mac\_lld\_release\_transmit\_descriptor (MACTransmitDescriptor \*tdp)**  
*Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.*
- **msg\_t mac\_lld\_get\_receive\_descriptor (MACDriver \*macp, MACReceiveDescriptor \*rdp)**  
*Returns a receive descriptor.*
- **void mac\_lld\_release\_receive\_descriptor (MACReceiveDescriptor \*rdp)**  
*Releases a receive descriptor.*
- **bool mac\_lld\_poll\_link\_status (MACDriver \*macp)**  
*Updates and returns the link status.*
- **size\_t mac\_lld\_write\_transmit\_descriptor (MACTransmitDescriptor \*tdp, uint8\_t \*buf, size\_t size)**  
*Writes to a transmit descriptor's stream.*
- **size\_t mac\_lld\_read\_receive\_descriptor (MACReceiveDescriptor \*rdp, uint8\_t \*buf, size\_t size)**  
*Reads from a receive descriptor's stream.*
- **uint8\_t \* mac\_lld\_get\_next\_transmit\_buffer (MACTransmitDescriptor \*tdp, size\_t size, size\_t \*sizep)**  
*Returns a pointer to the next transmit buffer in the descriptor chain.*
- **const uint8\_t \* mac\_lld\_get\_next\_receive\_buffer (MACReceiveDescriptor \*rdp, size\_t \*sizep)**  
*Returns a pointer to the next receive buffer in the descriptor chain.*

## Variables

- **MACDriver ETHD1**  
*MAC1 driver identifier.*

### 9.50.1 Detailed Description

PLATFORM MAC subsystem low level driver source.

## 9.51 mac\_lld.h File Reference

PLATFORM MAC subsystem low level driver header.

### Data Structures

- struct **MACConfig**  
*Driver configuration structure.*
- struct **MACDriver**  
*Structure representing a MAC driver.*
- struct **MACTransmitDescriptor**  
*Structure representing a transmit descriptor.*
- struct **MACReceiveDescriptor**  
*Structure representing a receive descriptor.*

### Macros

- #define **MAC\_SUPPORTS\_ZERO\_COPY** TRUE  
*This implementation supports the zero-copy mode API.*

### PLATFORM configuration options

- #define **PLATFORM\_MAC\_USE\_MAC1** FALSE  
*MAC driver enable switch.*

### Functions

- void **mac\_lld\_init** (void)  
*Low level MAC initialization.*
- void **mac\_lld\_start** (**MACDriver** \*macp)  
*Configures and activates the MAC peripheral.*
- void **mac\_lld\_stop** (**MACDriver** \*macp)  
*Deactivates the MAC peripheral.*
- **msg\_t** **mac\_lld\_get\_transmit\_descriptor** (**MACDriver** \*macp, **MACTransmitDescriptor** \*tdp)  
*Returns a transmission descriptor.*
- void **mac\_lld\_release\_transmit\_descriptor** (**MACTransmitDescriptor** \*tdp)  
*Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.*
- **msg\_t** **mac\_lld\_get\_receive\_descriptor** (**MACDriver** \*macp, **MACReceiveDescriptor** \*rdp)  
*Returns a receive descriptor.*
- void **mac\_lld\_release\_receive\_descriptor** (**MACReceiveDescriptor** \*rdp)

- `bool mac_lld_poll_link_status (MACDriver *macp)`  
*Updates and returns the link status.*
- `size_t mac_lld_write_transmit_descriptor (MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)`  
*Writes to a transmit descriptor's stream.*
- `size_t mac_lld_read_receive_descriptor (MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)`  
*Reads from a receive descriptor's stream.*
- `uint8_t * mac_lld_get_next_transmit_buffer (MACTransmitDescriptor *tdp, size_t size, size_t *sizep)`  
*Returns a pointer to the next transmit buffer in the descriptor chain.*
- `const uint8_t * mac_lld_get_next_receive_buffer (MACReceiveDescriptor *rdp, size_t *sizep)`  
*Returns a pointer to the next receive buffer in the descriptor chain.*

### 9.51.1 Detailed Description

PLATFORM MAC subsystem low level driver header.

## 9.52 mii.h File Reference

MII macros and structures.

### Macros

#### Generic MII registers

- `#define MII_BMCR 0x00`
- `#define MII_BMSR 0x01`
- `#define MII_PHYSID1 0x02`
- `#define MII_PHYSID2 0x03`
- `#define MII_ADVERTISE 0x04`
- `#define MII_LPA 0x05`
- `#define MII_EXPANSION 0x06`
- `#define MII_ANNPTR 0x07`
- `#define MII_CTRL1000 0x09`
- `#define MII_STAT1000 0x0a`
- `#define MII_ESTATUS 0x0f`
- `#define MII_PHYSTS 0x10`
- `#define MII_MICR 0x11`
- `#define MII_DCOUNTER 0x12`
- `#define MII_FCS COUNTER 0x13`
- `#define MII_NWAYTEST 0x14`
- `#define MII_RERRCOUNTER 0x15`
- `#define MII_SREVISION 0x16`
- `#define MII_RESV1 0x17`
- `#define MII_LBRERROR 0x18`
- `#define MII_PHYADDR 0x19`
- `#define MII_RESV2 0x1a`
- `#define MII_TPISTATUS 0x1b`
- `#define MII_NCONFIG 0x1c`

#### Basic mode control register

- `#define BMCR_RESV 0x007f`
- `#define BMCR_CTST 0x0080`
- `#define BMCR_FULLDPLX 0x0100`
- `#define BMCR_ANRESTART 0x0200`
- `#define BMCR_ISOLATE 0x0400`

- #define **BMCR\_PDOWN** 0x0800
- #define **BMCR\_ANENABLE** 0x1000
- #define **BMCR\_SPEED100** 0x2000
- #define **BMCR\_LOOPBACK** 0x4000
- #define **BMCR\_RESET** 0x8000

### Basic mode status register

- #define **BMSR\_ERCAP** 0x0001
- #define **BMSR\_JCD** 0x0002
- #define **BMSR\_LSTATUS** 0x0004
- #define **BMSR\_ANEGCAPABLE** 0x0008
- #define **BMSR\_RFAULT** 0x0010
- #define **BMSR\_ANEGCOMPLETE** 0x0020
- #define **BMSR\_MFPRESUPPCAP** 0x0040
- #define **BMSR\_RESV** 0x0780
- #define **BMSR\_10HALF** 0x0800
- #define **BMSR\_10FULL** 0x1000
- #define **BMSR\_100HALF** 0x2000
- #define **BMSR\_100FULL** 0x4000
- #define **BMSR\_100BASE4** 0x8000

### Advertisement control register

- #define **ADVERTISE\_SLCT** 0x001f
- #define **ADVERTISE\_CSMA** 0x0001
- #define **ADVERTISE\_10HALF** 0x0020
- #define **ADVERTISE\_10FULL** 0x0040
- #define **ADVERTISE\_100HALF** 0x0080
- #define **ADVERTISE\_100FULL** 0x0100
- #define **ADVERTISE\_100BASE4** 0x0200
- #define **ADVERTISE\_PAUSE\_CAP** 0x0400
- #define **ADVERTISE\_PAUSE\_ASYM** 0x0800
- #define **ADVERTISE\_RESV** 0x1000
- #define **ADVERTISE\_RFAULT** 0x2000
- #define **ADVERTISE\_LPACK** 0x4000
- #define **ADVERTISE\_NPAGE** 0x8000
- #define **ADVERTISE\_FULL**
- #define **ADVERTISE\_ALL**

### Link partner ability register

- #define **LPA\_SLCT** 0x001f
- #define **LPA\_10HALF** 0x0020
- #define **LPA\_10FULL** 0x0040
- #define **LPA\_100HALF** 0x0080
- #define **LPA\_100FULL** 0x0100
- #define **LPA\_100BASE4** 0x0200
- #define **LPA\_PAUSE\_CAP** 0x0400
- #define **LPA\_PAUSE\_ASYM** 0x0800
- #define **LPA\_RESV** 0x1000
- #define **LPA\_RFAULT** 0x2000
- #define **LPA\_LPACK** 0x4000
- #define **LPA\_NPAGE** 0x8000
- #define **LPA\_DUPLEX** (**LPA\_10FULL** | **LPA\_100FULL**)
- #define **LPA\_100** (**LPA\_100FULL** | **LPA\_100HALF** | **LPA\_100BASE4**)

### Expansion register for auto-negotiation

- #define **EXPANSION\_NWAY** 0x0001
- #define **EXPANSION\_LCWP** 0x0002
- #define **EXPANSION\_ENABLENPAGE** 0x0004
- #define **EXPANSION\_NPCAPABLE** 0x0008

- #define EXPANSION\_MFAULTS 0x0010
- #define EXPANSION\_RESV 0xffe0

#### N-way test register

- #define NWAYTEST\_RESV1 0x00ff
- #define NWAYTEST\_LOOPBACK 0x0100
- #define NWAYTEST\_RESV2 0xfe00

#### PHY identifiers

- #define MII\_DM9161\_ID 0x0181b8a0
- #define MII\_AM79C875\_ID 0x00225540
- #define MII\_KS8721\_ID 0x00221610
- #define MII\_STE101P\_ID 0x00061C50
- #define MII\_DP83848I\_ID 0x20005C90
- #define MII\_LAN8710A\_ID 0x0007C0F1
- #define MII\_LAN8720\_ID 0x0007C0F0
- #define MII\_LAN8742A\_ID 0x0007C130

### 9.52.1 Detailed Description

MII macros and structures.

## 9.53 mmc\_spi.c File Reference

MMC over SPI driver code.

```
#include <string.h>
#include "hal.h"
```

### Functions

- static uint8\_t **crc7** (uint8\_t crc, const uint8\_t \*buffer, size\_t len)
 

*Calculate the MMC standard CRC-7 based on a lookup table.*
- static void **wait** (MMCDriver \*mmcp)
 

*Waits an idle condition.*
- static void **send\_hdr** (MMCDriver \*mmcp, uint8\_t cmd, uint32\_t arg)
 

*Sends a command header.*
- static uint8\_t **recv1** (MMCDriver \*mmcp)
 

*Receives a single byte response.*
- static uint8\_t **recv3** (MMCDriver \*mmcp, uint8\_t \*buffer)
 

*Receives a three byte response.*
- static uint8\_t **send\_command\_R1** (MMCDriver \*mmcp, uint8\_t cmd, uint32\_t arg)
 

*Sends a command an returns a single byte response.*
- static uint8\_t **send\_command\_R3** (MMCDriver \*mmcp, uint8\_t cmd, uint32\_t arg, uint8\_t \*response)
 

*Sends a command which returns a five bytes response (R3).*
- static bool **read\_CxD** (MMCDriver \*mmcp, uint8\_t cmd, uint32\_t cxd[4])
 

*Reads the CSD.*
- static void **sync** (MMCDriver \*mmcp)
 

*Waits that the card reaches an idle state.*
- void **mmcInit** (void)

- MMC over SPI driver initialization.*
- void `mmcObjectInit (MMCDriver *mmcp)`  
*Initializes an instance.*
  - void `mmcStart (MMCDriver *mmcp, const MMCCConfig *config)`  
*Configures and activates the MMC peripheral.*
  - void `mmcStop (MMCDriver *mmcp)`  
*Disables the MMC peripheral.*
  - bool `mmcConnect (MMCDriver *mmcp)`  
*Performs the initialization procedure on the inserted card.*
  - bool `mmcDisconnect (MMCDriver *mmcp)`  
*Brings the driver in a state safe for card removal.*
  - bool `mmcStartSequentialRead (MMCDriver *mmcp, uint32_t startblk)`  
*Starts a sequential read.*
  - bool `mmcSequentialRead (MMCDriver *mmcp, uint8_t *buffer)`  
*Reads a block within a sequential read operation.*
  - bool `mmcStopSequentialRead (MMCDriver *mmcp)`  
*Stops a sequential read gracefully.*
  - bool `mmcStartSequentialWrite (MMCDriver *mmcp, uint32_t startblk)`  
*Starts a sequential write.*
  - bool `mmcSequentialWrite (MMCDriver *mmcp, const uint8_t *buffer)`  
*Writes a block within a sequential write operation.*
  - bool `mmcStopSequentialWrite (MMCDriver *mmcp)`  
*Stops a sequential write gracefully.*
  - bool `mmcSync (MMCDriver *mmcp)`  
*Waits for card idle condition.*
  - bool `mmcGetInfo (MMCDriver *mmcp, BlockDeviceInfo *bdip)`  
*Returns the media info.*
  - bool `mmcErase (MMCDriver *mmcp, uint32_t startblk, uint32_t endblk)`  
*Erases blocks.*

## Variables

- static const struct MMCDriverVMT `mmc_vmt`  
*Virtual methods table.*
- static const uint8\_t `crc7_lookup_table [256]`  
*Lookup table for CRC-7 ( based on polynomial  $x^7 + x^3 + 1$ ).*

### 9.53.1 Detailed Description

MMC over SPI driver code.

## 9.54 mmc\_spi.h File Reference

MMC over SPI driver header.

## Data Structures

- struct **MMCConfig**  
*MMC/SD over SPI driver configuration structure.*
- struct **MMCDriverVMT**  
*MMCDriver virtual methods table.*
- struct **MMCDriver**  
*Structure representing a MMC/SD over SPI driver.*

## Macros

- #define **\_mmc\_driver\_methods\_mmcspi\_block\_device\_methods**  
*MMCDriver specific methods.*

### MMC\_SPI configuration options

- #define **MMC\_NICE\_WAITING** TRUE  
*Delays insertions.*

### Macro Functions

- #define **mmclsCardInserted**(mmcp) mmc\_lld\_is\_card\_inserted(mmcp)  
*Returns the card insertion status.*
- #define **mmclsWriteProtected**(mmcp) mmc\_lld\_is\_write\_protected(mmcp)  
*Returns the write protect status.*

## Functions

- void **mmcInit** (void)  
*MMC over SPI driver initialization.*
- void **mmcObjectInit** (MMCDriver \*mmcp)  
*Initializes an instance.*
- void **mmcStart** (MMCDriver \*mmcp, const MMCConfig \*config)  
*Configures and activates the MMC peripheral.*
- void **mmcStop** (MMCDriver \*mmcp)  
*Disables the MMC peripheral.*
- bool **mmcConnect** (MMCDriver \*mmcp)  
*Performs the initialization procedure on the inserted card.*
- bool **mmcDisconnect** (MMCDriver \*mmcp)  
*Brings the driver in a state safe for card removal.*
- bool **mmcStartSequentialRead** (MMCDriver \*mmcp, uint32\_t startblk)  
*Starts a sequential read.*
- bool **mmcSequentialRead** (MMCDriver \*mmcp, uint8\_t \*buffer)  
*Reads a block within a sequential read operation.*
- bool **mmcStopSequentialRead** (MMCDriver \*mmcp)  
*Stops a sequential read gracefully.*
- bool **mmcStartSequentialWrite** (MMCDriver \*mmcp, uint32\_t startblk)  
*Starts a sequential write.*
- bool **mmcSequentialWrite** (MMCDriver \*mmcp, const uint8\_t \*buffer)  
*Writes a block within a sequential write operation.*
- bool **mmcStopSequentialWrite** (MMCDriver \*mmcp)  
*Stops a sequential write gracefully.*

- `bool mmcSync (MMCDriver *mmcp)`  
*Waits for card idle condition.*
- `bool mmcGetInfo (MMCDriver *mmcp, BlockDeviceInfo *bdip)`  
*Returns the media info.*
- `bool mmcErase (MMCDriver *mmcp, uint32_t startblk, uint32_t endblk)`  
*Erases blocks.*

### 9.54.1 Detailed Description

MMC over SPI driver header.

## 9.55 osal.c File Reference

OSAL module code.

```
#include "osal.h"
```

### Functions

- `void osallInit (void)`  
*OSAL module initialization.*
- `void osalSysHalt (const char *reason)`  
*System halt with error message.*
- `void osalSysPolledDelayX (rtcnt_t cycles)`  
*Polled delay.*
- `void osalOsTimerHandler1 (void)`  
*System timer handler.*
- `void osalOsRescheduleS (void)`  
*Checks if a reschedule is required and performs it.*
- `systime_t osalOsGetSystemTimeX (void)`  
*Current system time.*
- `void osalThreadSleepS (systime_t time)`  
*Suspends the invoking thread for the specified time.*
- `void osalThreadSleep (systime_t time)`  
*Suspends the invoking thread for the specified time.*
- `msg_t osalThreadSuspendS (thread_reference_t *trp)`  
*Sends the current thread sleeping and sets a reference variable.*
- `msg_t osalThreadSuspendTimeoutS (thread_reference_t *trp, systime_t timeout)`  
*Sends the current thread sleeping and sets a reference variable.*
- `void osalThreadResume1 (thread_reference_t *trp, msg_t msg)`  
*Wakes up a thread waiting on a thread reference object.*
- `void osalThreadResumeS (thread_reference_t *trp, msg_t msg)`  
*Wakes up a thread waiting on a thread reference object.*
- `msg_t osalThreadEnqueueTimeoutS (threads_queue_t *tqp, systime_t timeout)`  
*Enqueues the caller thread.*
- `void osalThreadDequeueNext1 (threads_queue_t *tqp, msg_t msg)`  
*Dequeues and wakes up one thread from the queue, if any.*
- `void osalThreadDequeueAll (threads_queue_t *tqp, msg_t msg)`  
*Dequeues and wakes up all threads from the queue.*

- void `osalEventBroadcastFlagsI` (`event_source_t` \*esp, `eventflags_t` flags)  
*Add flags to an event source object.*
- void `osalEventBroadcastFlags` (`event_source_t` \*esp, `eventflags_t` flags)  
*Add flags to an event source object.*
- void `osalEventSetCallback` (`event_source_t` \*esp, `eventcallback_t` cb, void \*param)  
*Event callback setup.*
- void `osalMutexLock` (`mutex_t` \*mp)  
*Locks the specified mutex.*
- void `osalMutexUnlock` (`mutex_t` \*mp)  
*Unlocks the specified mutex.*

## Variables

- const char \* `osal_halt_msg`  
*Pointer to a halt error message.*

### 9.55.1 Detailed Description

OSAL module code.

## 9.56 osal.h File Reference

OSAL module header.

```
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
```

## Data Structures

- struct `event_source`  
*Events source object.*
- struct `threads_queue_t`  
*Type of a thread queue.*

## Macros

- #define `OSAL_DBG_ENABLE_ASSERTS` FALSE  
*Enables OSAL assertions.*
- #define `OSAL_DBG_ENABLE_CHECKS` FALSE  
*Enables OSAL functions parameters checks.*

## Common constants

- #define `FALSE` 0
- #define `TRUE` 1
- #define `OSAL_SUCCESS` false
- #define `OSAL_FAILED` true

## Messages

- #define **MSG\_OK** (`msg_t`)0
- #define **MSG\_RESET** (`msg_t`)-1
- #define **MSG\_TIMEOUT** (`msg_t`)-2

### Special time constants

- #define **TIME\_IMMEDIATE** ((`systime_t`)0)
- #define **TIME\_INFINITE** ((`systime_t`)-1)

### Systick modes.

- #define **OSAL\_ST\_MODE\_NONE** 0
- #define **OSAL\_ST\_MODE\_PERIODIC** 1
- #define **OSAL\_ST\_MODE\_FREERUNNING** 2

### Systick parameters.

- #define **OSAL\_ST\_RESOLUTION** 32  
*Size in bits of the `systick_t` type.*
- #define **OSAL\_ST\_FREQUENCY** 1000  
*Required systick frequency or resolution.*
- #define **OSAL\_ST\_MODE** OSAL\_ST\_MODE\_PERIODIC  
*Systick mode required by the underlying OS.*

### IRQ-related constants

- #define **OSAL\_IRQ\_PRIORITY\_LEVELS** 16U  
*Total priority levels.*
- #define **OSAL\_IRQ\_MAXIMUM\_PRIORITY** 0U  
*Highest IRQ priority for HAL drivers.*

### Debug related macros

- #define `osalDbgAssert(c, remark)`  
*Condition assertion.*
- #define `osalDbgCheck(c)`  
*Function parameters check.*
- #define `osalDbgCheckClassI()`  
*I-Class state check.*
- #define `osalDbgCheckClassS()`  
*S-Class state check.*

### IRQ service routines wrappers

- #define **OSAL\_IRQ\_IS\_VALID\_PRIORITY**(n) (((n) >= **OSAL\_IRQ\_MAXIMUM\_PRIORITY**) && ((n) < **OSAL\_IRQ\_PRIORITY\_LEVELS**))  
*Priority level verification macro.*
- #define **OSAL\_IRQ\_PROLOGUE**()  
*IRQ prologue code.*
- #define **OSAL\_IRQ\_EPILOGUE**()  
*IRQ epilogue code.*
- #define **OSAL\_IRQ\_HANDLER**(id) void id(void)  
*IRQ handler function declaration.*

### Time conversion utilities

- #define **OSAL\_S2ST**(sec) ((`systime_t`)((`uint32_t`)(sec) \* (`uint32_t`)**OSAL\_ST\_FREQUENCY**))  
*Seconds to system ticks.*
- #define **OSAL\_MS2ST**(msec)

- #define OSAL\_US2ST(usec)
 

*Microseconds to system ticks.*

### Time conversion utilities for the realtime counter

- #define OSAL\_S2RTC(freq, sec) ((freq) \* (sec))
 

*Seconds to realtime counter.*
- #define OSAL\_MS2RTC(freq, msec) (rtcnt\_t)((((freq) + 999UL) / 1000UL) \* (msec))
 

*Milliseconds to realtime counter.*
- #define OSAL\_US2RTC(freq, usec) (rtcnt\_t)((((freq) + 999999UL) / 1000000UL) \* (usec))
 

*Microseconds to realtime counter.*

### Sleep macros using absolute time

- #define osalThreadSleepSeconds(sec) osalThreadSleep(OSAL\_S2ST(sec))
 

*Delays the invoking thread for the specified number of seconds.*
- #define osalThreadSleepMilliseconds(msec) osalThreadSleep(OSAL\_MS2ST(msec))
 

*Delays the invoking thread for the specified number of milliseconds.*
- #define osalThreadSleepMicroseconds(usec) osalThreadSleep(OSAL\_US2ST(usec))
 

*Delays the invoking thread for the specified number of microseconds.*

## Typedefs

- typedef uint32\_t syssts\_t
 

*Type of a system status word.*
- typedef int32\_t msg\_t
 

*Type of a message.*
- typedef uint32\_t systime\_t
 

*Type of system time counter.*
- typedef uint32\_t rtcnt\_t
 

*Type of realtime counter.*
- typedef void \* thread\_reference\_t
 

*Type of a thread reference.*
- typedef struct event\_source event\_source\_t
 

*Type of an event flags object.*
- typedef void(\* eventcallback\_t) (event\_source\_t \*esp)
 

*Type of an event source callback.*
- typedef uint32\_t eventflags\_t
 

*Type of an event flags mask.*
- typedef uint32\_t mutex\_t
 

*Type of a mutex.*

## Functions

- void osallinit (void)
 

*OSAL module initialization.*
- void osalSysHalt (const char \*reason)
 

*System halt with error message.*
- void osalSysPolledDelayX (rtcnt\_t cycles)
 

*Polled delay.*
- void osalOsTimerHandlerl (void)
 

*System timer handler.*

- void `osalOsRescheduleS` (void)
 

*Checks if a reschedule is required and performs it.*
- `systime_t osalOsGetSystemTimeX` (void)
 

*Current system time.*
- void `osalThreadSleepS` (`systime_t` time)
 

*Suspends the invoking thread for the specified time.*
- void `osalThreadSleep` (`systime_t` time)
 

*Suspends the invoking thread for the specified time.*
- `msg_t osalThreadSuspendS` (`thread_reference_t` \*trp)
 

*Sends the current thread sleeping and sets a reference variable.*
- `msg_t osalThreadSuspendTimeoutS` (`thread_reference_t` \*trp, `systime_t` timeout)
 

*Sends the current thread sleeping and sets a reference variable.*
- void `osalThreadResumeI` (`thread_reference_t` \*trp, `msg_t` msg)
 

*Wakes up a thread waiting on a thread reference object.*
- void `osalThreadResumeS` (`thread_reference_t` \*trp, `msg_t` msg)
 

*Wakes up a thread waiting on a thread reference object.*
- `msg_t osalThreadEnqueueTimeoutS` (`threads_queue_t` \*tqp, `systime_t` timeout)
 

*Enqueues the caller thread.*
- void `osalThreadDequeueNextI` (`threads_queue_t` \*tqp, `msg_t` msg)
 

*Dequeues and wakes up one thread from the queue, if any.*
- void `osalThreadDequeueAllI` (`threads_queue_t` \*tqp, `msg_t` msg)
 

*Dequeues and wakes up all threads from the queue.*
- void `osalEventBroadcastFlagsI` (`event_source_t` \*esp, `eventflags_t` flags)
 

*Add flags to an event source object.*
- void `osalEventBroadcastFlags` (`event_source_t` \*esp, `eventflags_t` flags)
 

*Add flags to an event source object.*
- void `osalEventSetCallback` (`event_source_t` \*esp, `eventcallback_t` cb, void \*param)
 

*Event callback setup.*
- void `osalMutexLock` (`mutex_t` \*mp)
 

*Locks the specified mutex.*
- void `osalMutexUnlock` (`mutex_t` \*mp)
 

*Unlocks the specified mutex.*
- static void `osalSysDisable` (void)
 

*Disables interrupts globally.*
- static void `osalSysEnable` (void)
 

*Enables interrupts globally.*
- static void `osalSysLock` (void)
 

*Enters a critical zone from thread context.*
- static void `osalSysUnlock` (void)
 

*Leaves a critical zone from thread context.*
- static void `osalSysLockFromISR` (void)
 

*Enters a critical zone from ISR context.*
- static void `osalSysUnlockFromISR` (void)
 

*Leaves a critical zone from ISR context.*
- static `syssts_t osalSysGetStatusAndLockX` (void)
 

*Returns the execution status and enters a critical zone.*
- static void `osalSysRestoreStatusX` (`syssts_t` sts)
 

*Restores the specified execution status and leaves a critical zone.*
- static bool `osalOslsTimeWithinX` (`systime_t` time, `systime_t` start, `systime_t` end)
 

*Checks if the specified time is within the specified time window.*
- static void `osalThreadQueueObjectInit` (`threads_queue_t` \*tqp)

- static void **osalEventObjectInit** (**event\_source\_t** \*esp)  
*Initializes a threads queue object.*
- static void **osalMutexObjectInit** (**mutex\_t** \*mp)  
*Initializes a mutex\_t object.*

## Variables

- const char \* **osal\_halt\_msg**  
*Pointer to a halt error message.*

### 9.56.1 Detailed Description

OSAL module header.

## 9.57 pal.c File Reference

I/O Ports Abstraction Layer code.

```
#include "hal.h"
```

## Functions

- **ioportmask\_t palReadBus** (**IOBus** \*bus)  
*Read from an I/O bus.*
- void **palWriteBus** (**IOBus** \*bus, **ioportmask\_t** bits)  
*Write to an I/O bus.*
- void **palSetBusMode** (**IOBus** \*bus, **iomode\_t** mode)  
*Programs a bus with the specified mode.*

### 9.57.1 Detailed Description

I/O Ports Abstraction Layer code.

## 9.58 pal.h File Reference

I/O Ports Abstraction Layer macros, types and structures.

```
#include "pal_lld.h"
```

## Data Structures

- struct **IOBus**  
*I/O bus descriptor.*

## Macros

- `#define PAL_PORT_BIT(n) ((ioportmask_t)(1U << (n)))`  
*Port bit helper macro.*
- `#define PAL_GROUP_MASK(width) ((ioportmask_t)(1U << (width)) - 1U)`  
*Bits group mask helper.*
- `#define _IOBUS_DATA(name, port, width, offset) {port, PAL_GROUP_MASK(width), offset}`  
*Data part of a static I/O bus initializer.*
- `#define IOBUS_DECL(name, port, width, offset) IOBus name = _IOBUS_DATA(name, port, width, offset)`  
*Static I/O bus initializer.*

## Pads mode constants

- `#define PAL_MODE_RESET 0U`  
*After reset state.*
- `#define PAL_MODE_UNCONNECTED 1U`  
*Safe state for **unconnected** pads.*
- `#define PAL_MODE_INPUT 2U`  
*Regular input high-Z pad.*
- `#define PAL_MODE_INPUT_PULLUP 3U`  
*Input pad with weak pull up resistor.*
- `#define PAL_MODE_INPUT_PULLDOWN 4U`  
*Input pad with weak pull down resistor.*
- `#define PAL_MODE_INPUT_ANALOG 5U`  
*Analog input mode.*
- `#define PAL_MODE_OUTPUT_PUSH_PULL 6U`  
*Push-pull output pad.*
- `#define PAL_MODE_OUTPUT_OPENDRAIN 7U`  
*Open-drain output pad.*

## Logic level constants

- `#define PAL_LOW 0U`  
*Logical low state.*
- `#define PAL_HIGH 1U`  
*Logical high state.*

## Macro Functions

- `#define palInit(config) pal_lld_init(config)`  
*PAL subsystem initialization.*
- `#define palReadPort(port) ((void)(port), 0U)`  
*Reads the physical I/O port states.*
- `#define palReadLatch(port) ((void)(port), 0U)`  
*Reads the output latch.*
- `#define palWritePort(port, bits) ((void)(port), (void)(bits))`  
*Writes a bits mask on a I/O port.*
- `#define palSetPort(port, bits) palWritePort(port, palReadLatch(port) | (bits))`  
*Sets a bits mask on a I/O port.*
- `#define palClearPort(port, bits) palWritePort(port, palReadLatch(port) & ~ (bits))`  
*Clears a bits mask on a I/O port.*
- `#define palTogglePort(port, bits) palWritePort(port, palReadLatch(port) ^ (bits))`  
*Toggles a bits mask on a I/O port.*
- `#define palReadGroup(port, mask, offset) ((palReadPort(port) >> (offset)) & (mask))`  
*Reads a group of bits.*
- `#define palWriteGroup(port, mask, offset, bits)`  
*Writes a group of bits.*

- `#define palSetGroupMode(port, mask, offset, mode)`  
*Pads group mode setup.*
- `#define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1U)`  
*Reads an input pad logic state.*
- `#define palWritePad(port, pad, bit)`  
*Writes a logic state on an output pad.*
- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`  
*Sets a pad logic state to PAL\_HIGH.*
- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`  
*Clears a pad logic state to PAL\_LOW.*
- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`  
*Toggles a pad logic state.*
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)`  
*Pad mode setup.*
- `#define palReadLine(line) palReadPad(PAL_PORT(line), PAL_PAD(line))`  
*Reads an input line logic state.*
- `#define palWriteLine(line, bit) palWritePad(PAL_PORT(line), PAL_PAD(line), bit)`  
*Writes a logic state on an output line.*
- `#define palSetLine(line) palSetPad(PAL_PORT(line), PAL_PAD(line))`  
*Sets a line logic state to PAL\_HIGH.*
- `#define palClearLine(line) palClearPad(PAL_PORT(line), PAL_PAD(line))`  
*Clears a line logic state to PAL\_LOW.*
- `#define palToggleLine(line) palTogglePad(PAL_PORT(line), PAL_PAD(line))`  
*Toggles a line logic state.*
- `#define palSetLineMode(line, mode) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)`  
*Line mode setup.*

## Functions

- `ioportmask_t palReadBus (IOBus *bus)`  
*Read from an I/O bus.*
- `void palWriteBus (IOBus *bus, ioportmask_t bits)`  
*Write to an I/O bus.*
- `void palSetBusMode (IOBus *bus, iomode_t mode)`  
*Programs a bus with the specified mode.*

### 9.58.1 Detailed Description

I/O Ports Abstraction Layer macros, types and structures.

## 9.59 pal\_lld.c File Reference

PLATFORM PAL subsystem low level driver source.

```
#include "hal.h"
```

## Functions

- `void _pal_lld_init (const PALConfig *config)`  
*STM32 I/O ports configuration.*
- `void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)`  
*Pads mode setup.*

### 9.59.1 Detailed Description

PLATFORM PAL subsystem low level driver source.

## 9.60 pal\_lld.h File Reference

PLATFORM PAL subsystem low level driver header.

### Data Structures

- struct [PALConfig](#)  
*Generic I/O ports static initializer.*

### Macros

- #define [IOPORT1](#) 0  
*First I/O port identifier.*
- #define [pal\\_lld\\_init](#)(config) [\\_pal\\_lld\\_init](#)(config)  
*Low level PAL subsystem initialization.*
- #define [pal\\_lld\\_readport](#)(port) 0U  
*Reads the physical I/O port states.*
- #define [pal\\_lld\\_readlatch](#)(port) 0U  
*Reads the output latch.*
- #define [pal\\_lld\\_writeport](#)(port, bits)  
*Writes a bits mask on a I/O port.*
- #define [pal\\_lld\\_setport](#)(port, bits)  
*Sets a bits mask on a I/O port.*
- #define [pal\\_lld\\_clearport](#)(port, bits)  
*Clears a bits mask on a I/O port.*
- #define [pal\\_lld\\_toggleport](#)(port, bits)  
*Toggles a bits mask on a I/O port.*
- #define [pal\\_lld\\_readgroup](#)(port, mask, offset) 0U  
*Reads a group of bits.*
- #define [pal\\_lld\\_writegroup](#)(port, mask, offset, bits)  
*Writes a group of bits.*
- #define [pal\\_lld\\_setgroupmode](#)(port, mask, offset, mode) [\\_pal\\_lld\\_setgroupmode](#)(port, mask << offset, mode)  
*Pads group mode setup.*
- #define [pal\\_lld\\_readpad](#)(port, pad) [PAL\\_LOW](#)  
*Reads a logical state from an I/O pad.*
- #define [pal\\_lld\\_writepad](#)(port, pad, bit)  
*Writes a logical state on an output pad.*
- #define [pal\\_lld\\_setpad](#)(port, pad)  
*Sets a pad logical state to [PAL\\_HIGH](#).*
- #define [pal\\_lld\\_clearpad](#)(port, pad)  
*Clears a pad logical state to [PAL\\_LOW](#).*
- #define [pal\\_lld\\_togglepad](#)(port, pad)  
*Toggles a pad logical state.*
- #define [pal\\_lld\\_setpadmode](#)(port, pad, mode)

*Pad mode setup.*

### Port related definitions

- #define **PAL\_IOPORTS\_WIDTH** 16U  
*Width, in bits, of an I/O port.*
- #define **PAL\_WHOLE\_PORT** ((**ioportmask\_t**)0xFFFFU)  
*Whole port mask.*

### Line handling macros

- #define **PAL\_LINE**(port, pad) ((**ioline\_t**)((**uint32\_t**)(port)) | ((**uint32\_t**)(pad)))  
*Forms a line identifier.*
- #define **PAL\_PORT**(line) ((**stm32\_gpio\_t** \*)(((**uint32\_t**)(line)) & 0xFFFFFFFF0U))  
*Decodes a port identifier from a line identifier.*
- #define **PAL\_PAD**(line) ((**uint32\_t**)((**uint32\_t**)(line) & 0x0000000FU))  
*Decodes a pad identifier from a line identifier.*
- #define **PAL\_NOLINE** 0U  
*Value identifying an invalid line.*

### Typedefs

- typedef **uint32\_t** **ioportmask\_t**  
*Digital I/O port sized unsigned type.*
- typedef **uint32\_t** **iomode\_t**  
*Digital I/O modes.*
- typedef **uint32\_t** **ioline\_t**  
*Type of an I/O line.*
- typedef **uint32\_t** **ioportid\_t**  
*Port Identifier.*

### Functions

- void **\_pal\_lld\_init** (const **PALConfig** \*config)  
*STM32 I/O ports configuration.*
- void **\_pal\_lld\_setgroupmode** (**ioportid\_t** port, **ioportmask\_t** mask, **iomode\_t** mode)  
*Pads mode setup.*

#### 9.60.1 Detailed Description

PLATFORM PAL subsystem low level driver header.

## 9.61 pwm.c File Reference

PWM Driver code.

```
#include "hal.h"
```

## Functions

- void **pwmInit** (void)
 

*PWM Driver initialization.*
- void **pwmObjectInit** (**PWMDriver** \*pwmp)
 

*Initializes the standard part of a **PWMDriver** structure.*
- void **pwmStart** (**PWMDriver** \*pwmp, const **PWMConfig** \*config)
 

*Configures and activates the PWM peripheral.*
- void **pwmStop** (**PWMDriver** \*pwmp)
 

*Deactivates the PWM peripheral.*
- void **pwmChangePeriod** (**PWMDriver** \*pwmp, **pwmcnt\_t** period)
 

*Changes the period the PWM peripheral.*
- void **pwmEnableChannel** (**PWMDriver** \*pwmp, **pwmchannel\_t** channel, **pwmcnt\_t** width)
 

*Enables a PWM channel.*
- void **pwmDisableChannel** (**PWMDriver** \*pwmp, **pwmchannel\_t** channel)
 

*Disables a PWM channel and its notification.*
- void **pwmEnablePeriodicNotification** (**PWMDriver** \*pwmp)
 

*Enables the periodic activation edge notification.*
- void **pwmDisablePeriodicNotification** (**PWMDriver** \*pwmp)
 

*Disables the periodic activation edge notification.*
- void **pwmEnableChannelNotification** (**PWMDriver** \*pwmp, **pwmchannel\_t** channel)
 

*Enables a channel de-activation edge notification.*
- void **pwmDisableChannelNotification** (**PWMDriver** \*pwmp, **pwmchannel\_t** channel)
 

*Disables a channel de-activation edge notification.*

### 9.61.1 Detailed Description

PWM Driver code.

## 9.62 pwm.h File Reference

PWM Driver macros and structures.

```
#include "pwm_lld.h"
```

## Macros

### PWM output mode macros

- #define **PWM\_OUTPUT\_MASK** 0x0FU
 

*Standard output modes mask.*
- #define **PWM\_OUTPUT\_DISABLED** 0x00U
 

*Output not driven, callback only.*
- #define **PWM\_OUTPUT\_ACTIVE\_HIGH** 0x01U
 

*Positive PWM logic, active is logic level one.*
- #define **PWM\_OUTPUT\_ACTIVE\_LOW** 0x02U
 

*Inverse PWM logic, active is logic level zero.*

### PWM duty cycle conversion

- #define **PWM\_FRACTION\_TO\_WIDTH**(pwmp, denominator, numerator)

- Converts from fraction to pulse width.
- #define `PWM_DEGREES_TO_WIDTH`(pwmp, degrees) `PWM_FRACTION_TO_WIDTH`(pwmp, 36000, degrees)
  - Converts from degrees to pulse width.
- #define `PWM_PERCENTAGE_TO_WIDTH`(pwmp, percentage) `PWM_FRACTION_TO_WIDTH`(pwmp, 10000, percentage)
  - Converts from percentage to pulse width.

## Macro Functions

- #define `pwmChangePeriodI`(pwmp, value)
  - Changes the period the PWM peripheral.
- #define `pwmEnableChannelI`(pwmp, channel, width)
  - Enables a PWM channel.
- #define `pwmDisableChannelI`(pwmp, channel)
  - Disables a PWM channel.
- #define `pwmlsChannelEnabledI`(pwmp, channel) (((pwmp)->enabled & ((`pwmchnmsk_t`)1U << (`pwmchnmsk_t`)(channel))) != 0U)
  - Returns a PWM channel status.
- #define `pwmEnablePeriodicNotificationI`(pwmp) `pwm_lld_enable_periodic_notification`(pwmp)
  - Enables the periodic activation edge notification.
- #define `pwmDisablePeriodicNotificationI`(pwmp) `pwm_lld_disable_periodic_notification`(pwmp)
  - Disables the periodic activation edge notification.
- #define `pwmEnableChannelNotificationI`(pwmp, channel) `pwm_lld_enable_channel_notification`(pwmp, channel)
  - Enables a channel de-activation edge notification.
- #define `pwmDisableChannelNotificationI`(pwmp, channel) `pwm_lld_disable_channel_notification`(pwmp, channel)
  - Disables a channel de-activation edge notification.

## Typedefs

- typedef struct `PWMDriver` `PWMDriver`
  - Type of a structure representing a PWM driver.
- typedef void(\* `pwmcallback_t`) (`PWMDriver` \*pwmp)
  - Type of a PWM notification callback.

## Enumerations

- enum `pwmstate_t` { `PWM_UNINIT` = 0, `PWM_STOP` = 1, `PWM_READY` = 2 }
  - Driver state machine possible states.

## Functions

- void `pwmInit` (void)
  - PWM Driver initialization.
- void `pwmObjectInit` (`PWMDriver` \*pwmp)
  - Initializes the standard part of a `PWMDriver` structure.
- void `pwmStart` (`PWMDriver` \*pwmp, const `PWMConfig` \*config)
  - Configures and activates the PWM peripheral.
- void `pwmStop` (`PWMDriver` \*pwmp)
  - Deactivates the PWM peripheral.
- void `pwmChangePeriod` (`PWMDriver` \*pwmp, `pwmcnt_t` period)
  - Changes the period the PWM peripheral.

- void `pwmEnableChannel (PWMDriver *pwmp, pwmchannel_t channel, pwcnt_t width)`  
*Enables a PWM channel.*
- void `pwmDisableChannel (PWMDriver *pwmp, pwmchannel_t channel)`  
*Disables a PWM channel and its notification.*
- void `pwmEnablePeriodicNotification (PWMDriver *pwmp)`  
*Enables the periodic activation edge notification.*
- void `pwmDisablePeriodicNotification (PWMDriver *pwmp)`  
*Disables the periodic activation edge notification.*
- void `pwmEnableChannelNotification (PWMDriver *pwmp, pwmchannel_t channel)`  
*Enables a channel de-activation edge notification.*
- void `pwmDisableChannelNotification (PWMDriver *pwmp, pwmchannel_t channel)`  
*Disables a channel de-activation edge notification.*

### 9.62.1 Detailed Description

PWM Driver macros and structures.

## 9.63 pwm\_lld.c File Reference

PLATFORM PWM subsystem low level driver source.

```
#include "hal.h"
```

### Functions

- void `pwm_lld_init (void)`  
*Low level PWM driver initialization.*
- void `pwm_lld_start (PWMDriver *pwmp)`  
*Configures and activates the PWM peripheral.*
- void `pwm_lld_stop (PWMDriver *pwmp)`  
*Deactivates the PWM peripheral.*
- void `pwm_lld_enable_channel (PWMDriver *pwmp, pwmchannel_t channel, pwcnt_t width)`  
*Enables a PWM channel.*
- void `pwm_lld_disable_channel (PWMDriver *pwmp, pwmchannel_t channel)`  
*Disables a PWM channel and its notification.*
- void `pwm_lld_enable_periodic_notification (PWMDriver *pwmp)`  
*Enables the periodic activation edge notification.*
- void `pwm_lld_disable_periodic_notification (PWMDriver *pwmp)`  
*Disables the periodic activation edge notification.*
- void `pwm_lld_enable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)`  
*Enables a channel de-activation edge notification.*
- void `pwm_lld_disable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)`  
*Disables a channel de-activation edge notification.*

### Variables

- PWMDriver PWMD1  
*PWMD1 driver identifier.*

### 9.63.1 Detailed Description

PLATFORM PWM subsystem low level driver source.

## 9.64 pwm\_lld.h File Reference

PLATFORM PWM subsystem low level driver header.

### Data Structures

- struct [PWMChannelConfig](#)  
*Type of a PWM driver channel configuration structure.*
- struct [PWMConfig](#)  
*Type of a PWM driver configuration structure.*
- struct [PWMDriver](#)  
*Structure representing a PWM driver.*

### Macros

- #define [PWM\\_CHANNELS](#) 4  
*Number of PWM channels per PWM driver.*
- #define [pwm\\_lld\\_change\\_period](#)(pwmp, period)  
*Changes the period the PWM peripheral.*

### PLATFORM configuration options

- #define [PLATFORM\\_PWM\\_USE\\_PWM1](#) FALSE  
*PWMD1 driver enable switch.*

### Typedefs

- typedef uint32\_t [pwmemode\\_t](#)  
*Type of a PWM mode.*
- typedef uint8\_t [pwmchannel\\_t](#)  
*Type of a PWM channel.*
- typedef uint32\_t [pwmchnmsk\\_t](#)  
*Type of a channels mask.*
- typedef uint32\_t [pwmcnt\\_t](#)  
*Type of a PWM counter.*

### Functions

- void [pwm\\_lld\\_init](#) (void)  
*Low level PWM driver initialization.*
- void [pwm\\_lld\\_start](#) (PWMDriver \*pwmp)  
*Configures and activates the PWM peripheral.*
- void [pwm\\_lld\\_stop](#) (PWMDriver \*pwmp)  
*Deactivates the PWM peripheral.*
- void [pwm\\_lld\\_enable\\_channel](#) (PWMDriver \*pwmp, pwmchannel\_t channel, pwmcnt\_t width)

- void `pwm_lll_disable_channel (PWMDriver *pwmp, pwmchannel_t channel)`

*Disables a PWM channel and its notification.*
- void `pwm_lll_enable_periodic_notification (PWMDriver *pwmp)`

*Enables the periodic activation edge notification.*
- void `pwm_lll_disable_periodic_notification (PWMDriver *pwmp)`

*Disables the periodic activation edge notification.*
- void `pwm_lll_enable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)`

*Enables a channel de-activation edge notification.*
- void `pwm_lll_disable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)`

*Disables a channel de-activation edge notification.*

### 9.64.1 Detailed Description

PLATFORM PWM subsystem low level driver header.

## 9.65 rtc.c File Reference

RTC Driver code.

```
#include "hal.h"
```

### Functions

- void `rtcInit (void)`

*RTC Driver initialization.*
- void `rtcObjectInit (RTCDriver *rtcp)`

*Initializes a generic RTC driver object.*
- void `rtcSetTime (RTCDriver *rtcp, const RTCDateTime *timespec)`

*Set current time.*
- void `rtcGetTime (RTCDriver *rtcp, RTCDateTime *timespec)`

*Get current time.*
- void `rtcSetAlarm (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAlarm *alarmspec)`

*Set alarm time.*
- void `rtcGetAlarm (RTCDriver *rtcp, rtcalarm_t alarm, RTCAlarm *alarmspec)`

*Get current alarm.*
- void `rtcSetCallback (RTCDriver *rtcp, rtccb_t callback)`

*Enables or disables RTC callbacks.*
- void `rtcConvertDateTimeToStructTm (const RTCDateTime *timespec, struct tm *timp, uint32_t *tv_msec)`

*Convert `RTCDateTime` to broken-down time structure.*
- void `rtcConvertStructTmToDateTm (const struct tm *timp, uint32_t tv_msec, RTCDateTime *timespec)`

*Convert broken-down time structure to `RTCDateTime`.*
- uint32\_t `rtcConvertDateTimeToFAT (const RTCDateTime *timespec)`

*Get current time in format suitable for usage in FAT file system.*

### 9.65.1 Detailed Description

RTC Driver code.

## 9.66 rtc.h File Reference

RTC Driver macros and structures.

```
#include <time.h>
#include "rtc_lld.h"
```

### Data Structures

- struct **RTCDateTime**  
*Type of a structure representing an RTC date/time stamp.*

### Macros

- #define **RTC\_BASE\_YEAR** 1980U  
*Base year of the calendar.*

#### Date/Time bit masks for FAT format

- #define **RTC\_FAT\_TIME\_SECONDS\_MASK** 0x00000001FU
- #define **RTC\_FAT\_TIME\_MINUTES\_MASK** 0x000007E0U
- #define **RTC\_FAT\_TIME\_HOURS\_MASK** 0x0000F800U
- #define **RTC\_FAT\_DATE\_DAYS\_MASK** 0x001F0000U
- #define **RTC\_FAT\_DATE\_MONTHS\_MASK** 0x01E00000U
- #define **RTC\_FAT\_DATE\_YEARS\_MASK** 0xFE000000U

#### Day of week encoding

- #define **RTC\_DAY\_CATURDAY** 0U
- #define **RTC\_DAY\_MONDAY** 1U
- #define **RTC\_DAY\_TUESDAY** 2U
- #define **RTC\_DAY\_WEDNESDAY** 3U
- #define **RTC\_DAY\_THURSDAY** 4U
- #define **RTC\_DAY\_FRIDAY** 5U
- #define **RTC\_DAY\_SATURDAY** 6U
- #define **RTC\_DAY\_SUNDAY** 7U

### Typedefs

- typedef struct **RTCDriver** **RTCDriver**  
*Type of a structure representing an RTC driver.*

### Functions

- void **rtcInit** (void)  
*RTC Driver initialization.*
- void **rtcObjectInit** (**RTCDriver** \*rtcp)  
*Initializes a generic RTC driver object.*
- void **rtcSetTime** (**RTCDriver** \*rtcp, const **RTCDateTime** \*timespec)  
*Set current time.*
- void **rtcGetTime** (**RTCDriver** \*rtcp, **RTCDateTime** \*timespec)  
*Get current time.*
- void **rtcSetCallback** (**RTCDriver** \*rtcp, **rtccb\_t** callback)

*Enables or disables RTC callbacks.*

- void `rtcConvertDateTimeToStructTm` (const `RTCDateTime` \*timespec, struct tm \*timp, uint32\_t \*tv\_msec)  
*Convert `RTCDateTime` to broken-down time structure.*
- void `rtcConvertStructTmToDateTm` (const struct tm \*timp, uint32\_t tv\_msec, `RTCDateTime` \*timespec)  
*Convert broken-down time structure to `RTCDateTime`.*
- uint32\_t `rtcConvertDateTimeToFAT` (const `RTCDateTime` \*timespec)  
*Get current time in format suitable for usage in FAT file system.*

### 9.66.1 Detailed Description

RTC Driver macros and structures.

## 9.67 rtc\_lld.c File Reference

PLATFORM RTC subsystem low level driver source.

```
#include "hal.h"
```

### Functions

- void `rtc_lld_init` (void)  
*Enable access to registers.*
- void `rtc_lld_set_time` (`RTCDriver` \*rtcp, const `RTCDateTime` \*timespec)  
*Set current time.*
- void `rtc_lld_get_time` (`RTCDriver` \*rtcp, `RTCDateTime` \*timespec)  
*Get current time.*
- void `rtc_lld_set_alarm` (`RTCDriver` \*rtcp, `rtcalarm_t` alarm, const `RTCAlarm` \*alarmspec)  
*Set alarm time.*
- void `rtc_lld_get_alarm` (`RTCDriver` \*rtcp, `rtcalarm_t` alarm, `RTCAlarm` \*alarmspec)  
*Get alarm time.*

### Variables

- `RTCDriver RTCD1`  
*RTC driver identifier.*

### 9.67.1 Detailed Description

PLATFORM RTC subsystem low level driver source.

## 9.68 rtc\_lld.h File Reference

PLATFORM RTC subsystem low level driver header.

## Data Structures

- struct **RTCAlarm**  
*Type of a structure representing an RTC alarm time stamp.*
- struct **RTCDriverVMT**  
*RTCDriver virtual methods table.*
- struct **RTCDriver**  
*Structure representing an RTC driver.*

## Macros

- #define **\_rtc\_driver\_methods\_file\_stream\_methods**  
*FileStream specific methods.*

## Implementation capabilities

- #define **RTC\_SUPPORTS\_CALLBACKS** TRUE  
*Callback support int the driver.*
- #define **RTC\_ALARMS** 2  
*Number of alarms available.*
- #define **RTC\_HAS\_STORAGE** FALSE  
*Presence of a local persistent storage.*

## PLATFORM configuration options

- #define **PLATFORM\_RTC\_USE\_RTC1** FALSE  
*RTCD1 driver enable switch.*

## Typedefs

- typedef uint32\_t **rtcalarm\_t**  
*Type of an RTC alarm number.*
- typedef void(\* **rtccb\_t**) (**RTCDriver** \*rtcp, **rtcevent\_t** event)  
*Type of a generic RTC callback.*

## Enumerations

- enum **rtcevent\_t**  
*Type of an RTC event.*

## Functions

- void **rtc\_ll\_init** (void)  
*Enable access to registers.*
- void **rtc\_ll\_set\_time** (**RTCDriver** \*rtcp, const **RTCDateTime** \*timespec)  
*Set current time.*
- void **rtc\_ll\_get\_time** (**RTCDriver** \*rtcp, **RTCDateTime** \*timespec)  
*Get current time.*
- void **rtc\_ll\_set\_alarm** (**RTCDriver** \*rtcp, **rtcalarm\_t** alarm, const **RTCAlarm** \*alarmspec)  
*Set alarm time.*
- void **rtc\_ll\_get\_alarm** (**RTCDriver** \*rtcp, **rtcalarm\_t** alarm, **RTCAlarm** \*alarmspec)  
*Get alarm time.*

### 9.68.1 Detailed Description

PLATFORM RTC subsystem low level driver header.

## 9.69 sdc.c File Reference

SDC Driver code.

```
#include <string.h>
#include "hal.h"
```

### Enumerations

- enum `mmc_switch_t`  
*MMC switch mode.*
- enum `sd_switch_t`  
*SDC switch mode.*
- enum `sd_switch_function_t`  
*SDC switch function.*

### Functions

- static bool `mode_detect (SDCDriver *sdcp)`  
*Detects card mode.*
- static bool `mmc_init (SDCDriver *sdcp)`  
*Init procedure for MMC.*
- static bool `sdc_init (SDCDriver *sdcp)`  
*Init procedure for SDC.*
- static uint32\_t `mmc_cmd6_construct (mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set)`  
*Constructs CMD6 argument for MMC.*
- static uint32\_t `sdc_cmd6_construct (sd_switch_t mode, sd_switch_function_t function, uint32_t value)`  
*Constructs CMD6 argument for SDC.*
- static uint16\_t `sdc_cmd6_extract_info (sd_switch_function_t function, const uint8_t *buf)`  
*Extracts information from CMD6 answer.*
- static bool `sdc_cmd6_check_status (sd_switch_function_t function, const uint8_t *buf)`  
*Checks status after switching using CMD6.*
- static bool `sdc_detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)`  
*Reads supported bus clock and switch SDC to appropriate mode.*
- static bool `mmc_detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)`  
*Reads supported bus clock and switch MMC to appropriate mode.*
- static bool `detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)`  
*Reads supported bus clock and switch card to appropriate mode.*
- static bool `sdc_set_bus_width (SDCDriver *sdcp)`  
*Sets bus width for SDC.*
- static bool `mmc_set_bus_width (SDCDriver *sdcp)`  
*Sets bus width for MMC.*
- bool `_sdc_wait_for_transfer_state (SDCDriver *sdcp)`  
*Wait for the card to complete pending operations.*
- void `sdclinit (void)`

- **sdcObjectInit (SDCDriver \*sdcp)**  
*Initializes the standard part of a `SDCDriver` structure.*
- **sdcStart (SDCDriver \*sdcp, const SDCCConfig \*config)**  
*Configures and activates the SDC peripheral.*
- **sdcStop (SDCDriver \*sdcp)**  
*Deactivates the SDC peripheral.*
- **bool sdcConnect (SDCDriver \*sdcp)**  
*Performs the initialization procedure on the inserted card.*
- **bool sdcDisconnect (SDCDriver \*sdcp)**  
*Brings the driver in a state safe for card removal.*
- **bool sdcRead (SDCDriver \*sdcp, uint32\_t startblk, uint8\_t \*buf, uint32\_t n)**  
*Reads one or more blocks.*
- **bool sdcWrite (SDCDriver \*sdcp, uint32\_t startblk, const uint8\_t \*buf, uint32\_t n)**  
*Writes one or more blocks.*
- **sdcflags\_t sdcGetAndClearErrors (SDCDriver \*sdcp)**  
*Returns the errors mask associated to the previous operation.*
- **bool sdcSync (SDCDriver \*sdcp)**  
*Waits for card idle condition.*
- **bool sdcGetInfo (SDCDriver \*sdcp, BlockDeviceInfo \*bdip)**  
*Returns the media info.*
- **bool sdcErase (SDCDriver \*sdcp, uint32\_t startblk, uint32\_t endblk)**  
*Erases the supplied blocks.*

## Variables

- **static const struct SDCDriverVMT sdc\_vmt**  
*Virtual methods table.*

### 9.69.1 Detailed Description

SDC Driver code.

## 9.70 sdc.h File Reference

SDC Driver macros and structures.

```
#include "sdc_lld.h"
```

## Macros

### SD card types

- #define **SDC\_MODE\_CARDTYPE\_MASK** 0xFU
- #define **SDC\_MODE\_CARDTYPE\_SDV11** 0U
- #define **SDC\_MODE\_CARDTYPE\_SDV20** 1U
- #define **SDC\_MODE\_CARDTYPE\_MMC** 2U
- #define **SDC\_MODE\_HIGH\_CAPACITY** 0x10U

### SDC bus error conditions

- #define **SDC\_NO\_ERROR** 0U
- #define **SDC\_CMD\_CRC\_ERROR** 1U
- #define **SDC\_DATA\_CRC\_ERROR** 2U
- #define **SDC\_DATA\_TIMEOUT** 4U
- #define **SDC\_COMMAND\_TIMEOUT** 8U
- #define **SDC\_TX\_UNDERRUN** 16U
- #define **SDC\_RX\_OVERRUN** 32U
- #define **SDC\_STARTBIT\_ERROR** 64U
- #define **SDC\_OVERFLOW\_ERROR** 128U
- #define **SDC\_UNHANDLED\_ERROR** 0xFFFFFFFFU

### SDC configuration options

- #define **SDC\_INIT\_RETRY** 100  
*Number of initialization attempts before rejecting the card.*
- #define **SDC\_MMC\_SUPPORT** FALSE  
*Include support for MMC cards.*
- #define **SDC\_NICE\_WAITING** TRUE  
*Delays insertions.*

### Macro Functions

- #define **sdclsCardInserted**(sdcp) (**sdc\_lld\_is\_card\_inserted**(sdcp))  
*Returns the card insertion status.*
- #define **sdclsWriteProtected**(sdcp) (**sdc\_lld\_is\_write\_protected**(sdcp))  
*Returns the write protect status.*

### Enumerations

- enum **sdcbusmode\_t**  
*Type of SDIO bus mode.*
- enum **sdcbusclk\_t**  
*Max supported clock.*

### Functions

- void **sdcInit** (void)  
*SDC Driver initialization.*
- void **sdcObjectInit** (**SDCDriver** \*sdcp)  
*Initializes the standard part of a **SDCDriver** structure.*
- void **sdcStart** (**SDCDriver** \*sdcp, const **SDCConfig** \*config)  
*Configures and activates the SDC peripheral.*
- void **sdcStop** (**SDCDriver** \*sdcp)  
*Deactivates the SDC peripheral.*
- bool **sdcConnect** (**SDCDriver** \*sdcp)  
*Performs the initialization procedure on the inserted card.*
- bool **sdcDisconnect** (**SDCDriver** \*sdcp)  
*Brings the driver in a state safe for card removal.*
- bool **sdcRead** (**SDCDriver** \*sdcp, uint32\_t startblk, uint8\_t \*buf, uint32\_t n)  
*Reads one or more blocks.*
- bool **sdcWrite** (**SDCDriver** \*sdcp, uint32\_t startblk, const uint8\_t \*buf, uint32\_t n)  
*Writes one or more blocks.*
- **sdcflags\_t** **sdcGetAndClearErrors** (**SDCDriver** \*sdcp)  
*Returns the errors mask associated to the previous operation.*

- `bool sdcSync (SDCDriver *sdcp)`  
*Waits for card idle condition.*
- `bool sdcGetInfo (SDCDriver *sdcp, BlockDeviceInfo *bdip)`  
*Returns the media info.*
- `bool sdcErase (SDCDriver *sdcp, uint32_t startblk, uint32_t endblk)`  
*Erases the supplied blocks.*
- `bool _sdc_wait_for_transfer_state (SDCDriver *sdcp)`  
*Wait for the card to complete pending operations.*

### 9.70.1 Detailed Description

SDC Driver macros and structures.

## 9.71 sdc\_lld.c File Reference

PLATFORM SDC subsystem low level driver source.

```
#include "hal.h"
```

### Functions

- `void sdc_lld_init (void)`  
*Low level SDC driver initialization.*
- `void sdc_lld_start (SDCDriver *sdcp)`  
*Configures and activates the SDC peripheral.*
- `void sdc_lld_stop (SDCDriver *sdcp)`  
*Deactivates the SDC peripheral.*
- `void sdc_lld_start_clk (SDCDriver *sdcp)`  
*Starts the SDIO clock and sets it to init mode (400kHz or less).*
- `void sdc_lld_set_data_clk (SDCDriver *sdcp, sdcbusclk_t clk)`  
*Sets the SDIO clock to data mode (25MHz or less).*
- `void sdc_lld_stop_clk (SDCDriver *sdcp)`  
*Stops the SDIO clock.*
- `void sdc_lld_set_bus_mode (SDCDriver *sdcp, sdcbusmode_t mode)`  
*Switches the bus to 4 bits mode.*
- `void sdc_lld_send_cmd_none (SDCDriver *sdcp, uint8_t cmd, uint32_t arg)`  
*Sends an SDIO command with no response expected.*
- `bool sdc_lld_send_cmd_short (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`  
*Sends an SDIO command with a short response expected.*
- `bool sdc_lld_send_cmd_short_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`  
*Sends an SDIO command with a short response expected and CRC.*
- `bool sdc_lld_send_cmd_long_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`  
*Sends an SDIO command with a long response expected and CRC.*
- `bool sdc_lld_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`  
*Reads one or more blocks.*
- `bool sdc_lld_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`  
*Writes one or more blocks.*
- `bool sdc_lld_sync (SDCDriver *sdcp)`  
*Waits for card idle condition.*

## Variables

- **SDCDriver SDCD1**  
*SDCD1 driver identifier.*

### 9.71.1 Detailed Description

PLATFORM SDC subsystem low level driver source.

## 9.72 sdc\_lld.h File Reference

PLATFORM SDC subsystem low level driver header.

### Data Structures

- struct **SDCConfig**  
*Driver configuration structure.*
- struct **SDCDriverVMT**  
*SDCDriver virtual methods table.*
- struct **SDCDriver**  
*Structure representing an SDC driver.*

### Macros

- #define **\_sdc\_driver\_methods \_mmcsd\_block\_device\_methods**  
*SDCDriver specific methods.*

### PLATFORM configuration options

- #define **PLATFORM\_SDC\_USE\_SDC1** FALSE  
*PWMD1 driver enable switch.*

### Typedefs

- typedef uint32\_t **sdcmode\_t**  
*Type of card flags.*
- typedef uint32\_t **sdcflags\_t**  
*SDC Driver condition flags type.*
- typedef struct **SDCDriver SDCDriver**  
*Type of a structure representing an SDC driver.*

### Functions

- void **sdc\_lld\_init** (void)  
*Low level SDC driver initialization.*
- void **sdc\_lld\_start** (**SDCDriver** \*sdcp)  
*Configures and activates the SDC peripheral.*
- void **sdc\_lld\_stop** (**SDCDriver** \*sdcp)  
*Deactivates the SDC peripheral.*

- void **sdc\_lld\_start\_clk** (**SDCDriver** \*sdcp)  
*Starts the SDIO clock and sets it to init mode (400kHz or less).*
- void **sdc\_lld\_set\_data\_clk** (**SDCDriver** \*sdcp, **sdcbusclk\_t** clk)  
*Sets the SDIO clock to data mode (25MHz or less).*
- void **sdc\_lld\_stop\_clk** (**SDCDriver** \*sdcp)  
*Stops the SDIO clock.*
- void **sdc\_lld\_set\_bus\_mode** (**SDCDriver** \*sdcp, **sdcbusmode\_t** mode)  
*Switches the bus to 4 bits mode.*
- void **sdc\_lld\_send\_cmd\_none** (**SDCDriver** \*sdcp, **uint8\_t** cmd, **uint32\_t** arg)  
*Sends an SDIO command with no response expected.*
- bool **sdc\_lld\_send\_cmd\_short** (**SDCDriver** \*sdcp, **uint8\_t** cmd, **uint32\_t** arg, **uint32\_t** \*resp)  
*Sends an SDIO command with a short response expected.*
- bool **sdc\_lld\_send\_cmd\_short\_crc** (**SDCDriver** \*sdcp, **uint8\_t** cmd, **uint32\_t** arg, **uint32\_t** \*resp)  
*Sends an SDIO command with a short response expected and CRC.*
- bool **sdc\_lld\_send\_cmd\_long\_crc** (**SDCDriver** \*sdcp, **uint8\_t** cmd, **uint32\_t** arg, **uint32\_t** \*resp)  
*Sends an SDIO command with a long response expected and CRC.*
- bool **sdc\_lld\_read** (**SDCDriver** \*sdcp, **uint32\_t** startblk, **uint8\_t** \*buf, **uint32\_t** n)  
*Reads one or more blocks.*
- bool **sdc\_lld\_write** (**SDCDriver** \*sdcp, **uint32\_t** startblk, **const uint8\_t** \*buf, **uint32\_t** n)  
*Writes one or more blocks.*
- bool **sdc\_lld\_sync** (**SDCDriver** \*sdcp)  
*Waits for card idle condition.*

### 9.72.1 Detailed Description

PLATFORM SDC subsystem low level driver header.

## 9.73 serial.c File Reference

Serial Driver code.

```
#include "hal.h"
```

### Functions

- void **sdInit** (void)  
*Serial Driver initialization.*
- void **sdObjectInit** (**SerialDriver** \*sdp, **qnotify\_t** inotify, **qnotify\_t** onotify)  
*Initializes a generic full duplex driver object.*
- void **sdStart** (**SerialDriver** \*sdp, **const SerialConfig** \*config)  
*Configures and starts the driver.*
- void **sdStop** (**SerialDriver** \*sdp)  
*Stops the driver.*
- void **sdIncomingData** (**SerialDriver** \*sdp, **uint8\_t** b)  
*Handles incoming data.*
- **msg\_t** **sdRequestData** (**SerialDriver** \*sdp)  
*Handles outgoing data.*
- bool **sdPutWouldBlock** (**SerialDriver** \*sdp)  
*Direct output check on a **SerialDriver**.*
- bool **sdGetWouldBlock** (**SerialDriver** \*sdp)  
*Direct input check on a **SerialDriver**.*

### 9.73.1 Detailed Description

Serial Driver code.

## 9.74 serial.h File Reference

Serial Driver macros and structures.

```
#include "serial_lld.h"
```

### Data Structures

- struct **SerialDriverVMT**  
*SerialDriver virtual methods table.*
- struct **SerialDriver**  
*Full duplex serial driver class.*

### Macros

- #define **\_serial\_driver\_methods** **\_base\_asynchronous\_channel\_methods**  
*SerialDriver specific methods.*

### Serial status flags

- #define **SD\_PARITY\_ERROR** (**eventflags\_t**)32  
*Parity.*
- #define **SD\_FRAMING\_ERROR** (**eventflags\_t**)64  
*Framing.*
- #define **SD\_OVERRUN\_ERROR** (**eventflags\_t**)128  
*Overflow.*
- #define **SD\_NOISE\_ERROR** (**eventflags\_t**)256  
*Line noise.*
- #define **SD\_BREAK\_DETECTED** (**eventflags\_t**)512  
*LIN Break.*

### Serial configuration options

- #define **SERIAL\_DEFAULT\_BITRATE** 38400  
*Default bit rate.*
- #define **SERIAL\_BUFFERS\_SIZE** 16  
*Serial buffers size.*

### Macro Functions

- #define **sdPut**(**sdp**, **b**) **oqPut**(&(**sdp**)->**oqueue**, **b**)  
*Direct write to a SerialDriver.*
- #define **sdPutTimeout**(**sdp**, **b**, **t**) **oqPutTimeout**(&(**sdp**)->**oqueue**, **b**, **t**)  
*Direct write to a SerialDriver with timeout specification.*
- #define **sdGet**(**sdp**) **iqGet**(&(**sdp**)->**iqueue**)  
*Direct read from a SerialDriver.*
- #define **sdGetTimeout**(**sdp**, **t**) **iqGetTimeout**(&(**sdp**)->**iqueue**, **t**)  
*Direct read from a SerialDriver with timeout specification.*
- #define **sdWrite**(**sdp**, **b**, **n**) **oqWriteTimeout**(&(**sdp**)->**oqueue**, **b**, **n**, **TIME\_INFINITE**)  
*Direct blocking write to a SerialDriver.*

- #define `sdWriteTimeout`(sdp, b, n, t) `oqWriteTimeout`(&(sdp)->oqueue, b, n, t)  
*Direct blocking write to a `SerialDriver` with timeout specification.*
- #define `sdAsynchronousWrite`(sdp, b, n) `oqWriteTimeout`(&(sdp)->oqueue, b, n, TIME\_IMMEDIATE)  
*Direct non-blocking write to a `SerialDriver`.*
- #define `sdRead`(sdp, b, n) `iqReadTimeout`(&(sdp)->iqueue, b, n, TIME\_INFINITE)  
*Direct blocking read from a `SerialDriver`.*
- #define `sdReadTimeout`(sdp, b, n, t) `iqReadTimeout`(&(sdp)->iqueue, b, n, t)  
*Direct blocking read from a `SerialDriver` with timeout specification.*
- #define `sdAsynchronousRead`(sdp, b, n) `iqReadTimeout`(&(sdp)->iqueue, b, n, TIME\_IMMEDIATE)  
*Direct non-blocking read from a `SerialDriver`.*

## Typedefs

- typedef struct `SerialDriver` `SerialDriver`  
*Structure representing a serial driver.*

## Enumerations

- enum `sdstate_t` { `SD_UNINIT` = 0, `SD_STOP` = 1, `SD_READY` = 2 }  
*Driver state machine possible states.*

## Functions

- void `sdInit` (void)  
*Serial Driver initialization.*
- void `sdObjectInit` (`SerialDriver` \*sdp, `qnotify_t` inotify, `qnotify_t` onotify)  
*Initializes a generic full duplex driver object.*
- void `sdStart` (`SerialDriver` \*sdp, const `SerialConfig` \*config)  
*Configures and starts the driver.*
- void `sdStop` (`SerialDriver` \*sdp)  
*Stops the driver.*
- void `sdIncomingData` (`SerialDriver` \*sdp, `uint8_t` b)  
*Handles incoming data.*
- `msg_t` `sdRequestData` (`SerialDriver` \*sdp)  
*Handles outgoing data.*
- bool `sdPutWouldBlock` (`SerialDriver` \*sdp)  
*Direct output check on a `SerialDriver`.*
- bool `sdGetWouldBlock` (`SerialDriver` \*sdp)  
*Direct input check on a `SerialDriver`.*

### 9.74.1 Detailed Description

Serial Driver macros and structures.

## 9.75 serial\_lld.c File Reference

PLATFORM serial subsystem low level driver source.

```
#include "hal.h"
```

## Functions

- void `sd_lld_init` (void)  
*Low level serial driver initialization.*
- void `sd_lld_start` (`SerialDriver` \*`sdp`, const `SerialConfig` \*`config`)  
*Low level serial driver configuration and (re)start.*
- void `sd_lld_stop` (`SerialDriver` \*`sdp`)  
*Low level serial driver stop.*

## Variables

- `SerialDriver SD1`  
*USART1 serial driver identifier.*
- static const `SerialConfig default_config`  
*Driver default configuration.*

### 9.75.1 Detailed Description

PLATFORM serial subsystem low level driver source.

## 9.76 serial\_lld.h File Reference

PLATFORM serial subsystem low level driver header.

### Data Structures

- struct `SerialConfig`  
*PLATFORM Serial Driver configuration structure.*

### Macros

- #define `_serial_driver_data`  
*SerialDriver specific data.*

### PLATFORM configuration options

- #define `PLATFORM_SERIAL_USE_USART1` FALSE  
*USART1 driver enable switch.*

## Functions

- void `sd_lld_init` (void)  
*Low level serial driver initialization.*
- void `sd_lld_start` (`SerialDriver` \*`sdp`, const `SerialConfig` \*`config`)  
*Low level serial driver configuration and (re)start.*
- void `sd_lld_stop` (`SerialDriver` \*`sdp`)  
*Low level serial driver stop.*

### 9.76.1 Detailed Description

PLATFORM serial subsystem low level driver header.

## 9.77 serial\_usb.c File Reference

Serial over USB Driver code.

```
#include "hal.h"
```

### Functions

- static void **ibnotify** (**io\_buffers\_queue\_t** \*bqp)  
*Notification of empty buffer released into the input buffers queue.*
- static void **obnotify** (**io\_buffers\_queue\_t** \*bqp)  
*Notification of filled buffer inserted into the output buffers queue.*
- void **sduInit** (void)  
*Serial Driver initialization.*
- void **sduObjectInit** (**SerialUSBDriver** \*sdup)  
*Initializes a generic full duplex driver object.*
- void **sduStart** (**SerialUSBDriver** \*sdup, const **SerialUSBConfig** \*config)  
*Configures and starts the driver.*
- void **sduStop** (**SerialUSBDriver** \*sdup)  
*Stops the driver.*
- void **sduDisconnect** (**SerialUSBDriver** \*sdup)  
*USB device disconnection handler.*
- void **sduConfigureHook** (**SerialUSBDriver** \*sdup)  
*USB device configured handler.*
- bool **sduRequestsHook** (**USBDriver** \*usbp)  
*Default requests hook.*
- void **sduSOFHook** (**SerialUSBDriver** \*sdup)  
*SOF handler.*
- void **sduDataTransmitted** (**USBDriver** \*usbp, **usbep\_t** ep)  
*Default data transmitted callback.*
- void **sduDataReceived** (**USBDriver** \*usbp, **usbep\_t** ep)  
*Default data received callback.*
- void **sduInterruptTransmitted** (**USBDriver** \*usbp, **usbep\_t** ep)  
*Default data received callback.*

### 9.77.1 Detailed Description

Serial over USB Driver code.

## 9.78 serial\_usb.h File Reference

Serial over USB Driver macros and structures.

```
#include "usb_cdc.h"
```

## Data Structures

- struct **SerialUSBConfig**  
*Serial over USB Driver configuration structure.*
- struct **SerialUSBDriverVMT**  
*SerialDriver virtual methods table.*
- struct **SerialUSBDriver**  
*Full duplex serial driver class.*

## Macros

- #define **\_serial\_usb\_driver\_data**  
*SerialDriver specific data.*
- #define **\_serial\_usb\_driver\_methods \_base\_asynchronous\_channel\_methods**  
*SerialUSBDriver specific methods.*

## SERIAL\_USB configuration options

- #define **SERIAL\_USB\_BUFFERS\_SIZE** 256  
*Serial over USB buffers size.*
- #define **SERIAL\_USB\_BUFFERS\_NUMBER** 2  
*Serial over USB number of buffers.*

## Typedefs

- typedef struct **SerialUSBDriver** **SerialUSBDriver**  
*Structure representing a serial over USB driver.*

## Enumerations

- enum **sdustate\_t** { **SDU\_UNINIT** = 0, **SDU\_STOP** = 1, **SDU\_READY** = 2 }  
*Driver state machine possible states.*

## Functions

- void **sduInit** (void)  
*Serial Driver initialization.*
- void **sduObjectInit** (**SerialUSBDriver** \*sdup)  
*Initializes a generic full duplex driver object.*
- void **sduStart** (**SerialUSBDriver** \*sdup, const **SerialUSBConfig** \*config)  
*Configures and starts the driver.*
- void **sduStop** (**SerialUSBDriver** \*sdup)  
*Stops the driver.*
- void **sduDisconnectl** (**SerialUSBDriver** \*sdup)  
*USB device disconnection handler.*
- void **sduConfigureHookl** (**SerialUSBDriver** \*sdup)  
*USB device configured handler.*
- bool **sduRequestsHook** (**USBDriver** \*usbp)  
*Default requests hook.*
- void **sduSOFHookl** (**SerialUSBDriver** \*sdup)  
*SOF handler.*

- void **sduDataTransmitted** (USBDriver \*usbp, usbep\_t ep)  
*Default data transmitted callback.*
- void **sduDataReceived** (USBDriver \*usbp, usbep\_t ep)  
*Default data received callback.*
- void **sduInterruptTransmitted** (USBDriver \*usbp, usbep\_t ep)  
*Default data received callback.*

### 9.78.1 Detailed Description

Serial over USB Driver macros and structures.

## 9.79 spi.c File Reference

SPI Driver code.

```
#include "hal.h"
```

### Functions

- void **spilInit** (void)  
*SPI Driver initialization.*
- void **spiObjectInit** (SPIDriver \*spip)  
*Initializes the standard part of a `SPIDriver` structure.*
- void **spiStart** (SPIDriver \*spip, const SPICconfig \*config)  
*Configures and activates the SPI peripheral.*
- void **spiStop** (SPIDriver \*spip)  
*Deactivates the SPI peripheral.*
- void **spiSelect** (SPIDriver \*spip)  
*Asserts the slave select signal and prepares for transfers.*
- void **spiUnselect** (SPIDriver \*spip)  
*Deasserts the slave select signal.*
- void **spiStartIgnore** (SPIDriver \*spip, size\_t n)  
*Ignores data on the SPI bus.*
- void **spiStartExchange** (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)  
*Exchanges data on the SPI bus.*
- void **spiStartSend** (SPIDriver \*spip, size\_t n, const void \*txbuf)  
*Sends data over the SPI bus.*
- void **spiStartReceive** (SPIDriver \*spip, size\_t n, void \*rxbuf)  
*Receives data from the SPI bus.*
- void **spilgnore** (SPIDriver \*spip, size\_t n)  
*Ignores data on the SPI bus.*
- void **spiExchange** (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)  
*Exchanges data on the SPI bus.*
- void **spiSend** (SPIDriver \*spip, size\_t n, const void \*txbuf)  
*Sends data over the SPI bus.*
- void **spiReceive** (SPIDriver \*spip, size\_t n, void \*rxbuf)  
*Receives data from the SPI bus.*
- void **spiAcquireBus** (SPIDriver \*spip)  
*Gains exclusive access to the SPI bus.*
- void **spiReleaseBus** (SPIDriver \*spip)  
*Releases exclusive access to the SPI bus.*

### 9.79.1 Detailed Description

SPI Driver code.

## 9.80 spi.h File Reference

SPI Driver macros and structures.

```
#include "spi_lld.h"
```

### Macros

#### SPI configuration options

- #define `SPI_USE_WAIT` TRUE  
*Enables synchronous APIs.*
- #define `SPI_USE_MUTUAL_EXCLUSION` TRUE  
*Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.*

#### Macro Functions

- #define `spiSelectl`(spip)  
*Asserts the slave select signal and prepares for transfers.*
- #define `spiUnselectl`(spip)  
*Deasserts the slave select signal.*
- #define `spiStartIgnorel`(spip, n)  
*Ignores data on the SPI bus.*
- #define `spiStartExchangel`(spip, n, txbuf, rxbuf)  
*Exchanges data on the SPI bus.*
- #define `spiStartSendl`(spip, n, txbuf)  
*Sends data over the SPI bus.*
- #define `spiStartReceivel`(spip, n, rxbuf)  
*Receives data from the SPI bus.*
- #define `spiPolledExchange`(spip, frame) `spi_lld_polled_exchange`(spip, frame)  
*Exchanges one frame using a polled wait.*

#### Low level driver helper macros

- #define `_spi_wakeup_isr`(spip)  
*Wakes up the waiting thread.*
- #define `_spi_isr_code`(spip)  
*Common ISR code.*

### Enumerations

- enum `spistate_t` {  
`SPI_UNINIT` = 0, `SPI_STOP` = 1, `SPI_READY` = 2, `SPI_ACTIVE` = 3,  
`SPI_COMPLETE` = 4 }  
*Driver state machine possible states.*

## Functions

- void **spilInit** (void)
 

*SPI Driver initialization.*
- void **spiObjectInit** (SPIDriver \*spip)
 

*Initializes the standard part of a `SPIDriver` structure.*
- void **spiStart** (SPIDriver \*spip, const SPIConfig \*config)
 

*Configures and activates the SPI peripheral.*
- void **spiStop** (SPIDriver \*spip)
 

*Deactivates the SPI peripheral.*
- void **spiSelect** (SPIDriver \*spip)
 

*Asserts the slave select signal and prepares for transfers.*
- void **spiUnselect** (SPIDriver \*spip)
 

*Deasserts the slave select signal.*
- void **spiStartIgnore** (SPIDriver \*spip, size\_t n)
 

*Ignores data on the SPI bus.*
- void **spiStartExchange** (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)
 

*Exchanges data on the SPI bus.*
- void **spiStartSend** (SPIDriver \*spip, size\_t n, const void \*txbuf)
 

*Sends data over the SPI bus.*
- void **spiStartReceive** (SPIDriver \*spip, size\_t n, void \*rxbuf)
 

*Receives data from the SPI bus.*
- void **spilIgnore** (SPIDriver \*spip, size\_t n)
 

*Ignores data on the SPI bus.*
- void **spiExchange** (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)
 

*Exchanges data on the SPI bus.*
- void **spiSend** (SPIDriver \*spip, size\_t n, const void \*txbuf)
 

*Sends data over the SPI bus.*
- void **spiReceive** (SPIDriver \*spip, size\_t n, void \*rxbuf)
 

*Receives data from the SPI bus.*
- void **spiAcquireBus** (SPIDriver \*spip)
 

*Gains exclusive access to the SPI bus.*
- void **spiReleaseBus** (SPIDriver \*spip)
 

*Releases exclusive access to the SPI bus.*

### 9.80.1 Detailed Description

SPI Driver macros and structures.

## 9.81 spi\_lld.c File Reference

PLATFORM SPI subsystem low level driver source.

```
#include "hal.h"
```

## Functions

- void `spi_lld_init` (void)  
*Low level SPI driver initialization.*
- void `spi_lld_start` (SPIDriver \*spip)  
*Configures and activates the SPI peripheral.*
- void `spi_lld_stop` (SPIDriver \*spip)  
*Deactivates the SPI peripheral.*
- void `spi_lld_select` (SPIDriver \*spip)  
*Asserts the slave select signal and prepares for transfers.*
- void `spi_lld_unselect` (SPIDriver \*spip)  
*Deasserts the slave select signal.*
- void `spi_lld_ignore` (SPIDriver \*spip, size\_t n)  
*Ignores data on the SPI bus.*
- void `spi_lld_exchange` (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rdbuf)  
*Exchanges data on the SPI bus.*
- void `spi_lld_send` (SPIDriver \*spip, size\_t n, const void \*txbuf)  
*Sends data over the SPI bus.*
- void `spi_lld_receive` (SPIDriver \*spip, size\_t n, void \*rdbuf)  
*Receives data from the SPI bus.*
- uint16\_t `spi_lld_polled_exchange` (SPIDriver \*spip, uint16\_t frame)  
*Exchanges one frame using a polled wait.*

## Variables

- SPIDriver `SPI1`  
*SPI1 driver identifier.*

### 9.81.1 Detailed Description

PLATFORM SPI subsystem low level driver source.

## 9.82 spi\_lld.h File Reference

PLATFORM SPI subsystem low level driver header.

### Data Structures

- struct `SPIConfig`  
*Driver configuration structure.*
- struct `SPIDriver`  
*Structure representing an SPI driver.*

### Macros

#### PLATFORM configuration options

- #define `PLATFORM_SPI_USE_SPI1` FALSE  
*SPI1 driver enable switch.*

## Typedefs

- **typedef struct SPIDriver SPIDriver**  
*Type of a structure representing an SPI driver.*
- **typedef void(\* spicallback\_t) (SPIDriver \*spip)**  
*SPI notification callback type.*

## Functions

- **void spi\_lld\_init (void)**  
*Low level SPI driver initialization.*
- **void spi\_lld\_start (SPIDriver \*spip)**  
*Configures and activates the SPI peripheral.*
- **void spi\_lld\_stop (SPIDriver \*spip)**  
*Deactivates the SPI peripheral.*
- **void spi\_lld\_select (SPIDriver \*spip)**  
*Asserts the slave select signal and prepares for transfers.*
- **void spi\_lld\_unselect (SPIDriver \*spip)**  
*Deasserts the slave select signal.*
- **void spi\_lld\_ignore (SPIDriver \*spip, size\_t n)**  
*Ignores data on the SPI bus.*
- **void spi\_lld\_exchange (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)**  
*Exchanges data on the SPI bus.*
- **void spi\_lld\_send (SPIDriver \*spip, size\_t n, const void \*txbuf)**  
*Sends data over the SPI bus.*
- **void spi\_lld\_receive (SPIDriver \*spip, size\_t n, void \*rxbuf)**  
*Receives data from the SPI bus.*
- **uint16\_t spi\_lld\_polled\_exchange (SPIDriver \*spip, uint16\_t frame)**  
*Exchanges one frame using a polled wait.*

### 9.82.1 Detailed Description

PLATFORM SPI subsystem low level driver header.

## 9.83 st.c File Reference

ST Driver code.

```
#include "hal.h"
```

## Functions

- **void stInit (void)**  
*ST Driver initialization.*
- **void stStartAlarm (systime\_t abstime)**  
*Starts the alarm.*
- **void stStopAlarm (void)**  
*Stops the alarm interrupt.*
- **void stSetAlarm (systime\_t abstime)**

- Sets the alarm time.
- **systime\_t stGetAlarm (void)**  
*Returns the current alarm time.*

### 9.83.1 Detailed Description

ST Driver code.

## 9.84 st.h File Reference

ST Driver macros and structures.

```
#include "st_lld.h"
```

### Macros

#### Macro Functions

- **#define stGetCounter() st\_lld\_get\_counter()**  
*Returns the time counter value.*
- **#define stIsAlarmActive() st\_lld\_is\_alarm\_active()**  
*Determines if the alarm is active.*

### Functions

- **void stInit (void)**  
*ST Driver initialization.*
- **void stStartAlarm (systime\_t abstime)**  
*Starts the alarm.*
- **void stStopAlarm (void)**  
*Stops the alarm interrupt.*
- **void stSetAlarm (systime\_t abstime)**  
*Sets the alarm time.*
- **systime\_t stGetAlarm (void)**  
*Returns the current alarm time.*

### 9.84.1 Detailed Description

ST Driver macros and structures.

This header is designed to be include-able without having to include other files from the HAL.

## 9.85 st\_lld.c File Reference

PLATFORM ST subsystem low level driver source.

```
#include "hal.h"
```

## Functions

- void [st\\_lld\\_init](#) (void)  
*Low level ST driver initialization.*

### 9.85.1 Detailed Description

PLATFORM ST subsystem low level driver source.

## 9.86 st\_lld.h File Reference

PLATFORM ST subsystem low level driver header.

## Functions

- void [st\\_lld\\_init](#) (void)  
*Low level ST driver initialization.*
- static [systime\\_t st\\_lld\\_get\\_counter](#) (void)  
*Returns the time counter value.*
- static void [st\\_lld\\_start\\_alarm](#) ([systime\\_t](#) abstime)  
*Starts the alarm.*
- static void [st\\_lld\\_stop\\_alarm](#) (void)  
*Stops the alarm interrupt.*
- static void [st\\_lld\\_set\\_alarm](#) ([systime\\_t](#) abstime)  
*Sets the alarm time.*
- static [systime\\_t st\\_lld\\_get\\_alarm](#) (void)  
*Returns the current alarm time.*
- static bool [st\\_lld\\_is\\_alarm\\_active](#) (void)  
*Determines if the alarm is active.*

### 9.86.1 Detailed Description

PLATFORM ST subsystem low level driver header.

This header is designed to be include-able without having to include other files from the HAL.

## 9.87 uart.c File Reference

UART Driver code.

```
#include "hal.h"
```

## Functions

- void [uartInit](#) (void)  
*UART Driver initialization.*
- void [uartObjectInit](#) ([UARTDriver](#) \*uartp)  
*Initializes the standard part of a [UARTDriver](#) structure.*

- void `uartStart` (`UARTDriver` \*`uartp`, const `UARTConfig` \*`config`)  
*Configures and activates the UART peripheral.*
- void `uartStop` (`UARTDriver` \*`uartp`)  
*Deactivates the UART peripheral.*
- void `uartStartSend` (`UARTDriver` \*`uartp`, `size_t` `n`, const `void` \*`txbuf`)  
*Starts a transmission on the UART peripheral.*
- void `uartStartSendl` (`UARTDriver` \*`uartp`, `size_t` `n`, const `void` \*`txbuf`)  
*Starts a transmission on the UART peripheral.*
- `size_t` `uartStopSend` (`UARTDriver` \*`uartp`)  
*Stops any ongoing transmission.*
- `size_t` `uartStopSendl` (`UARTDriver` \*`uartp`)  
*Stops any ongoing transmission.*
- void `uartStartReceive` (`UARTDriver` \*`uartp`, `size_t` `n`, `void` \*`rxbuf`)  
*Starts a receive operation on the UART peripheral.*
- void `uartStartReceivevl` (`UARTDriver` \*`uartp`, `size_t` `n`, `void` \*`rxbuf`)  
*Starts a receive operation on the UART peripheral.*
- `size_t` `uartStopReceive` (`UARTDriver` \*`uartp`)  
*Stops any ongoing receive operation.*
- `size_t` `uartStopReceivevl` (`UARTDriver` \*`uartp`)  
*Stops any ongoing receive operation.*
- `msg_t` `uartSendTimeout` (`UARTDriver` \*`uartp`, `size_t` \*`np`, const `void` \*`txbuf`, `systime_t` `timeout`)  
*Performs a transmission on the UART peripheral.*
- `msg_t` `uartSendFullTimeout` (`UARTDriver` \*`uartp`, `size_t` \*`np`, const `void` \*`txbuf`, `systime_t` `timeout`)  
*Performs a transmission on the UART peripheral.*
- `msg_t` `uartReceiveTimeout` (`UARTDriver` \*`uartp`, `size_t` \*`np`, `void` \*`rxbuf`, `systime_t` `timeout`)  
*Performs a receive operation on the UART peripheral.*
- void `uartAcquireBus` (`UARTDriver` \*`uartp`)  
*Gains exclusive access to the UART bus.*
- void `uartReleaseBus` (`UARTDriver` \*`uartp`)  
*Releases exclusive access to the UART bus.*

### 9.87.1 Detailed Description

UART Driver code.

## 9.88 uart.h File Reference

UART Driver macros and structures.

```
#include "uart_llld.h"
```

### Macros

#### UART status flags

- #define `UART_NO_ERROR` 0  
*No pending conditions.*
- #define `UART_PARITY_ERROR` 4  
*Parity error happened.*
- #define `UART_FRAMING_ERROR` 8

- `#define UART_OVERRUN_ERROR 16`  
*Framing error happened.*
- `#define UART_NOISE_ERROR 32`  
*Overflow happened.*
- `#define UART_BREAK_DETECTED 64`  
*Noise on the line.*
- `#define UART_BREAK_DETECTED 64`  
*Break detected.*

### UART configuration options

- `#define UART_USE_WAIT FALSE`  
*Enables synchronous APIs.*
- `#define UART_USE_MUTUAL_EXCLUSION FALSE`  
*Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.*

### Low level driver helper macros

- `#define _uart_wakeup_tx1_isr(uartp)`  
*Wakes up the waiting thread in case of early TX complete.*
- `#define _uart_wakeup_tx2_isr(uartp)`  
*Wakes up the waiting thread in case of late TX complete.*
- `#define _uart_wakeup_rx_complete_isr(uartp)`  
*Wakes up the waiting thread in case of RX complete.*
- `#define _uart_wakeup_rx_error_isr(uartp)`  
*Wakes up the waiting thread in case of RX error.*
- `#define _uart_tx1_isr_code(uartp)`  
*Common ISR code for early TX.*
- `#define _uart_tx2_isr_code(uartp)`  
*Common ISR code for late TX.*
- `#define _uart_rx_complete_isr_code(uartp)`  
*Common ISR code for RX complete.*
- `#define _uart_rx_error_isr_code(uartp, errors)`  
*Common ISR code for RX error.*
- `#define _uart_rx_idle_code(uartp)`  
*Common ISR code for RX on idle.*

### Enumerations

- enum `uartstate_t` { `UART_UNINIT` = 0, `UART_STOP` = 1, `UART_READY` = 2 }  
*Driver state machine possible states.*
- enum `uartxstate_t` { `UART_TX_IDLE` = 0, `UART_TX_ACTIVE` = 1, `UART_TX_COMPLETE` = 2 }  
*Transmitter state machine states.*
- enum `uartrxstate_t` { `UART_RX_IDLE` = 0, `UART_RX_ACTIVE` = 1, `UART_RX_COMPLETE` = 2 }  
*Receiver state machine states.*

### Functions

- void `uartInit (void)`  
*UART Driver initialization.*
- void `uartObjectInit (UARTDriver *uartp)`  
*Initializes the standard part of a `UARTDriver` structure.*
- void `uartStart (UARTDriver *uartp, const UARTConfig *config)`  
*Configures and activates the UART peripheral.*
- void `uartStop (UARTDriver *uartp)`  
*Deactivates the UART peripheral.*

- void `uartStartSend (UARTDriver *uartp, size_t n, const void *txbuf)`  
*Starts a transmission on the UART peripheral.*
- void `uartStartSendl (UARTDriver *uartp, size_t n, const void *txbuf)`  
*Starts a transmission on the UART peripheral.*
- size\_t `uartStopSend (UARTDriver *uartp)`  
*Stops any ongoing transmission.*
- size\_t `uartStopSendl (UARTDriver *uartp)`  
*Stops any ongoing transmission.*
- void `uartStartReceive (UARTDriver *uartp, size_t n, void *rxbuf)`  
*Starts a receive operation on the UART peripheral.*
- void `uartStartReceivel (UARTDriver *uartp, size_t n, void *rxbuf)`  
*Starts a receive operation on the UART peripheral.*
- size\_t `uartStopReceive (UARTDriver *uartp)`  
*Stops any ongoing receive operation.*
- size\_t `uartStopReceivel (UARTDriver *uartp)`  
*Stops any ongoing receive operation.*
- msg\_t `uartSendTimeout (UARTDriver *uartp, size_t *np, const void *txbuf, systime_t timeout)`  
*Performs a transmission on the UART peripheral.*
- msg\_t `uartSendFullTimeout (UARTDriver *uartp, size_t *np, const void *txbuf, systime_t timeout)`  
*Performs a transmission on the UART peripheral.*
- msg\_t `uartReceiveTimeout (UARTDriver *uartp, size_t *np, void *rxbuf, systime_t timeout)`  
*Performs a receive operation on the UART peripheral.*
- void `uartAcquireBus (UARTDriver *uartp)`  
*Gains exclusive access to the UART bus.*
- void `uartReleaseBus (UARTDriver *uartp)`  
*Releases exclusive access to the UART bus.*

### 9.88.1 Detailed Description

UART Driver macros and structures.

## 9.89 uart\_lld.c File Reference

PLATFORM UART subsystem low level driver source.

```
#include "hal.h"
```

### Functions

- void `uart_lld_init (void)`  
*Low level UART driver initialization.*
- void `uart_lld_start (UARTDriver *uartp)`  
*Configures and activates the UART peripheral.*
- void `uart_lld_stop (UARTDriver *uartp)`  
*Deactivates the UART peripheral.*
- void `uart_lld_start_send (UARTDriver *uartp, size_t n, const void *txbuf)`  
*Starts a transmission on the UART peripheral.*
- size\_t `uart_lld_stop_send (UARTDriver *uartp)`  
*Stops any ongoing transmission.*

- void [uart\\_ll\\_start\\_receive](#) (UARTDriver \*uartp, size\_t n, void \*rdbuf)  
*Starts a receive operation on the UART peripheral.*
- size\_t [uart\\_ll\\_stop\\_receive](#) (UARTDriver \*uartp)  
*Stops any ongoing receive operation.*

## Variables

- [UARTDriver UARTD1](#)  
*UART1 driver identifier.*

### 9.89.1 Detailed Description

PLATFORM UART subsystem low level driver source.

## 9.90 uart\_ll.h File Reference

PLATFORM UART subsystem low level driver header.

### Data Structures

- struct [UARTConfig](#)  
*Driver configuration structure.*
- struct [UARTDriver](#)  
*Structure representing an UART driver.*

### Macros

#### PLATFORM configuration options

- #define [PLATFORM\\_UART\\_USE\\_UART1](#) FALSE  
*UART driver enable switch.*

### Typedefs

- typedef uint32\_t [uartflags\\_t](#)  
*UART driver condition flags type.*
- typedef struct [UARTDriver](#) [UARTDriver](#)  
*Type of structure representing an UART driver.*
- typedef void(\* [uartcb\\_t](#)) ([UARTDriver](#) \*uartp)  
*Generic UART notification callback type.*
- typedef void(\* [uartccb\\_t](#)) ([UARTDriver](#) \*uartp, uint16\_t c)  
*Character received UART notification callback type.*
- typedef void(\* [uar tcb\\_t](#)) ([UARTDriver](#) \*uartp, [uartflags\\_t](#) e)  
*Receive error UART notification callback type.*

## Functions

- void `uart_lld_init` (void)  
*Low level UART driver initialization.*
- void `uart_lld_start` (`UARTDriver` \*uartp)  
*Configures and activates the UART peripheral.*
- void `uart_lld_stop` (`UARTDriver` \*uartp)  
*Deactivates the UART peripheral.*
- void `uart_lld_start_send` (`UARTDriver` \*uartp, `size_t` n, const `void` \*txbuf)  
*Starts a transmission on the UART peripheral.*
- `size_t` `uart_lld_stop_send` (`UARTDriver` \*uartp)  
*Stops any ongoing transmission.*
- void `uart_lld_start_receive` (`UARTDriver` \*uartp, `size_t` n, `void` \*rxbuf)  
*Starts a receive operation on the UART peripheral.*
- `size_t` `uart_lld_stop_receive` (`UARTDriver` \*uartp)  
*Stops any ongoing receive operation.*

### 9.90.1 Detailed Description

PLATFORM UART subsystem low level driver header.

## 9.91 usb.c File Reference

USB Driver code.

```
#include <string.h>
#include "hal.h"
```

## Functions

- static void `set_address` (`USBDriver` \*usbp)  
*SET ADDRESS transaction callback.*
- static bool `default_handler` (`USBDriver` \*usbp)  
*Standard requests handler.*
- void `usbInit` (void)  
*USB Driver initialization.*
- void `usbObjectInit` (`USBDriver` \*usbp)  
*Initializes the standard part of a `USBDriver` structure.*
- void `usbStart` (`USBDriver` \*usbp, const `USBConfig` \*config)  
*Configures and activates the USB peripheral.*
- void `usbStop` (`USBDriver` \*usbp)  
*Deactivates the USB peripheral.*
- void `usbInitEndpointI` (`USBDriver` \*usbp, `usbep_t` ep, const `USBEndpointConfig` \*epcp)  
*Enables an endpoint.*
- void `usbDisableEndpointsI` (`USBDriver` \*usbp)  
*Disables all the active endpoints.*
- void `usbStartReceiveI` (`USBDriver` \*usbp, `usbep_t` ep, `uint8_t` \*buf, `size_t` n)  
*Starts a receive transaction on an OUT endpoint.*
- void `usbStartTransmitI` (`USBDriver` \*usbp, `usbep_t` ep, const `uint8_t` \*buf, `size_t` n)

- **msg\_t usbReceive (USBDriver \*usbp, usbep\_t ep, uint8\_t \*buf, size\_t n)**  
*Performs a receive transaction on an OUT endpoint.*
- **msg\_t usbTransmit (USBDriver \*usbp, usbep\_t ep, const uint8\_t \*buf, size\_t n)**  
*Performs a transmit transaction on an IN endpoint.*
- **bool usbStallReceiveI (USBDriver \*usbp, usbep\_t ep)**  
*Stalls an OUT endpoint.*
- **bool usbStallTransmit (USBDriver \*usbp, usbep\_t ep)**  
*Stalls an IN endpoint.*
- **void \_usb\_reset (USBDriver \*usbp)**  
*USB reset routine.*
- **void \_usb\_suspend (USBDriver \*usbp)**  
*USB suspend routine.*
- **void \_usb\_wakeup (USBDriver \*usbp)**  
*USB wake-up routine.*
- **void \_usb\_ep0setup (USBDriver \*usbp, usbep\_t ep)**  
*Default EP0 SETUP callback.*
- **void \_usb\_ep0in (USBDriver \*usbp, usbep\_t ep)**  
*Default EP0 IN callback.*
- **void \_usb\_ep0out (USBDriver \*usbp, usbep\_t ep)**  
*Default EP0 OUT callback.*

### 9.91.1 Detailed Description

USB Driver code.

## 9.92 usb.h File Reference

USB Driver macros and structures.

```
#include "usb_llld.h"
```

### Data Structures

- **struct USBDescriptor**  
*Type of an USB descriptor.*

### Macros

- **#define USB\_USE\_WAIT FALSE**  
*Enables synchronous APIs.*

### Helper macros for USB descriptors

- **#define USB\_DESC\_INDEX(i) ((uint8\_t)(i))**  
*Helper macro for index values into descriptor strings.*
- **#define USB\_DESC\_BYTE(b) ((uint8\_t)(b))**  
*Helper macro for byte values into descriptor strings.*
- **#define USB\_DESC\_WORD(w)**  
*Helper macro for word values into descriptor strings.*

- `#define USB_DESC_BCD(bcd)`  
*Helper macro for BCD values into descriptor strings.*
- `#define USB_DESC_DEVICE_SIZE 18U`
- `#define USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations)`  
*Device Descriptor helper macro.*
- `#define USB_DESC_CONFIGURATION_SIZE 9U`  
*Configuration Descriptor size.*
- `#define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower)`  
*Configuration Descriptor helper macro.*
- `#define USB_DESC_INTERFACE_SIZE 9U`  
*Interface Descriptor size.*
- `#define USB_DESC_INTERFACE(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol, iInterface)`  
*Interface Descriptor helper macro.*
- `#define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U`  
*Interface Association Descriptor size.*
- `#define USB_DESC_INTERFACE_ASSOCIATION(bFirstInterface, bInterfaceCount, bFunctionClass, bFunctionSubClass, bFunctionProtocol, iInterface)`  
*Interface Association Descriptor helper macro.*
- `#define USB_DESC_ENDPOINT_SIZE 7U`  
*Endpoint Descriptor size.*
- `#define USB_DESC_ENDPOINT(bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval)`  
*Endpoint Descriptor helper macro.*

### Endpoint types and settings

- `#define USB_EP_MODE_TYPE 0x0003U`
- `#define USB_EP_MODE_TYPE_CTRL 0x0000U`
- `#define USB_EP_MODE_TYPE_ISOC 0x0001U`
- `#define USB_EP_MODE_TYPE_BULK 0x0002U`
- `#define USB_EP_MODE_TYPE_INTR 0x0003U`

### Macro Functions

- `#define usbGetDriverState(usbp) ((usbp)->state)`  
*Returns the driver state.*
- `#define usbConnectBus(usbp) usb_lld_connect_bus(usbp)`  
*Connects the USB device.*
- `#define usbDisconnectBus(usbp) usb_lld_disconnect_bus(usbp)`  
*Disconnect the USB device.*
- `#define usbGetFrameNumberX(usbp) usb_lld_get_frame_number(usbp)`  
*Returns the current frame number.*
- `#define usbGetTransmitStatus1(usbp, ep) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)`  
*Returns the status of an IN endpoint.*
- `#define usbGetReceiveStatus1(usbp, ep) (((usbp)->receiving & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)`  
*Returns the status of an OUT endpoint.*
- `#define usbGetReceiveTransactionSizeX(usbp, ep) usb_lld_get_transaction_size(usbp, ep)`  
*Returns the exact size of a receive transaction.*
- `#define usbSetupTransfer(usbp, buf, n, endcb)`  
*Request transfer setup.*
- `#define usbReadSetup(usbp, ep, buf) usb_lld_read_setup(usbp, ep, buf)`  
*Reads a setup packet from the dedicated packet buffer.*

### Low level driver helper macros

- `#define _usb_isr_invoke_event_cb(usbp, evt)`  
*Common ISR code, usb event callback.*
- `#define _usb_isr_invoke_sof_cb(usbp)`  
*Common ISR code, SOF callback.*
- `#define _usb_isr_invoke_setup_cb(usbp, ep)`  
*Common ISR code, setup packet callback.*
- `#define _usb_isr_invoke_in_cb(usbp, ep)`  
*Common ISR code, IN endpoint callback.*
- `#define _usb_isr_invoke_out_cb(usbp, ep)`  
*Common ISR code, OUT endpoint event.*

## Typedefs

- `typedef struct USBDriver USBDriver`  
*Type of a structure representing an USB driver.*
- `typedef uint8_t usbep_t`  
*Type of an endpoint identifier.*
- `typedef void(* usbcallback_t) (USBDriver *usbp)`  
*Type of an USB generic notification callback.*
- `typedef void(* usbepcallback_t) (USBDriver *usbp, usbep_t ep)`  
*Type of an USB endpoint callback.*
- `typedef void(* usbeventcb_t) (USBDriver *usbp, usbevent_t event)`  
*Type of an USB event notification callback.*
- `typedef bool(* usbreqhandler_t) (USBDriver *usbp)`  
*Type of a requests handler callback.*
- `typedef const USBDescriptor *(* usbgetdescriptor_t) (USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`  
*Type of an USB descriptor-retrieving callback.*

## Enumerations

- `enum usbstate_t {`  
`USB_UNINIT = 0, USB_STOP = 1, USB_READY = 2, USB_SELECTED = 3,`  
`USB_ACTIVE = 4, USB_SUSPENDED = 5 }`  
*Type of a driver state machine possible states.*
- `enum usbepstatus_t { EP_STATUS_DISABLED = 0, EP_STATUS_STALLED = 1, EP_STATUS_ACTIVE = 2 }`  
*Type of an endpoint status.*
- `enum usbep0state_t {`  
`USB_EP0_WAITING_SETUP, USB_EP0_TX, USB_EP0_WAITING_TX0, USB_EP0_WAITING_STS,`  
`USB_EP0_RX, USB_EP0_SENDING_STS, USB_EP0_ERROR }`  
*Type of an endpoint zero state machine states.*
- `enum usbevent_t {`  
`USB_EVENT_RESET = 0, USB_EVENT_ADDRESS = 1, USB_EVENT_CONFIGURED = 2, USB_EVENT_SUSPEND = 3,`  
`USB_EVENT_WAKEUP = 4, USB_EVENT_STALLED = 5 }`  
*Type of an enumeration of the possible USB events.*

## Functions

- void `usbInit` (void)
 

*USB Driver initialization.*
- void `usbObjectInit` (`USBDriver` \*`usbp`)
 

*Initializes the standard part of a `USBDriver` structure.*
- void `usbStart` (`USBDriver` \*`usbp`, const `USBConfig` \*`config`)
 

*Configures and activates the USB peripheral.*
- void `usbStop` (`USBDriver` \*`usbp`)
 

*Deactivates the USB peripheral.*
- void `usbInitEndpointl` (`USBDriver` \*`usbp`, `usbep_t` `ep`, const `USBEndpointConfig` \*`epcp`)
 

*Enables an endpoint.*
- void `usbDisableEndpointsl` (`USBDriver` \*`usbp`)
 

*Disables all the active endpoints.*
- void `usbStartReceive1` (`USBDriver` \*`usbp`, `usbep_t` `ep`, `uint8_t` \*`buf`, `size_t` `n`)
 

*Starts a receive transaction on an OUT endpoint.*
- void `usbStartTransmitl` (`USBDriver` \*`usbp`, `usbep_t` `ep`, const `uint8_t` \*`buf`, `size_t` `n`)
 

*Starts a transmit transaction on an IN endpoint.*
- `msg_t` `usbReceive` (`USBDriver` \*`usbp`, `usbep_t` `ep`, `uint8_t` \*`buf`, `size_t` `n`)
 

*Performs a receive transaction on an OUT endpoint.*
- `msg_t` `usbTransmit` (`USBDriver` \*`usbp`, `usbep_t` `ep`, const `uint8_t` \*`buf`, `size_t` `n`)
 

*Performs a transmit transaction on an IN endpoint.*
- bool `usbStallReceive1` (`USBDriver` \*`usbp`, `usbep_t` `ep`)
 

*Stalls an OUT endpoint.*
- bool `usbStallTransmitl` (`USBDriver` \*`usbp`, `usbep_t` `ep`)
 

*Stalls an IN endpoint.*
- void `_usb_reset` (`USBDriver` \*`usbp`)
 

*USB reset routine.*
- void `_usb_suspend` (`USBDriver` \*`usbp`)
 

*USB suspend routine.*
- void `_usb_wakeup` (`USBDriver` \*`usbp`)
 

*USB wake-up routine.*
- void `_usb_ep0setup` (`USBDriver` \*`usbp`, `usbep_t` `ep`)
 

*Default EP0 SETUP callback.*
- void `_usb_ep0in` (`USBDriver` \*`usbp`, `usbep_t` `ep`)
 

*Default EP0 IN callback.*
- void `_usb_ep0out` (`USBDriver` \*`usbp`, `usbep_t` `ep`)
 

*Default EP0 OUT callback.*

### 9.92.1 Detailed Description

USB Driver macros and structures.

## 9.93 usb\_cdc.h File Reference

USB CDC macros and structures.

### Data Structures

- struct `cdc_linecoding_t`

*Type of Line Coding structure.*

## Macros

### CDC specific messages.

- #define **CDC\_SEND\_ENCAPSULATED\_COMMAND** 0x00U
- #define **CDC\_GET\_ENCAPSULATED\_RESPONSE** 0x01U
- #define **CDC\_SET\_COMM\_FEATURE** 0x02U
- #define **CDC\_GET\_COMM\_FEATURE** 0x03U
- #define **CDC\_CLEAR\_COMM\_FEATURE** 0x04U
- #define **CDC\_SET\_AUX\_LINE\_STATE** 0x10U
- #define **CDC\_SET\_HOOK\_STATE** 0x11U
- #define **CDC\_PULSE\_SETUP** 0x12U
- #define **CDC\_SEND\_PULSE** 0x13U
- #define **CDC\_SET\_PULSE\_TIME** 0x14U
- #define **CDC\_RING\_AUX\_JACK** 0x15U
- #define **CDC\_SET\_LINE\_CODING** 0x20U
- #define **CDC\_GET\_LINE\_CODING** 0x21U
- #define **CDC\_SET\_CONTROL\_LINE\_STATE** 0x22U
- #define **CDC\_SEND\_BREAK** 0x23U
- #define **CDC\_SET\_RINGER\_PARMS** 0x30U
- #define **CDC\_GET\_RINGER\_PARMS** 0x31U
- #define **CDC\_SET\_OPERATION\_PARMS** 0x32U
- #define **CDC\_GET\_OPERATION\_PARMS** 0x33U

### CDC classes

- #define **CDC\_COMMUNICATION\_INTERFACE\_CLASS** 0x02U
- #define **CDC\_DATA\_INTERFACE\_CLASS** 0x0AU

### CDC subclasses

- #define **CDC\_ABSTRACT\_CONTROL\_MODEL** 0x02U

### CDC descriptors

- #define **CDC\_CS\_INTERFACE** 0x24U

### CDC subdescriptors

- #define **CDC\_HEADER** 0x00U
- #define **CDC\_CALL\_MANAGEMENT** 0x01U
- #define **CDC\_ABSTRACT\_CONTROL\_MANAGEMENT** 0x02U
- #define **CDC\_UNION** 0x06U

### Line Control bit definitions.

- #define **LC\_STOP\_1** 0U
- #define **LC\_STOP\_1P5** 1U
- #define **LC\_STOP\_2** 2U
- #define **LC\_PARITY\_NONE** 0U
- #define **LC\_PARITY\_ODD** 1U
- #define **LC\_PARITY\_EVEN** 2U
- #define **LC\_PARITY\_MARK** 3U
- #define **LC\_PARITY\_SPACE** 4U

## 9.93.1 Detailed Description

USB CDC macros and structures.

## 9.94 usb\_lld.c File Reference

PLATFORM USB subsystem low level driver source.

```
#include "hal.h"
```

### Functions

- void `usb_lld_init` (void)
 

*Low level USB driver initialization.*
- void `usb_lld_start` (USBDriver \*usbp)
 

*Configures and activates the USB peripheral.*
- void `usb_lld_stop` (USBDriver \*usbp)
 

*Deactivates the USB peripheral.*
- void `usb_lld_reset` (USBDriver \*usbp)
 

*USB low level reset routine.*
- void `usb_lld_set_address` (USBDriver \*usbp)
 

*Sets the USB address.*
- void `usb_lld_init_endpoint` (USBDriver \*usbp, usbep\_t ep)
 

*Enables an endpoint.*
- void `usb_lld_disable_endpoints` (USBDriver \*usbp)
 

*Disables all the active endpoints except the endpoint zero.*
- usbepstatus\_t `usb_lld_get_status_out` (USBDriver \*usbp, usbep\_t ep)
 

*Returns the status of an OUT endpoint.*
- usbepstatus\_t `usb_lld_get_status_in` (USBDriver \*usbp, usbep\_t ep)
 

*Returns the status of an IN endpoint.*
- void `usb_lld_read_setup` (USBDriver \*usbp, usbep\_t ep, uint8\_t \*buf)
 

*Reads a setup packet from the dedicated packet buffer.*
- void `usb_lld_prepare_receive` (USBDriver \*usbp, usbep\_t ep)
 

*Prepares for a receive operation.*
- void `usb_lld_prepare_transmit` (USBDriver \*usbp, usbep\_t ep)
 

*Prepares for a transmit operation.*
- void `usb_lld_start_out` (USBDriver \*usbp, usbep\_t ep)
 

*Starts a receive operation on an OUT endpoint.*
- void `usb_lld_start_in` (USBDriver \*usbp, usbep\_t ep)
 

*Starts a transmit operation on an IN endpoint.*
- void `usb_lld_stall_out` (USBDriver \*usbp, usbep\_t ep)
 

*Brings an OUT endpoint in the stalled state.*
- void `usb_lld_stall_in` (USBDriver \*usbp, usbep\_t ep)
 

*Brings an IN endpoint in the stalled state.*
- void `usb_lld_clear_out` (USBDriver \*usbp, usbep\_t ep)
 

*Brings an OUT endpoint in the active state.*
- void `usb_lld_clear_in` (USBDriver \*usbp, usbep\_t ep)
 

*Brings an IN endpoint in the active state.*

## Variables

- [USBDriver USBD1](#)  
*USB1 driver identifier.*
- union {  
    [USBInEndpointState](#) in  
    *IN EP0 state.*  
    [USBOutEndpointState](#) out  
    *OUT EP0 state.*  
} [ep0\\_state](#)  
  
*EP0 state.*
- static const [USBEndpointConfig](#) ep0config  
*EP0 initialization structure.*

### 9.94.1 Detailed Description

PLATFORM USB subsystem low level driver source.

### 9.94.2 Variable Documentation

#### 9.94.2.1 [USBInEndpointState](#) in

IN EP0 state.

#### 9.94.2.2 [USBOutEndpointState](#) out

OUT EP0 state.

## 9.95 usb\_ll.h File Reference

PLATFORM USB subsystem low level driver header.

## Data Structures

- struct [USBInEndpointState](#)  
*Type of an IN endpoint state structure.*
- struct [USBOutEndpointState](#)  
*Type of an OUT endpoint state structure.*
- struct [USBEndpointConfig](#)  
*Type of an USB endpoint configuration structure.*
- struct [USBConfig](#)  
*Type of an USB driver configuration structure.*
- struct [USBDriver](#)  
*Structure representing an USB driver.*

## Macros

- `#define USB_MAX_ENDPOINTS 4`  
*Maximum endpoint address.*
- `#define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW`  
*Status stage handling method.*
- `#define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS`  
*The address can be changed immediately upon packet reception.*
- `#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW`  
*Method for set address acknowledge.*
- `#define usb_lld_get_frame_number(usbp) 0`  
*Returns the current frame number.*
- `#define usb_lld_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)`  
*Returns the exact size of a receive transaction.*
- `#define usb_lld_connect_bus(usbp)`  
*Connects the USB device.*
- `#define usb_lld_disconnect_bus(usbp)`  
*Disconnect the USB device.*

## PLATFORM configuration options

- `#define PLATFORM_USB_USE_USB1 FALSE`  
*USB driver enable switch.*

## Functions

- `void usb_lld_init (void)`  
*Low level USB driver initialization.*
- `void usb_lld_start (USBDriver *usbp)`  
*Configures and activates the USB peripheral.*
- `void usb_lld_stop (USBDriver *usbp)`  
*Deactivates the USB peripheral.*
- `void usb_lld_reset (USBDriver *usbp)`  
*USB low level reset routine.*
- `void usb_lld_set_address (USBDriver *usbp)`  
*Sets the USB address.*
- `void usb_lld_init_endpoint (USBDriver *usbp, usbep_t ep)`  
*Enables an endpoint.*
- `void usb_lld_disable_endpoints (USBDriver *usbp)`  
*Disables all the active endpoints except the endpoint zero.*
- `usbepstatus_t usb_lld_get_status_in (USBDriver *usbp, usbep_t ep)`  
*Returns the status of an IN endpoint.*
- `usbepstatus_t usb_lld_get_status_out (USBDriver *usbp, usbep_t ep)`  
*Returns the status of an OUT endpoint.*
- `void usb_lld_read_setup (USBDriver *usbp, usbep_t ep, uint8_t *buf)`  
*Reads a setup packet from the dedicated packet buffer.*
- `void usb_lld_prepare_receive (USBDriver *usbp, usbep_t ep)`  
*Prepares for a receive operation.*
- `void usb_lld_prepare_transmit (USBDriver *usbp, usbep_t ep)`  
*Prepares for a transmit operation.*
- `void usb_lld_start_out (USBDriver *usbp, usbep_t ep)`

- void **usb\_lld\_start\_in** (USBDriver \*usbp, usbep\_t ep)
 

*Starts a receive operation on an OUT endpoint.*
- void **usb\_lld\_stall\_out** (USBDriver \*usbp, usbep\_t ep)
 

*Starts a transmit operation on an IN endpoint.*
- void **usb\_lld\_stall\_in** (USBDriver \*usbp, usbep\_t ep)
 

*Brings an OUT endpoint in the stalled state.*
- void **usb\_lld\_clear\_out** (USBDriver \*usbp, usbep\_t ep)
 

*Brings an OUT endpoint in the active state.*
- void **usb\_lld\_clear\_in** (USBDriver \*usbp, usbep\_t ep)
 

*Brings an IN endpoint in the active state.*

### 9.95.1 Detailed Description

PLATFORM USB subsystem low level driver header.

## 9.96 wdg.c File Reference

WDG Driver code.

```
#include "hal.h"
```

### Functions

- void **wdgInit** (void)
 

*WDG Driver initialization.*
- void **wdgStart** (WDGDriver \*wdgp, const WDGConfig \*config)
 

*Configures and activates the WDG peripheral.*
- void **wdgStop** (WDGDriver \*wdgp)
 

*Deactivates the WDG peripheral.*
- void **wdgReset** (WDGDriver \*wdgp)
 

*Resets WDG's counter.*

### 9.96.1 Detailed Description

WDG Driver code.

## 9.97 wdg.h File Reference

WDG Driver macros and structures.

```
#include "wdg_lld.h"
```

### Macros

- #define **wdgReset**(wdgp) **wdg\_lld\_reset**(wdgp)
 

*Resets WDG's counter.*

## Enumerations

- enum `wdgstate_t` { `WDG_UNINIT` = 0, `WDG_STOP` = 1, `WDG_READY` = 2 }
- Driver state machine possible states.*

## Functions

- void `wdgInit` (void)  
*WDG Driver initialization.*
- void `wdgStart` (`WDGDriver` \*`wdgp`, const `WDGConfig` \*`config`)  
*Configures and activates the WDG peripheral.*
- void `wdgStop` (`WDGDriver` \*`wdgp`)  
*Deactivates the WDG peripheral.*
- void `wdgReset` (`WDGDriver` \*`wdgp`)  
*Resets WDG's counter.*

### 9.97.1 Detailed Description

WDG Driver macros and structures.

## 9.98 wdg\_lld.c File Reference

WDG Driver subsystem low level driver source template.

```
#include "hal.h"
```

## Functions

- void `wdg_lld_init` (void)  
*Low level WDG driver initialization.*
- void `wdg_lld_start` (`WDGDriver` \*`wdgp`)  
*Configures and activates the WDG peripheral.*
- void `wdg_lld_stop` (`WDGDriver` \*`wdgp`)  
*Deactivates the WDG peripheral.*
- void `wdg_lld_reset` (`WDGDriver` \*`wdgp`)  
*Reloads WDG's counter.*

### 9.98.1 Detailed Description

WDG Driver subsystem low level driver source template.

## 9.99 wdg\_lld.h File Reference

WDG Driver subsystem low level driver header template.

## Data Structures

- struct [WDGConfig](#)  
*Driver configuration structure.*
- struct [WDGDriver](#)  
*Structure representing an WDG driver.*

## Macros

### Configuration options

- #define [PLATFORM\\_WDG\\_USE\\_WDG1](#) FALSE  
*WDG1 driver enable switch.*

## Typedefs

- typedef struct [WDGDriver](#) [WDGDriver](#)  
*Type of a structure representing an WDG driver.*

## Functions

- void [wdg\\_ll\\_init](#) (void)  
*Low level WDG driver initialization.*
- void [wdg\\_ll\\_start](#) ([WDGDriver](#) \*wdgp)  
*Configures and activates the WDG peripheral.*
- void [wdg\\_ll\\_stop](#) ([WDGDriver](#) \*wdgp)  
*Deactivates the WDG peripheral.*
- void [wdg\\_ll\\_reset](#) ([WDGDriver](#) \*wdgp)  
*Reloads WDG's counter.*

### 9.99.1 Detailed Description

WDG Driver subsystem low level driver header template.



# Index

\_CHIBIOS\_HAL\_  
    HAL Driver, 96  
\_IOBUS\_DATA  
    PAL Driver, 297  
\_adc\_isr\_error\_code  
    ADC Driver, 23  
\_adc\_isr\_full\_code  
    ADC Driver, 23  
\_adc\_isr\_half\_code  
    ADC Driver, 22  
\_adc\_reset\_i  
    ADC Driver, 21  
\_adc\_reset\_s  
    ADC Driver, 21  
\_adc\_timeout\_isr  
    ADC Driver, 22  
\_adc\_wakeup\_isr  
    ADC Driver, 22  
\_base\_asynchronous\_channel\_data  
    Abstract I/O Channel, 122  
\_base\_asynchronous\_channel\_methods  
    Abstract I/O Channel, 122  
\_base\_block\_device\_data  
    Abstract I/O Block Device, 131  
\_base\_block\_device\_methods  
    Abstract I/O Block Device, 131  
\_base\_channel\_data  
    Abstract I/O Channel, 119  
\_base\_channel\_methods  
    Abstract I/O Channel, 119  
\_base\_sequential\_stream\_data  
    Abstract Streams, 154  
\_base\_sequential\_stream\_methods  
    Abstract Streams, 153  
\_dac\_isr\_error\_code  
    DAC Driver, 56  
\_dac\_isr\_full\_code  
    DAC Driver, 55  
\_dac\_isr\_half\_code  
    DAC Driver, 55  
\_dac\_reset\_i  
    DAC Driver, 54  
\_dac\_reset\_s  
    DAC Driver, 54  
\_dac\_timeout\_isr  
    DAC Driver, 55  
\_dac\_wait\_s  
    DAC Driver, 53  
\_dac\_wakeup\_isr  
    DAC Driver, 54  
  
\_file\_stream\_data  
    Abstract Files, 125  
\_file\_stream\_methods  
    Abstract Files, 125  
\_i2c\_wakeup\_error\_isr  
    I2C Driver, 159  
\_i2c\_wakeup\_isr  
    I2C Driver, 159  
\_i2s\_isr\_full\_code  
    I2S Driver, 172  
\_i2s\_isr\_half\_code  
    I2S Driver, 172  
\_icu\_isr\_invoke\_overflow\_cb  
    ICU Driver, 184  
\_icu\_isr\_invoke\_period\_cb  
    ICU Driver, 184  
\_icu\_isr\_invoke\_width\_cb  
    ICU Driver, 183  
\_mmc\_driver\_methods  
    MMC over SPI Driver, 262  
\_mmcsd\_block\_device\_data  
    MMC/SD Block Device, 287  
\_mmcsd\_block\_device\_methods  
    MMC/SD Block Device, 286  
\_mmcsd\_get\_capacity  
    MMC/SD Block Device, 288  
\_mmcsd\_get\_capacity\_ext  
    MMC/SD Block Device, 289  
\_mmcsd\_get\_slice  
    MMC/SD Block Device, 288  
\_mmcsd\_unpack\_csd\_mmc  
    MMC/SD Block Device, 290  
\_mmcsd\_unpack\_csd\_v10  
    MMC/SD Block Device, 290  
\_mmcsd\_unpack\_csd\_v20  
    MMC/SD Block Device, 291  
\_mmcsd\_unpack\_mmc\_cid  
    MMC/SD Block Device, 289  
\_mmcsd\_unpack\_sdc\_cid  
    MMC/SD Block Device, 289  
\_pal\_lld\_init  
    PAL Driver, 315  
\_pal\_lld\_setgroupmode  
    PAL Driver, 315  
\_rtc\_driver\_methods  
    RTC Driver, 342  
\_sdc\_driver\_methods  
    SDC Driver, 355

\_sdc\_wait\_for\_transfer\_state  
     SDC Driver, 364  
 \_serial\_driver\_data  
     Serial Driver, 385  
 \_serial\_driver\_methods  
     Serial Driver, 382  
 \_serial\_usb\_driver\_data  
     Serial over USB Driver, 396  
 \_serial\_usb\_driver\_methods  
     Serial over USB Driver, 396  
 \_spi\_isr\_code  
     SPI Driver, 412  
 \_spi\_wakeup\_isr  
     SPI Driver, 412  
 \_uart\_rx\_complete\_isr\_code  
     UART Driver, 443  
 \_uart\_rx\_error\_isr\_code  
     UART Driver, 443  
 \_uart\_rx\_idle\_code  
     UART Driver, 444  
 \_uart\_tx1\_isr\_code  
     UART Driver, 442  
 \_uart\_tx2\_isr\_code  
     UART Driver, 442  
 \_uart\_wakeup\_rx\_complete\_isr  
     UART Driver, 441  
 \_uart\_wakeup\_rx\_error\_isr  
     UART Driver, 441  
 \_uart\_wakeup\_tx1\_isr  
     UART Driver, 440  
 \_uart\_wakeup\_tx2\_isr  
     UART Driver, 440  
 \_usb\_ep0in  
     USB Driver, 493  
 \_usb\_ep0out  
     USB Driver, 494  
 \_usb\_ep0setup  
     USB Driver, 492  
 \_usb\_isr\_invoke\_event\_cb  
     USB Driver, 474  
 \_usb\_isr\_invoke\_in\_cb  
     USB Driver, 476  
 \_usb\_isr\_invoke\_out\_cb  
     USB Driver, 476  
 \_usb\_isr\_invoke\_setup\_cb  
     USB Driver, 476  
 \_usb\_isr\_invoke\_sof\_cb  
     USB Driver, 474  
 \_usb\_reset  
     USB Driver, 490  
 \_usb\_suspend  
     USB Driver, 491  
 \_usb\_wakeup  
     USB Driver, 491

ADC Driver, 17  
     \_adc\_isr\_error\_code, 23  
     \_adc\_isr\_full\_code, 23  
     \_adc\_isr\_half\_code, 22

\_adc\_reset\_i, 21  
 \_adc\_reset\_s, 21  
 \_adc\_timeout\_isr, 22  
 \_adc\_wakeup\_isr, 22  
 ADC\_ACTIVE, 25  
 ADC\_COMPLETE, 25  
 ADC\_ERR\_AWD, 25  
 ADC\_ERR\_DMAFAILURE, 25  
 ADC\_ERR\_OVERFLOW, 25  
 ADC\_ERROR, 25  
 ADC\_READY, 25  
 ADC\_STOP, 25  
 ADC\_UNINIT, 25  
 ADC\_USE\_MUTUAL\_EXCLUSION, 21  
 ADC\_USE\_WAIT, 21  
 ADCD1, 34  
 ADCDriver, 24  
 adc\_channels\_num\_t, 24  
 adc\_lld\_init, 32  
 adc\_lld\_start, 33  
 adc\_lld\_start\_conversion, 33  
 adc\_lld\_stop, 33  
 adc\_lld\_stop\_conversion, 33  
 adcAcquireBus, 31  
 adcConvert, 30  
 adcInit, 25  
 adcObjectInit, 26  
 adcReleaseBus, 32  
 adcStart, 26  
 adcStartConversion, 28  
 adcStartConversionl, 29  
 adcStop, 26  
 adcStopConversion, 29  
 adcStopConversionl, 30  
 adccallback\_t, 24  
 adcerror\_t, 25  
 adcerrorcallback\_t, 24  
 adcsample\_t, 24  
 adcstate\_t, 25  
 PLATFORM\_ADC\_USE\_ADC1, 24

ADC\_ACTIVE  
     ADC Driver, 25  
 ADC\_COMPLETE  
     ADC Driver, 25  
 ADC\_ERR\_AWD  
     ADC Driver, 25  
 ADC\_ERR\_DMAFAILURE  
     ADC Driver, 25  
 ADC\_ERR\_OVERFLOW  
     ADC Driver, 25  
 ADC\_ERROR  
     ADC Driver, 25  
 ADC\_READY  
     ADC Driver, 25  
 ADC\_STOP  
     ADC Driver, 25  
 ADC\_UNINIT  
     ADC Driver, 25

ADC\_USE\_MUTUAL\_EXCLUSION  
  ADC Driver, 21  
  Configuration, 219  
ADC\_USE\_WAIT  
  ADC Driver, 21  
  Configuration, 219  
ADCCfg, 511  
ADCConversionGroup, 512  
  circular, 513  
  end\_cb, 513  
  error\_cb, 513  
  num\_channels, 513  
ADCD1  
  ADC Driver, 34  
ADCDriver, 513  
  ADC Driver, 24  
  config, 515  
  depth, 515  
  grpp, 515  
  mutex, 515  
  samples, 515  
  state, 515  
  thread, 515  
ADVERTISE\_100BASE4  
  MII/RMII Header, 256  
ADVERTISE\_100FULL  
  MII/RMII Header, 256  
ADVERTISE\_100HALF  
  MII/RMII Header, 256  
ADVERTISE\_10FULL  
  MII/RMII Header, 256  
ADVERTISE\_10HALF  
  MII/RMII Header, 256  
ADVERTISE\_CSMA  
  MII/RMII Header, 256  
ADVERTISE\_LPACK  
  MII/RMII Header, 257  
ADVERTISE\_NPAGE  
  MII/RMII Header, 257  
ADVERTISE\_PAUSE\_ASYM  
  MII/RMII Header, 257  
ADVERTISE\_PAUSE\_CAP  
  MII/RMII Header, 256  
ADVERTISE\_RESV  
  MII/RMII Header, 257  
ADVERTISE\_RFAULT  
  MII/RMII Header, 257  
ADVERTISE\_SLCT  
  MII/RMII Header, 256  
Abstract Files, 124  
  \_file\_stream\_data, 125  
  \_file\_stream\_methods, 125  
FILE\_EOF, 125  
FILE\_ERROR, 125  
FILE\_OK, 125  
fileStreamClose, 127  
fileStreamGet, 127  
fileStreamGetError, 128  
fileStreamGetPosition, 128  
fileStreamGetSize, 128  
fileStreamPut, 126  
fileStreamRead, 126  
fileStreamSeek, 129  
fileStreamWrite, 125  
fileoffset\_t, 129  
Abstract I/O Block Device, 130  
  \_base\_block\_device\_data, 131  
  \_base\_block\_device\_methods, 131  
BLK\_ACTIVE, 136  
BLK\_CONNECTING, 136  
BLK\_DISCONNECTING, 136  
BLK\_READING, 136  
BLK\_READY, 136  
BLK\_STOP, 136  
BLK\_SYNCING, 136  
BLK\_UNINIT, 136  
BLK\_WRITING, 136  
blkConnect, 133  
blkDisconnect, 134  
blkGetDriverState, 132  
blkGetInfo, 135  
blkIsInserted, 133  
blkIsTransferring, 132  
blkIsWriteProtected, 133  
blkRead, 134  
blkSync, 135  
blkWrite, 135  
blkstate\_t, 136  
Abstract I/O Channel, 118  
  \_base\_asynchronous\_channel\_data, 122  
  \_base\_asynchronous\_channel\_methods, 122  
  \_base\_channel\_data, 119  
  \_base\_channel\_methods, 119  
CHN\_CONNECTED, 122  
CHN\_DISCONNECTED, 122  
CHN\_INPUT\_AVAILABLE, 122  
CHN\_NO\_ERROR, 122  
CHN\_OUTPUT\_EMPTY, 122  
CHN\_TRANSMISSION\_END, 122  
chnAddFlagsI, 123  
chnGetEventSource, 123  
chnGetTimeout, 120  
chnPutTimeout, 119  
chnRead, 121  
chnReadTimeout, 121  
chnWrite, 120  
chnWriteTimeout, 121  
Abstract Streams, 153  
  \_base\_sequential\_stream\_data, 154  
  \_base\_sequential\_stream\_methods, 153  
streamGet, 155  
streamPut, 154  
streamRead, 154  
streamWrite, 154  
adc.c, 619  
adc.h, 620

adc\_channels\_num\_t  
     ADC Driver, 24

adc\_lld.c, 621

adc\_lld.h, 621

adc\_lld\_init  
     ADC Driver, 32

adc\_lld\_start  
     ADC Driver, 33

adc\_lld\_start\_conversion  
     ADC Driver, 33

adc\_lld\_stop  
     ADC Driver, 33

adc\_lld\_stop\_conversion  
     ADC Driver, 33

adcAcquireBus  
     ADC Driver, 31

adcConvert  
     ADC Driver, 30

adlInit  
     ADC Driver, 25

adcObjectInit  
     ADC Driver, 26

adcReleaseBus  
     ADC Driver, 32

adcStart  
     ADC Driver, 26

adcStartConversion  
     ADC Driver, 28

adcStartConversionI  
     ADC Driver, 29

adcStop  
     ADC Driver, 26

adcStopConversion  
     ADC Driver, 29

adcStopConversionI  
     ADC Driver, 30

adccallback\_t  
     ADC Driver, 24

adcerror\_t  
     ADC Driver, 25

adcerrorcallback\_t  
     ADC Driver, 24

adcsample\_t  
     ADC Driver, 24

adcstate\_t  
     ADC Driver, 25

address  
     USBDriver, 611

BLK\_ACTIVE  
     Abstract I/O Block Device, 136

BLK\_CONNECTING  
     Abstract I/O Block Device, 136

BLK\_DISCONNECTING  
     Abstract I/O Block Device, 136

BLK\_READING  
     Abstract I/O Block Device, 136

BLK\_READY  
     Abstract I/O Block Device, 136

BLK\_STOP  
     Abstract I/O Block Device, 136

BLK\_SYNCING  
     Abstract I/O Block Device, 136

BLK\_UNINIT  
     Abstract I/O Block Device, 136

BLK\_WRITING  
     Abstract I/O Block Device, 136

BMCR\_ANENABLE  
     MII/RMII Header, 254

BMCR\_ANRESTART  
     MII/RMII Header, 254

BMCR\_CTST  
     MII/RMII Header, 254

BMCR\_FULLDPLX  
     MII/RMII Header, 254

BMCR\_ISOLATE  
     MII/RMII Header, 254

BMCR\_LOOPBACK  
     MII/RMII Header, 255

BMCR\_PDOWN  
     MII/RMII Header, 254

BMCR\_RESET  
     MII/RMII Header, 255

BMCR\_RESV  
     MII/RMII Header, 254

BMCR\_SPEED100  
     MII/RMII Header, 255

BMSR\_100BASE4  
     MII/RMII Header, 256

BMSR\_100FULL  
     MII/RMII Header, 256

BMSR\_100HALF  
     MII/RMII Header, 256

BMSR\_10FULL  
     MII/RMII Header, 256

BMSR\_10HALF  
     MII/RMII Header, 255

BMSR\_ANEGCAPABLE  
     MII/RMII Header, 255

BMSR\_ANEGCOMPLETE  
     MII/RMII Header, 255

BMSR\_ERCAP  
     MII/RMII Header, 255

BMSR\_JCD  
     MII/RMII Header, 255

BMSR\_LSTATUS  
     MII/RMII Header, 255

BMSR\_MFPRESUPPCAP  
     MII/RMII Header, 255

BMSR\_RESV  
     MII/RMII Header, 255

BMSR\_RFAULT  
     MII/RMII Header, 255

BQ\_BUFFER\_SIZE  
     I/O Buffers Queues, 99

BaseAsynchronousChannel, 515

vmt, 517

BaseAsynchronousChannelVMT, 517  
BaseBlockDevice, 519  
    vmt, 521  
BaseBlockDeviceVMT, 521  
BaseChannel, 522  
    vmt, 524  
BaseChannelVMT, 524  
BaseSequentialStream, 526  
    vmt, 527  
BaseSequentialStreamVMT, 527  
bcounter  
    io\_buffers\_queue, 558  
blk\_num  
    BlockDeviceInfo, 529  
blk\_size  
    BlockDeviceInfo, 529  
blkConnect  
    Abstract I/O Block Device, 133  
blkDisconnect  
    Abstract I/O Block Device, 134  
blkGetDriverState  
    Abstract I/O Block Device, 132  
blkGetInfo  
    Abstract I/O Block Device, 135  
blkIsInserted  
    Abstract I/O Block Device, 133  
blkIsTransferring  
    Abstract I/O Block Device, 132  
blkIsWriteProtected  
    Abstract I/O Block Device, 133  
blkRead  
    Abstract I/O Block Device, 134  
blkSync  
    Abstract I/O Block Device, 135  
blkWrite  
    Abstract I/O Block Device, 135  
blkstate\_t  
    Abstract I/O Block Device, 136  
BlockDeviceInfo, 528  
    blk\_num, 529  
    blk\_size, 529  
bn  
    io\_buffers\_queue, 558  
bqGetLinkX  
    I/O Buffers Queues, 100  
bqSizeX  
    I/O Buffers Queues, 100  
bqSpaceI  
    I/O Buffers Queues, 100  
bqnotify\_t  
    I/O Buffers Queues, 102  
brdptr  
    io\_buffers\_queue, 558  
bsize  
    io\_buffers\_queue, 558  
btop  
    io\_buffers\_queue, 558  
buffers  
    io\_buffers\_queue, 558  
bulk\_in  
    SerialUSBConfig, 592  
bulk\_out  
    SerialUSBConfig, 592  
bus\_width  
    SDCConfig, 584  
bwrptr  
    io\_buffers\_queue, 558  
CAN Driver, 35  
    CAN\_ANY\_MAILBOX, 38  
    CAN\_BUS\_OFF\_ERROR, 38  
    CAN\_FRAMING\_ERROR, 38  
    CAN\_LIMIT\_ERROR, 37  
    CAN\_LIMIT\_WARNING, 37  
    CAN\_MAILBOX\_TO\_MASK, 38  
    CAN\_OVERFLOW\_ERROR, 38  
    CAN\_READY, 39  
    CAN\_RX\_MAILBOXES, 38  
    CAN\_SLEEP, 39  
    CAN\_STARTING, 39  
    CAN\_STOP, 39  
    CAN\_TX\_MAILBOXES, 38  
    CAN\_UNINIT, 39  
    CAN\_USE\_SLEEP\_MODE, 38  
    CAND1, 50  
    can\_lld\_init, 47  
    can\_lld\_is\_rx\_nonempty, 48  
    can\_lld\_is\_tx\_empty, 48  
    can\_lld\_receive, 49  
    can\_lld\_sleep, 49  
    can\_lld\_start, 47  
    can\_lld\_stop, 48  
    can\_lld\_transmit, 48  
    can\_lld\_wakeup, 49  
    canInit, 39  
    canObjectInit, 39  
    canReceive, 44  
    canSleep, 45  
    canStart, 40  
    canStop, 40  
    canTransmit, 43  
    canTryReceiveI, 42  
    canTryTransmitI, 41  
    canWakeup, 46  
    canmbx\_t, 39  
    canstate\_t, 39  
        PLATFORM\_CAN\_USE\_CAN1, 38  
CAN\_ANY\_MAILBOX  
    CAN Driver, 38  
CAN\_BUS\_OFF\_ERROR  
    CAN Driver, 38  
CAN\_FRAMING\_ERROR  
    CAN Driver, 38  
CAN\_LIMIT\_ERROR  
    CAN Driver, 37  
CAN\_LIMIT\_WARNING  
    CAN Driver, 37

CAN\_MAILBOX\_TO\_MASK  
   CAN Driver, 38

CAN\_OVERFLOW\_ERROR  
   CAN Driver, 38

CAN\_READY  
   CAN Driver, 39

CAN\_RX\_MAILBOXES  
   CAN Driver, 38

CAN\_SLEEP  
   CAN Driver, 39

CAN\_STARTING  
   CAN Driver, 39

CAN\_STOP  
   CAN Driver, 39

CAN\_TX\_MAILBOXES  
   CAN Driver, 38

CAN\_UNINIT  
   CAN Driver, 39

CAN\_USE\_SLEEP\_MODE  
   CAN Driver, 38  
   Configuration, 219

CANConfig, 529

CAND1  
   CAN Driver, 50

CANDriver, 530  
   config, 531  
   error\_event, 532  
   rxfull\_event, 531  
   rxqueue, 531  
   sleep\_event, 532  
   state, 531  
   txempty\_event, 531  
   txqueue, 531  
   wakeup\_event, 532

CANRxFrame, 532  
   DLC, 533  
   data16, 534  
   data32, 534  
   data8, 534  
   EID, 534  
   FMI, 533  
   IDE, 533  
   RTR, 533  
   SID, 534  
   TIME, 533

CANTxFrame, 534  
   DLC, 535  
   data16, 536  
   data32, 536  
   data8, 535  
   EID, 535  
   IDE, 535  
   RTR, 535  
   SID, 535

CH\_HAL\_MAJOR  
   HAL Driver, 96

CH\_HAL\_MINOR  
   HAL Driver, 96

CH\_HAL\_PATCH  
   HAL Driver, 96

CH\_HAL\_STABLE  
   HAL Driver, 96

CHN\_CONNECTED  
   Abstract I/O Channel, 122

CHN\_DISCONNECTED  
   Abstract I/O Channel, 122

CHN\_INPUT\_AVAILABLE  
   Abstract I/O Channel, 122

CHN\_NO\_ERROR  
   Abstract I/O Channel, 122

CHN\_OUTPUT\_EMPTY  
   Abstract I/O Channel, 122

CHN\_TRANSMISSION\_END  
   Abstract I/O Channel, 122

callback  
   GPTConfig, 548  
   PWMChannelConfig, 575  
   PWMConfig, 577

can.c, 623

can.h, 623

can\_lld.c, 624

can\_lld.h, 625

can\_lld\_init  
   CAN Driver, 47

can\_lld\_is\_rx\_nonempty  
   CAN Driver, 48

can\_lld\_is\_tx\_empty  
   CAN Driver, 48

can\_lld\_receive  
   CAN Driver, 49

can\_lld\_sleep  
   CAN Driver, 49

can\_lld\_start  
   CAN Driver, 47

can\_lld\_stop  
   CAN Driver, 48

can\_lld\_transmit  
   CAN Driver, 48

can\_lld\_wakeup  
   CAN Driver, 49

canInit  
   CAN Driver, 39

canObjectInit  
   CAN Driver, 39

canReceive  
   CAN Driver, 44

canSleep  
   CAN Driver, 45

canStart  
   CAN Driver, 40

canStop  
   CAN Driver, 40

canTransmit  
   CAN Driver, 43

canTryReceive1  
   CAN Driver, 42

canTryTransmitl  
    CAN Driver, 41

canWakeup  
    CAN Driver, 46

canmbx\_t  
    CAN Driver, 39

canstate\_t  
    CAN Driver, 39

cardmode  
    SDCDriver, 586

cb  
    EXTChannelConfig, 542  
    event\_source, 541

cdc\_linecoding\_t, 536

channels  
    EXTConfig, 543  
    PWMConfig, 577  
    PWMDriver, 579

chnAddFlagsI  
    Abstract I/O Channel, 123

chnGetEventSource  
    Abstract I/O Channel, 123

chnGetTimeout  
    Abstract I/O Channel, 120

chnPutTimeout  
    Abstract I/O Channel, 119

chnRead  
    Abstract I/O Channel, 121

chnReadTimeout  
    Abstract I/O Channel, 121

chnWrite  
    Abstract I/O Channel, 120

chnWriteTimeout  
    Abstract I/O Channel, 121

circular  
    ADCConversionGroup, 513

Complex Drivers, 223

config  
    ADCDriver, 515  
    CANDriver, 531  
    DACDriver, 540  
    EXTDriver, 544  
    GPTDriver, 549  
    I2CDriver, 551  
    I2SDriver, 554  
    ICUDriver, 556  
    MACDriver, 563  
    MMCDriver, 568  
    PWMDriver, 579  
    SDCDriver, 585  
    SPIDriver, 597  
    UARTDriver, 602  
    USBDriver, 610  
    WDGDriver, 617

Configuration, 215  
    ADC\_USE\_MUTUAL\_EXCLUSION, 219  
    ADC\_USE\_WAIT, 219  
    CAN\_USE\_SLEEP\_MODE, 219

HAL\_USE\_ADC, 217

HAL\_USE\_CAN, 217

HAL\_USE\_DAC, 217

HAL\_USE\_EXT, 217

HAL\_USE\_GPT, 217

HAL\_USE\_I2C, 218

HAL\_USE\_I2S, 218

HAL\_USE\_ICU, 218

HAL\_USE\_MAC, 218

HAL\_USE\_MMCSPI, 218

HAL\_USE\_PAL, 217

HAL\_USE\_PWM, 218

HAL\_USE\_RTC, 218

HAL\_USE\_SDC, 218

HAL\_USE\_SERIAL, 218

HAL\_USE\_SERIAL\_USB, 218

HAL\_USE\_SPI, 218

HAL\_USE\_UART, 218

HAL\_USE\_USB, 219

HAL\_USE\_WDG, 219

I2C\_USE\_MUTUAL\_EXCLUSION, 219

MAC\_USE\_EVENTS, 219

MAC\_USE\_ZERO\_COPY, 219

MMC\_NICE\_WAITING, 219

SDC\_INIT\_RETRY, 219

SDC\_MMCSUPPORT, 220

SDC\_NICE\_WAITING, 220

SERIAL\_BUFFERS\_SIZE, 220

SERIAL\_DEFAULT\_BITRATE, 220

SERIAL\_USB\_BUFFERS\_NUMBER, 220

SERIAL\_USB\_BUFFERS\_SIZE, 220

SPI\_USE\_MUTUAL\_EXCLUSION, 221

SPI\_USE\_WAIT, 220

UART\_USE\_MUTUAL\_EXCLUSION, 221

UART\_USE\_WAIT, 221

USB\_USE\_WAIT, 221

configuration  
    USBDriver, 611

crc7  
    MMC over SPI Driver, 263

crc7\_lookup\_table  
    MMC over SPI Driver, 280

DAC Driver, 51  
    \_dac\_isr\_error\_code, 56  
    \_dac\_isr\_full\_code, 55  
    \_dac\_isr\_half\_code, 55  
    \_dac\_reset\_i, 54  
    \_dac\_reset\_s, 54  
    \_dac\_timeout\_isr, 55  
    \_dac\_wait\_s, 53  
    \_dac\_wakeup\_isr, 54  
    DAC\_ACTIVE, 58  
    DAC\_COMPLETE, 58  
    DAC\_ERR\_DMAFAILURE, 58  
    DAC\_ERR\_UNDERFLOW, 58  
    DAC\_ERROR, 58  
    DAC\_MAX\_CHANNELS, 57  
    DAC\_READY, 58

DAC\_STOP, 58  
 DAC\_UNINIT, 58  
 DAC\_USE\_MUTUAL\_EXCLUSION, 53  
 DAC\_USE\_WAIT, 53  
 DACD1, 67  
 DACDriver, 57  
 dac\_lld\_init, 65  
 dac\_lld\_put\_channel, 66  
 dac\_lld\_start, 65  
 dac\_lld\_start\_conversion, 66  
 dac\_lld\_stop, 66  
 dac\_lld\_stop\_conversion, 66  
 dacAcquireBus, 64  
 dacConvert, 63  
 dacInit, 58  
 dacObjectInit, 58  
 dacPutChannelX, 60  
 dacReleaseBus, 64  
 dacStart, 59  
 dacStartConversion, 60  
 dacStartConversionl, 61  
 dacStop, 59  
 dacStopConversion, 62  
 dacStopConversionl, 62  
 daccallback\_t, 57  
 dacchannel\_t, 57  
 dacerror\_t, 58  
 dacerrorcallback\_t, 57  
 dacsample\_t, 57  
 dacstate\_t, 58  
 PLATFORM\_DAC\_USE\_DAC1, 57  
**DAC\_ACTIVE**  
 DAC Driver, 58  
**DAC\_COMPLETE**  
 DAC Driver, 58  
**DAC\_ERR\_DMAFAILURE**  
 DAC Driver, 58  
**DAC\_ERR\_UNDERFLOW**  
 DAC Driver, 58  
**DAC\_ERROR**  
 DAC Driver, 58  
**DAC\_MAX\_CHANNELS**  
 DAC Driver, 57  
**DAC\_READY**  
 DAC Driver, 58  
**DAC\_STOP**  
 DAC Driver, 58  
**DAC\_UNINIT**  
 DAC Driver, 58  
**DAC\_USE\_MUTUAL\_EXCLUSION**  
 DAC Driver, 53  
**DAC\_USE\_WAIT**  
 DAC Driver, 53  
**DACConfig**, 536  
**DACConversionGroup**, 537  
 end\_cb, 538  
 error\_cb, 538  
 num\_channels, 538  
 DACD1  
 DAC Driver, 67  
**DACDriver**, 538  
 config, 540  
 DAC Driver, 57  
 depth, 540  
 grpp, 540  
 mutex, 540  
 samples, 540  
 state, 540  
 thread, 540  
**DLC**  
 CANRxFrame, 533  
 CANTxFrame, 535  
**dac.c**, 626  
**dac.h**, 627  
**dac\_lld.c**, 629  
**dac\_lld.h**, 629  
**dac\_lld\_init**  
 DAC Driver, 65  
**dac\_lld\_put\_channel**  
 DAC Driver, 66  
**dac\_lld\_start**  
 DAC Driver, 65  
**dac\_lld\_start\_conversion**  
 DAC Driver, 66  
**dac\_lld\_stop**  
 DAC Driver, 66  
**dac\_lld\_stop\_conversion**  
 DAC Driver, 66  
**dacAcquireBus**  
 DAC Driver, 64  
**dacConvert**  
 DAC Driver, 63  
**dacInit**  
 DAC Driver, 58  
**dacObjectInit**  
 DAC Driver, 58  
**dacPutChannelX**  
 DAC Driver, 60  
**dacReleaseBus**  
 DAC Driver, 64  
**dacStart**  
 DAC Driver, 59  
**dacStartConversion**  
 DAC Driver, 60  
**dacStartConversionl**  
 DAC Driver, 61  
**dacStop**  
 DAC Driver, 59  
**dacStopConversion**  
 DAC Driver, 62  
**dacStopConversionl**  
 DAC Driver, 62  
**daccallback\_t**  
 DAC Driver, 57  
**dacchannel\_t**  
 DAC Driver, 57

dacerror\_t  
    DAC Driver, 58

dacerrorcallback\_t  
    DAC Driver, 57

dacsample\_t  
    DAC Driver, 57

dacstate\_t  
    DAC Driver, 58

data16  
    CANRxFrame, 534  
    CANTxFrame, 536

data32  
    CANRxFrame, 534  
    CANTxFrame, 536

data8  
    CANRxFrame, 534  
    CANTxFrame, 535

day  
    RTCDateTime, 581

dayofweek  
    RTCDateTime, 580

default\_config  
    Serial Driver, 392

default\_handler  
    USB Driver, 482

depth  
    ADCDriver, 515  
    DACDriver, 540

detect\_bus\_clk  
    SDC Driver, 361

dstflag  
    RTCDateTime, 580

EID  
    CANRxFrame, 534  
    CANTxFrame, 535

EP\_STATUS\_ACTIVE  
    USB Driver, 480

EP\_STATUS\_DISABLED  
    USB Driver, 480

EP\_STATUS\_STALLED  
    USB Driver, 480

ETHD1  
    MAC Driver, 211

EXPANSION\_ENABLENPAGE  
    MII/RMII Header, 258

EXPANSION\_LCWP  
    MII/RMII Header, 258

EXPANSION\_MFAULTS  
    MII/RMII Header, 258

EXPANSION\_NPCAPABLE  
    MII/RMII Header, 258

EXPANSION\_NWAY  
    MII/RMII Header, 258

EXPANSION\_RESV  
    MII/RMII Header, 258

EXT Driver, 68  
    EXT\_ACTIVE, 72  
    EXT\_CH\_MODE\_AUTOSTART, 71

EXT\_CH\_MODE\_BOTH\_EDGES, 71  
EXT\_CH\_MODE\_DISABLED, 70  
EXT\_CH\_MODE\_EDGES\_MASK, 70  
EXT\_CH\_MODE\_FALLING\_EDGE, 70  
EXT\_CH\_MODE\_RISING\_EDGE, 70  
EXT\_MAX\_CHANNELS, 72  
EXT\_STOP, 72  
EXT\_UNINIT, 72

EXTD1, 77  
EXTDriver, 72  
expchannel\_t, 72  
ext\_lld\_channel\_disable, 77  
ext\_lld\_channel\_enable, 77  
ext\_lld\_init, 76  
ext\_lld\_start, 76  
ext\_lld\_stop, 77  
extChannelDisable, 75  
extChannelDisablel, 71  
extChannelEnable, 74  
extChannelEnablel, 71  
extInit, 73  
extObjectInit, 73  
extSetChannelMode, 71  
extSetChannelModel, 75  
extStart, 73  
extStop, 74  
extcallback\_t, 72  
extstate\_t, 72  
PLATFORM\_EXT\_USE\_EXT1, 72

EXT\_ACTIVE  
    EXT Driver, 72

EXT\_CH\_MODE\_AUTOSTART  
    EXT Driver, 71

EXT\_CH\_MODE\_BOTH\_EDGES  
    EXT Driver, 71

EXT\_CH\_MODE\_DISABLED  
    EXT Driver, 70

EXT\_CH\_MODE\_EDGES\_MASK  
    EXT Driver, 70

EXT\_CH\_MODE\_FALLING\_EDGE  
    EXT Driver, 70

EXT\_CH\_MODE\_RISING\_EDGE  
    EXT Driver, 70

EXT\_MAX\_CHANNELS  
    EXT Driver, 72

EXT\_STOP  
    EXT Driver, 72

EXT\_UNINIT  
    EXT Driver, 72

EXTChannelConfig, 541  
    cb, 542  
    mode, 542

EXTConfig, 543  
    channels, 543

EXTD1  
    EXT Driver, 77

EXTDriver, 544  
    config, 544

EXT Driver, 72  
 state, 544  
 early  
     UARTDriver, 602  
 enabled  
     PWMDriver, 579  
 end\_cb  
     ADCConversionGroup, 513  
     DACConversionGroup, 538  
     I2SConfig, 553  
     SPIConfig, 596  
 ep0\_state  
     USB Driver, 502  
 ep0config  
     USB Driver, 502  
 ep0endcb  
     USBDriver, 611  
 ep0n  
     USBDriver, 611  
 ep0next  
     USBDriver, 611  
 ep0state  
     USBDriver, 611  
 ep\_mode  
     USBEndpointConfig, 613  
 epc  
     USBDriver, 610  
 error\_cb  
     ADCConversionGroup, 513  
     DACConversionGroup, 538  
 error\_event  
     CANDriver, 532  
 errors  
     I2CDriver, 551  
     SDCDriver, 586  
 event\_cb  
     USBConfig, 608  
 event\_source, 540  
     cb, 541  
     flags, 541  
     param, 541  
 event\_source\_t  
     OSAL, 238  
 eventcallback\_t  
     OSAL, 238  
 eventflags\_t  
     OSAL, 238  
 expchannel\_t  
     EXT Driver, 72  
 ext.c, 631  
 ext.h, 631  
 ext\_lld.c, 632  
 ext\_lld.h, 633  
 ext\_lld\_channel\_disable  
     EXT Driver, 77  
 ext\_lld\_channel\_enable  
     EXT Driver, 77  
 ext\_lld\_init  
     EXT Driver, 76  
 ext\_lld\_start  
     EXT Driver, 76  
 ext\_lld\_stop  
     EXT Driver, 77  
 extChannelDisable  
     EXT Driver, 75  
 extChannelDisableI  
     EXT Driver, 71  
 extChannelEnable  
     EXT Driver, 74  
 extChannelEnableI  
     EXT Driver, 71  
 extInit  
     EXT Driver, 73  
 extObjectInit  
     EXT Driver, 73  
 extSetChannelMode  
     EXT Driver, 71  
 extSetChannelModel  
     EXT Driver, 75  
 extStart  
     EXT Driver, 73  
 extStop  
     EXT Driver, 74  
 extcallback\_t  
     EXT Driver, 72  
 extstate\_t  
     EXT Driver, 72  
 FILE\_EOF  
     Abstract Files, 125  
 FILE\_ERROR  
     Abstract Files, 125  
 FILE\_OK  
     Abstract Files, 125  
 FMI  
     CANRxFrame, 533  
 FileStream, 545  
     vmt, 546  
 fileStreamClose  
     Abstract Files, 127  
 fileStreamGet  
     Abstract Files, 127  
 fileStreamGetPosition  
     Abstract Files, 128  
 fileStreamGetSize  
     Abstract Files, 128  
 fileStreamPut  
     Abstract Files, 126  
 fileStreamRead  
     Abstract Files, 126  
 fileStreamSeek  
     Abstract Files, 129  
 FileStreamVMT, 546  
 fileStreamWrite  
     Abstract Files, 125

fileoffset\_t  
    Abstract Files, 129

flags  
    event\_source, 541

frequency  
    GPTConfig, 548  
    ICUConfig, 555  
    PWMConfig, 577

GPT Driver, 78  
    GPT\_CONTINUOUS, 84  
    GPT\_ONESHOT, 84  
    GPT\_READY, 84  
    GPT\_STOP, 84  
    GPT\_UNINIT, 84  
    GPTD1, 94  
    GPTDriver, 84  
    gpt\_lld\_change\_interval, 83  
    gpt\_lld\_init, 92  
    gpt\_lld\_polled\_delay, 94  
    gpt\_lld\_start, 93  
    gpt\_lld\_start\_timer, 93  
    gpt\_lld\_stop, 93  
    gpt\_lld\_stop\_timer, 93  
    gptChangelInterval, 87  
    gptChangelIntervall, 81  
    gptGetCounterX, 81  
    gptGetIntervalX, 81  
    gptInit, 84  
    gptObjectInit, 85  
    gptPolledDelay, 92  
    gptStart, 85  
    gptStartContinuous, 88  
    gptStartContinuousl, 88  
    gptStartOneShot, 90  
    gptStartOneShotl, 90  
    gptStop, 85  
    gptStopTimer, 91  
    gptStopTimerl, 91  
    gptcallback\_t, 84  
    gptcnt\_t, 84  
    gptfreq\_t, 84  
    gptstate\_t, 84  
    PLATFORM\_GPT\_USE\_GPT1, 83

GPT\_CONTINUOUS  
    GPT Driver, 84

GPT\_ONESHOT  
    GPT Driver, 84

GPT\_READY  
    GPT Driver, 84

GPT\_STOP  
    GPT Driver, 84

GPT\_UNINIT  
    GPT Driver, 84

GPTConfig, 547  
    callback, 548  
    frequency, 548

GPTD1  
    GPT Driver, 94

GPTDriver, 549  
    config, 549  
    GPT Driver, 84  
    state, 549

get\_descriptor\_cb  
    USBConfig, 608

gpt.c, 634

gpt.h, 635

gpt\_lld.c, 636

gpt\_lld.h, 636

gpt\_lld\_change\_interval  
    GPT Driver, 83

gpt\_lld\_init  
    GPT Driver, 92

gpt\_lld\_polled\_delay  
    GPT Driver, 94

gpt\_lld\_start  
    GPT Driver, 93

gpt\_lld\_start\_timer  
    GPT Driver, 93

gpt\_lld\_stop  
    GPT Driver, 93

gpt\_lld\_stop\_timer  
    GPT Driver, 93

gptChangelInterval  
    GPT Driver, 87

gptChangelIntervall  
    GPT Driver, 81

gptGetCounterX  
    GPT Driver, 81

gptGetIntervalX  
    GPT Driver, 81

gptInit  
    GPT Driver, 84

gptObjectInit  
    GPT Driver, 85

gptPolledDelay  
    GPT Driver, 92

gptStart  
    GPT Driver, 85

gptStartContinuous  
    GPT Driver, 88

gptStartContinuousl  
    GPT Driver, 88

gptStartOneShot  
    GPT Driver, 90

gptStartOneShotl  
    GPT Driver, 90

gptStop  
    GPT Driver, 85

gptStopTimer  
    GPT Driver, 91

gptStopTimerl  
    GPT Driver, 91

gptcallback\_t  
    GPT Driver, 84

gptcnt\_t  
    GPT Driver, 84

gptfreq\_t  
     GPT Driver, 84  
 gptstate\_t  
     GPT Driver, 84  
 grpp  
     ADCDriver, 515  
     DACDriver, 540  
 HAL, 212  
 HAL Driver, 95  
     \_CHIBIOS\_HAL\_, 96  
     CH\_HAL\_MAJOR, 96  
     CH\_HAL\_MINOR, 96  
     CH\_HAL\_PATCH, 96  
     CH\_HAL\_STABLE, 96  
     HAL\_VERSION, 96  
     hal\_lld\_init, 97  
     hallinit, 96  
 HAL\_USE\_ADC  
     Configuration, 217  
 HAL\_USE\_CAN  
     Configuration, 217  
 HAL\_USE\_DAC  
     Configuration, 217  
 HAL\_USE\_EXT  
     Configuration, 217  
 HAL\_USE\_GPT  
     Configuration, 217  
 HAL\_USE\_I2C  
     Configuration, 218  
 HAL\_USE\_I2S  
     Configuration, 218  
 HAL\_USE\_ICU  
     Configuration, 218  
 HAL\_USE\_MAC  
     Configuration, 218  
 HAL\_USE\_MMC\_SPI  
     Configuration, 218  
 HAL\_USE\_PAL  
     Configuration, 217  
 HAL\_USE\_PWM  
     Configuration, 218  
 HAL\_USE\_RTC  
     Configuration, 218  
 HAL\_USE\_SDC  
     Configuration, 218  
 HAL\_USE\_SERIAL  
     Configuration, 218  
 HAL\_USE\_SERIAL\_USB  
     Configuration, 218  
 HAL\_USE\_SPI  
     Configuration, 218  
 HAL\_USE\_UART  
     Configuration, 218  
 HAL\_USE\_USB  
     Configuration, 219  
 HAL\_USE\_WDG  
     Configuration, 219  
 HAL\_VERSION  
     HAL Driver, 96  
     hal.c, 637  
     hal.h, 638  
     hal\_buffers.c, 639  
     hal\_buffers.h, 640  
     hal\_channels.h, 642  
     hal\_files.h, 643  
     hal\_ioblock.h, 644  
     hal\_lld.c, 646  
     hal\_lld.h, 646  
     hal\_lld\_init  
         HAL Driver, 97  
     hal\_mmcisd.c, 646  
     hal\_mmcisd.h, 647  
     hal\_queues.c, 651  
     hal\_queues.h, 652  
     hal\_streams.h, 653  
     hallinit  
         HAL Driver, 96  
     halconf.h, 654  
     hscfg  
         MMCConfig, 567  
 I/O Buffers Queues, 98  
     BQ\_BUFFER\_SIZE, 99  
     bqGetLinkX, 100  
     bqSizeX, 100  
     bqSpaceI, 100  
     bqnNotify\_t, 102  
     ibqGetEmptyBufferI, 104  
     ibqGetFullBufferTimeout, 105  
     ibqGetFullBufferTimeoutS, 106  
     ibqGetTimeout, 108  
     ibqlIsEmptyI, 100  
     ibqlIsFullI, 101  
     ibqObjectInit, 103  
     ibqPostFullBufferI, 104  
     ibqReadTimeout, 108  
     ibqReleaseEmptyBuffer, 107  
     ibqReleaseEmptyBufferS, 107  
     ibqResetI, 103  
     input\_buffers\_queue\_t, 103  
     io\_buffers\_queue\_t, 102  
     obqFlush, 117  
     obqGetEmptyBufferTimeout, 111  
     obqGetEmptyBufferTimeoutS, 112  
     obqGetFullBufferI, 110  
     obqlIsEmptyI, 101  
     obqlIsFullI, 102  
     obqObjectInit, 109  
     obqPostFullBuffer, 113  
     obqPostFullBufferS, 114  
     obqPutTimeout, 114  
     obqReleaseEmptyBufferI, 111  
     obqResetI, 110  
     obqTryFlushI, 116  
     obqWriteTimeout, 115  
     output\_buffers\_queue\_t, 103  
 I/O Bytes Queues, 137

input\_queue\_t, 144  
io\_queue\_t, 144  
iqGet, 141  
iqGetEmptyl, 140  
iqGetFulll, 140  
iqGetTimeout, 146  
iqIsEmptyl, 141  
iqIsFulll, 141  
iqObjectInit, 144  
iqPutl, 146  
iqReadTimeout, 147  
iqResetl, 145  
oqGetEmptyl, 142  
oqGetFulll, 142  
oqGetl, 150  
oqIsEmptyl, 143  
oqIsFulll, 143  
oqObjectInit, 148  
oqPut, 143  
oqPutTimeout, 149  
oqResetl, 149  
oqWriteTimeout, 151  
output\_queue\_t, 144  
Q\_EMPTY, 139  
Q\_FULL, 139  
Q\_OK, 138  
Q\_RESET, 139  
Q\_TIMEOUT, 138  
qGetLink, 139  
qSizeX, 139  
qSpaceI, 139  
qnotify\_t, 144  
I2C Driver, 156  
    \_i2c\_wakeup\_error\_isr, 159  
    \_i2c\_wakeup\_isr, 159  
    I2C\_ACK\_FAILURE, 159  
    I2C\_ACTIVE\_RX, 161  
    I2C\_ACTIVE\_TX, 161  
    I2C\_ARBITRATION\_LOST, 159  
    I2C\_BUS\_ERROR, 158  
    I2C\_NO\_ERROR, 158  
    I2C\_OVERRUN, 159  
    I2C\_PEC\_ERROR, 159  
    I2C\_READY, 161  
    I2C\_SMB\_ALERT, 159  
    I2C\_STOP, 161  
    I2C\_TIMEOUT, 159  
    I2C\_UNINIT, 161  
    I2C\_USE\_MUTUAL\_EXCLUSION, 159  
I2CD1, 169  
I2CDriver, 161  
i2c\_lld\_get\_errors, 160  
i2c\_lld\_init, 167  
i2c\_lld\_master\_receive\_timeout, 168  
i2c\_lld\_master\_transmit\_timeout, 168  
i2c\_lld\_start, 167  
i2c\_lld\_stop, 167  
i2cAcquireBus, 166  
    i2cGetErrors, 163  
    i2cInit, 161  
    i2cMasterReceive, 160  
    i2cMasterReceiveTimeout, 165  
    i2cMasterTransmit, 160  
    i2cMasterTransmitTimeout, 164  
    i2cObjectInit, 162  
    i2cReleaseBus, 166  
    i2cStart, 162  
    i2cStop, 163  
    i2caddr\_t, 161  
    i2cflags\_t, 161  
    i2cstate\_t, 161  
        PLATFORM\_I2C\_USE\_I2C1, 160  
I2C\_ACK\_FAILURE  
    I2C Driver, 159  
I2C\_ACTIVE\_RX  
    I2C Driver, 161  
I2C\_ACTIVE\_TX  
    I2C Driver, 161  
I2C\_ARBITRATION\_LOST  
    I2C Driver, 159  
I2C\_BUS\_ERROR  
    I2C Driver, 158  
I2C\_NO\_ERROR  
    I2C Driver, 158  
I2C\_OVERRUN  
    I2C Driver, 159  
I2C\_PEC\_ERROR  
    I2C Driver, 159  
I2C\_READY  
    I2C Driver, 161  
I2C\_SMB\_ALERT  
    I2C Driver, 159  
I2C\_STOP  
    I2C Driver, 161  
I2C\_TIMEOUT  
    I2C Driver, 159  
I2C\_UNINIT  
    I2C Driver, 161  
I2C\_USE\_MUTUAL\_EXCLUSION  
    Configuration, 219  
    I2C Driver, 159  
I2CConfig, 549  
I2CD1  
    I2C Driver, 169  
I2CDriver, 550  
    config, 551  
    errors, 551  
    I2C Driver, 161  
    state, 551  
I2S Driver, 170  
    \_i2s\_isr\_full\_code, 172  
    \_i2s\_isr\_half\_code, 172  
    I2S\_ACTIVE, 173  
    I2S\_COMPLETE, 173  
    I2S\_READY, 173  
    I2S\_STOP, 173

I2S\_UNINIT, 173  
 I2SD1, 177  
 I2SDriver, 173  
 i2s\_lld\_init, 176  
 i2s\_lld\_start, 177  
 i2sInit, 174  
 i2sObjectInit, 174  
 i2sStart, 174  
 i2sStartExchange, 175  
 i2sStartExchangel, 171  
 i2sStop, 175  
 i2sStopExchange, 176  
 i2sStopExchangel, 171  
 i2scallback\_t, 173  
 i2sstate\_t, 173  
 PLATFORM\_I2S\_USE\_I2S1, 173  
**I2S\_ACTIVE**  
 I2S Driver, 173  
**I2S\_COMPLETE**  
 I2S Driver, 173  
**I2S\_READY**  
 I2S Driver, 173  
**I2S\_STOP**  
 I2S Driver, 173  
**I2S\_UNINIT**  
 I2S Driver, 173  
**I2SConfig**, 551  
 end\_cb, 553  
 rx\_buffer, 552  
 size, 553  
 tx\_buffer, 552  
**I2SD1**  
 I2S Driver, 177  
**I2SDriver**, 553  
 config, 554  
 I2S Driver, 173  
 state, 554  
 i2c.c, 657  
 i2c.h, 657  
 i2c\_lld.c, 659  
 i2c\_lld.h, 659  
 i2c\_lld\_get\_errors  
 I2C Driver, 160  
 i2c\_lld\_init  
 I2C Driver, 167  
 i2c\_lld\_master\_receive\_timeout  
 I2C Driver, 168  
 i2c\_lld\_master\_transmit\_timeout  
 I2C Driver, 168  
 i2c\_lld\_start  
 I2C Driver, 167  
 i2c\_lld\_stop  
 I2C Driver, 167  
 i2cAcquireBus  
 I2C Driver, 166  
 i2cGetErrors  
 I2C Driver, 163  
 i2cInit  
 I2C Driver, 161  
 i2cMasterReceive  
 I2C Driver, 160  
 i2cMasterReceiveTimeout  
 I2C Driver, 165  
 i2cMasterTransmit  
 I2C Driver, 160  
 i2cMasterTransmitTimeout  
 I2C Driver, 164  
 i2cObjectInit  
 I2C Driver, 162  
 i2cReleaseBus  
 I2C Driver, 166  
 i2cStart  
 I2C Driver, 162  
 i2cStop  
 I2C Driver, 163  
 i2caddr\_t  
 I2C Driver, 161  
 i2cflags\_t  
 I2C Driver, 161  
 i2cstate\_t  
 I2C Driver, 161  
 i2s.c, 660  
 i2s.h, 661  
 i2s\_lld.c, 662  
 i2s\_lld.h, 662  
 i2s\_lld\_init  
 I2S Driver, 176  
 i2s\_lld\_start  
 I2S Driver, 177  
 i2sInit  
 I2S Driver, 174  
 i2sObjectInit  
 I2S Driver, 174  
 i2sStart  
 I2S Driver, 174  
 i2sStartExchange  
 I2S Driver, 175  
 i2sStartExchangel  
 I2S Driver, 171  
 i2sStop  
 I2S Driver, 175  
 i2sStopExchange  
 I2S Driver, 176  
 i2sStopExchangel  
 I2S Driver, 171  
 i2scallback\_t  
 I2S Driver, 173  
 i2sstate\_t  
 I2S Driver, 173  
**ICU Driver**, 178  
 \_icu\_isr\_invoke\_overflow\_cb, 184  
 \_icu\_isr\_invoke\_period\_cb, 184  
 \_icu\_isr\_invoke\_width\_cb, 183  
 ICU\_ACTIVE, 186  
 ICU\_INPUT\_ACTIVE\_HIGH, 187  
 ICU\_INPUT\_ACTIVE\_LOW, 187

ICU\_READY, 186  
ICU\_STOP, 186  
ICU\_UNINIT, 186  
ICU\_WAITING, 186  
ICUD1, 195  
ICUDriver, 186  
icu\_lld\_are\_notifications\_enabled, 185  
icu\_lld\_disable\_notifications, 195  
icu\_lld\_enable\_notifications, 195  
icu\_lld\_get\_period, 185  
icu\_lld\_get\_width, 185  
icu\_lld\_init, 193  
icu\_lld\_start, 193  
icu\_lld\_start\_capture, 194  
icu\_lld\_stop, 193  
icu\_lld\_stop\_capture, 194  
icu\_lld\_wait\_capture, 194  
icuAreNotificationsEnabledX, 182  
icuDisableNotifications, 192  
icuDisableNotificationsl, 182  
icuEnableNotifications, 191  
icuEnableNotificationsl, 182  
icuGetPeriodX, 183  
icuGetWidthX, 183  
icuInit, 187  
icuObjectInit, 187  
icuStart, 187  
icuStartCapture, 188  
icuStartCapturel, 181  
icuStop, 188  
icuStopCapture, 191  
icuStopCapturel, 181  
icuWaitCapture, 190  
icucallback\_t, 186  
icucnt\_t, 186  
icufreq\_t, 186  
icemode\_t, 186  
icustate\_t, 186  
PLATFORM\_ICU\_USE\_ICU1, 185  
**ICU\_ACTIVE**  
  ICU Driver, 186  
**ICU\_INPUT\_ACTIVE\_HIGH**  
  ICU Driver, 187  
**ICU\_INPUT\_ACTIVE\_LOW**  
  ICU Driver, 187  
**ICU\_READY**  
  ICU Driver, 186  
**ICU\_STOP**  
  ICU Driver, 186  
**ICU\_UNINIT**  
  ICU Driver, 186  
**ICU\_WAITING**  
  ICU Driver, 186  
**ICUConfig**, 554  
  frequency, 555  
  mode, 555  
  overflow\_cb, 555  
  period\_cb, 555  
  width\_cb, 555  
**ICUD1**  
  ICU Driver, 195  
**ICUDriver**, 555  
  config, 556  
  ICU Driver, 186  
  state, 556  
**IDE**  
  CANRxFrame, 533  
  CANTxFrame, 535  
**IOBUS\_DECL**  
  PAL Driver, 298  
**IOBus**, 561  
  mask, 561  
  offset, 561  
  portid, 561  
**IOPORT1**  
  PAL Driver, 307  
**ibnotify**  
  Serial over USB Driver, 397  
**ibqGetEmptyBufferl**  
  I/O Buffers Queues, 104  
**ibqGetFullBufferTimeout**  
  I/O Buffers Queues, 105  
**ibqGetFullBufferTimeoutS**  
  I/O Buffers Queues, 106  
**ibqGetTimeout**  
  I/O Buffers Queues, 108  
**ibqlIsEmptyl**  
  I/O Buffers Queues, 100  
**ibqlIsFulll**  
  I/O Buffers Queues, 101  
**ibqObjectInit**  
  I/O Buffers Queues, 103  
**ibqPostFullBufferl**  
  I/O Buffers Queues, 104  
**ibqReadTimeout**  
  I/O Buffers Queues, 108  
**ibqReleaseEmptyBuffer**  
  I/O Buffers Queues, 107  
**ibqReleaseEmptyBufferS**  
  I/O Buffers Queues, 107  
**ibqResetl**  
  I/O Buffers Queues, 103  
**icu.c**, 663  
**icu.h**, 664  
**icu\_lld.c**, 665  
**icu\_lld.h**, 666  
**icu\_lld\_are\_notifications\_enabled**  
  ICU Driver, 185  
**icu\_lld\_disable\_notifications**  
  ICU Driver, 195  
**icu\_lld\_enable\_notifications**  
  ICU Driver, 195  
**icu\_lld\_get\_period**  
  ICU Driver, 185  
**icu\_lld\_get\_width**  
  ICU Driver, 185

**icu\_lld\_init**  
 ICU Driver, 193  
**icu\_lld\_start**  
 ICU Driver, 193  
**icu\_lld\_start\_capture**  
 ICU Driver, 194  
**icu\_lld\_stop**  
 ICU Driver, 193  
**icu\_lld\_stop\_capture**  
 ICU Driver, 194  
**icu\_lld\_wait\_capture**  
 ICU Driver, 194  
**icuAreNotificationsEnabledX**  
 ICU Driver, 182  
**icuDisableNotifications**  
 ICU Driver, 192  
**icuDisableNotificationsI**  
 ICU Driver, 182  
**icuEnableNotifications**  
 ICU Driver, 191  
**icuEnableNotificationsI**  
 ICU Driver, 182  
**icuGetPeriodX**  
 ICU Driver, 183  
**icuGetWidthX**  
 ICU Driver, 183  
**iculInit**  
 ICU Driver, 187  
**icuObjectInit**  
 ICU Driver, 187  
**icuStart**  
 ICU Driver, 187  
**icuStartCapture**  
 ICU Driver, 188  
**icuStartCaptureI**  
 ICU Driver, 181  
**icuStop**  
 ICU Driver, 188  
**icuStopCapture**  
 ICU Driver, 191  
**icuStopCaptureI**  
 ICU Driver, 181  
**icuWaitCapture**  
 ICU Driver, 190  
**icucallback\_t**  
 ICU Driver, 186  
**icucnt\_t**  
 ICU Driver, 186  
**icufreq\_t**  
 ICU Driver, 186  
**icumode\_t**  
 ICU Driver, 186  
**icustate\_t**  
 ICU Driver, 186  
**in**  
 USB Driver, 502  
 usb\_lld.c, 723  
**in\_cb**

USBEndpointConfig, 613  
**in\_maxsize**  
 USBEndpointConfig, 614  
**in\_params**  
 USBDriver, 611  
**in\_state**  
 USBEndpointConfig, 614  
**Inner Code**, 225  
**input\_buffers\_queue\_t**  
 I/O Buffers Queues, 103  
**input\_queue\_t**  
 I/O Bytes Queues, 144  
**int\_in**  
 SerialUSBConfig, 592  
**Interfaces**, 224  
**io\_buffers\_queue**, 556  
 bcounter, 558  
 bn, 558  
 brptr, 558  
 bsize, 558  
 btop, 558  
 buffers, 558  
 bwrptr, 558  
 link, 559  
 notify, 559  
 ptr, 558  
 top, 558  
 waiting, 558  
**io\_buffers\_queue\_t**  
 I/O Buffers Queues, 102  
**io\_queue**, 559  
 q\_buffer, 560  
 q\_counter, 560  
 q\_link, 560  
 q\_notify, 560  
 q\_rptr, 560  
 q\_top, 560  
 q\_waiting, 560  
 q\_wptr, 560  
**io\_queue\_t**  
 I/O Bytes Queues, 144  
**ioline\_t**  
 PAL Driver, 314  
**iomode\_t**  
 PAL Driver, 314  
**ioportid\_t**  
 PAL Driver, 314  
**ioportmask\_t**  
 PAL Driver, 314  
**iqGet**  
 I/O Bytes Queues, 141  
**iqGetEmptyI**  
 I/O Bytes Queues, 140  
**iqGetFullI**  
 I/O Bytes Queues, 140  
**iqGetTimeout**  
 I/O Bytes Queues, 146  
**iqIsEmptyI**

I/O Bytes Queues, 141  
iqIsFull  
    I/O Bytes Queues, 141  
iqObjectInit  
    I/O Bytes Queues, 144  
iqPutl  
    I/O Bytes Queues, 146  
iqReadTimeout  
    I/O Bytes Queues, 147  
iqResetl  
    I/O Bytes Queues, 145

LPA\_100BASE4  
    MII/RMII Header, 257  
LPA\_100FULL  
    MII/RMII Header, 257  
LPA\_100HALF  
    MII/RMII Header, 257  
LPA\_10FULL  
    MII/RMII Header, 257  
LPA\_10HALF  
    MII/RMII Header, 257  
LPA\_LPACK  
    MII/RMII Header, 258  
LPA\_NPAGE  
    MII/RMII Header, 258  
LPA\_PAUSE\_ASYM  
    MII/RMII Header, 258  
LPA\_PAUSE\_CAP  
    MII/RMII Header, 257  
LPA\_RESV  
    MII/RMII Header, 258  
LPA\_RFAULT  
    MII/RMII Header, 258  
LPA\_SLCT  
    MII/RMII Header, 257

link  
    io\_buffers\_queue, 559

lscfg  
    MMCConfig, 566

MAC Driver, 196  
    ETHD1, 211  
    MAC\_ACTIVE, 200  
    MAC\_STOP, 200  
    MAC\_SUPPORTS\_ZERO\_COPY, 200  
    MAC\_UNINIT, 200  
    MAC\_USE\_EVENTS, 198  
    MAC\_USE\_ZERO\_COPY, 198  
    MACDriver, 200  
        mac\_lld\_get\_next\_receive\_buffer, 211  
        mac\_lld\_get\_next\_transmit\_buffer, 210  
        mac\_lld\_get\_receive\_descriptor, 209  
        mac\_lld\_get\_transmit\_descriptor, 208  
        mac\_lld\_init, 207  
        mac\_lld\_poll\_link\_status, 209  
        mac\_lld\_read\_receive\_descriptor, 210  
        mac\_lld\_release\_receive\_descriptor, 209  
        mac\_lld\_release\_transmit\_descriptor, 209

mac\_lld\_start, 208  
mac\_lld\_stop, 208  
mac\_lld\_write\_transmit\_descriptor, 210  
macGetNextReceiveBuffer, 199  
macGetNextTransmitBuffer, 199  
macGetReceiveEventSource, 198  
macInit, 201  
macObjectInit, 201  
macPollLinkStatus, 207  
macReadReceiveDescriptor, 199  
macReleaseReceiveDescriptor, 206  
macReleaseTransmitDescriptor, 203  
macStart, 201  
macStop, 202  
macWaitReceiveDescriptor, 205  
macWaitTransmitDescriptor, 202  
macWriteTransmitDescriptor, 198  
macstate\_t, 200  
PLATFORM\_MAC\_USE\_MAC1, 200

MAC\_ACTIVE  
    MAC Driver, 200

MAC\_STOP  
    MAC Driver, 200

MAC\_SUPPORTS\_ZERO\_COPY  
    MAC Driver, 200

MAC\_UNINIT  
    MAC Driver, 200

MAC\_USE\_EVENTS  
    Configuration, 219  
    MAC Driver, 198

MAC\_USE\_ZERO\_COPY  
    Configuration, 219  
    MAC Driver, 198

MACConfig, 562  
    mac\_address, 562

MACDriver, 562  
    config, 563  
    MAC Driver, 200  
    rdevent, 564  
    rdqueue, 563  
    state, 563  
    tdqueue, 563

MACReceiveDescriptor, 564  
    offset, 564  
    size, 564

MACTransmitDescriptor, 565  
    offset, 565  
    size, 565

MII/RMII Header, 250  
    ADVERTISE\_100BASE4, 256  
    ADVERTISE\_100FULL, 256  
    ADVERTISE\_100HALF, 256  
    ADVERTISE\_10FULL, 256  
    ADVERTISE\_10HALF, 256  
    ADVERTISE\_CSMA, 256  
    ADVERTISE\_LPACK, 257  
    ADVERTISE\_NPAGE, 257  
    ADVERTISE\_PAUSE\_ASYM, 257

ADVERTISE\_PAUSE\_CAP, 256  
 ADVERTISE\_RESV, 257  
 ADVERTISE\_RFAULT, 257  
 ADVERTISE\_SLCT, 256  
 BMCR\_ANENABLE, 254  
 BMCR\_ANRESTART, 254  
 BMCR\_CTST, 254  
 BMCR\_FULLDPLX, 254  
 BMCR\_ISOLATE, 254  
 BMCR\_LOOPBACK, 255  
 BMCR\_PDOWN, 254  
 BMCR\_RESET, 255  
 BMCR\_RESV, 254  
 BMCR\_SPEED100, 255  
 BMSR\_100BASE4, 256  
 BMSR\_100FULL, 256  
 BMSR\_100HALF, 256  
 BMSR\_10FULL, 256  
 BMSR\_10HALF, 255  
 BMSR\_ANEGCAPABLE, 255  
 BMSR\_ANEGCOMPLETE, 255  
 BMSR\_ERCAP, 255  
 BMSR\_JCD, 255  
 BMSR\_LSTATUS, 255  
 BMSR\_MFPRESUPPCAP, 255  
 BMSR\_RESV, 255  
 BMSR\_RFAULT, 255  
 EXPANSION\_ENABLENPAGE, 258  
 EXPANSION\_LCWP, 258  
 EXPANSION\_MFAULTS, 258  
 EXPANSION\_NPCAPABLE, 258  
 EXPANSION\_NWAY, 258  
 EXPANSION\_RESV, 258  
 LPA\_100BASE4, 257  
 LPA\_100FULL, 257  
 LPA\_100HALF, 257  
 LPA\_10FULL, 257  
 LPA\_10HALF, 257  
 LPA\_LPACK, 258  
 LPA\_NPAGE, 258  
 LPA\_PAUSE\_ASYM, 258  
 LPA\_PAUSE\_CAP, 257  
 LPA\_RESV, 258  
 LPA\_RFAULT, 258  
 LPA\_SLCT, 257  
 MII\_ADVERTISE, 252  
 MII\_ANNPTR, 253  
 MII\_BMCR, 252  
 MII\_BMSR, 252  
 MII\_CTRL1000, 253  
 MII\_DCOUNT, 253  
 MII\_ESTATUS, 253  
 MII\_EXPANSION, 252  
 MII\_FCSCOUNTER, 253  
 MII\_LBRERROR, 254  
 MII\_LPA, 252  
 MII\_MICR, 253  
 MII\_NCONFIG, 252  
 MII\_NWAYTEST, 253  
 MII\_PHYADDR, 254  
 MII\_PHYSID1, 252  
 MII\_PHYSID2, 252  
 MII\_PHYSTS, 253  
 MII\_RERRCOUNTER, 253  
 MII\_RESV1, 253  
 MII\_RESV2, 254  
 MII\_SREVISION, 253  
 MII\_STAT1000, 253  
 MII\_TPISTATUS, 254  
 NWAYTEST\_LOOPBACK, 259  
 NWAYTEST\_RESV1, 258  
 NWAYTEST\_RESV2, 259

MII\_ADVERTISE  
     MII/RMII Header, 252  
 MII\_ANNPTR  
     MII/RMII Header, 253  
 MII\_BMCR  
     MII/RMII Header, 252  
 MII\_BMSR  
     MII/RMII Header, 252  
 MII\_CTRL1000  
     MII/RMII Header, 253  
 MII\_DCOUNT  
     MII/RMII Header, 253  
 MII\_ESTATUS  
     MII/RMII Header, 253  
 MII\_EXPANSION  
     MII/RMII Header, 252  
 MII\_FCSCOUNTER  
     MII/RMII Header, 253  
 MII\_LBRERROR  
     MII/RMII Header, 254  
 MII\_LPA  
     MII/RMII Header, 252  
 MII\_MICR  
     MII/RMII Header, 253  
 MII\_NCONFIG  
     MII/RMII Header, 254  
 MII\_NWAYTEST  
     MII/RMII Header, 253  
 MII\_PHYADDR  
     MII/RMII Header, 254  
 MII\_PHYSID1  
     MII/RMII Header, 252  
 MII\_PHYSID2  
     MII/RMII Header, 252  
 MII\_PHYSTS  
     MII/RMII Header, 253  
 MII\_RERRCOUNTER  
     MII/RMII Header, 253  
 MII\_RESV1  
     MII/RMII Header, 253  
 MII\_RESV2  
     MII/RMII Header, 254  
 MII\_SREVISION  
     MII/RMII Header, 253

MII\_STAT1000  
MII/RMII Header, 253

MII\_TPISTATUS  
MII/RMII Header, 254

MMC over SPI Driver, 260  
\_mmc\_driver\_methods, 262  
crc7, 263  
crc7\_lookup\_table, 280

MMC\_NICE\_WAITING, 262

mmc\_vmt, 280

mmcConnect, 270

mmcDisconnect, 271

mmcErase, 279

mmcGetInfo, 279

mmcInit, 269

mmclsCardInserted, 262

mmclsWriteProtected, 262

mmcObjectInit, 269

mmcSequentialRead, 273

mmcSequentialWrite, 276

mmcStart, 269

mmcStartSequentialRead, 272

mmcStartSequentialWrite, 275

mmcStop, 269

mmcStopSequentialRead, 273

mmcStopSequentialWrite, 276

mmcSync, 278

read\_Cxd, 267

recv1, 264

recv3, 265

send\_command\_R1, 266

send\_command\_R3, 266

send\_hdr, 264

sync, 268

wait, 263

MMC/SD Block Device, 282  
\_mmcsd\_block\_device\_data, 287  
\_mmcsd\_block\_device\_methods, 286  
\_mmcsd\_get\_capacity, 288  
\_mmcsd\_get\_capacity\_ext, 289  
\_mmcsd\_get\_slice, 288  
\_mmcsd\_unpack\_csd\_mmc, 290  
\_mmcsd\_unpack\_csd\_v10, 290  
\_mmcsd\_unpack\_csd\_v20, 291  
\_mmcsd\_unpack\_mmc\_cid, 289  
\_mmcsd\_unpack\_sdc\_cid, 289

MMCSBLOCK\_SIZE, 286

MMCSDCID\_SDC\_CRC\_SLICE, 286

MMCSDCMD8\_PATTERN, 286

MMCSDCSD\_MMCCSD\_STRUCTURE\_SLICE, 286

MMCSD\_R1\_ERROR, 287

MMCSD\_R1\_ERROR\_MASK, 286

MMCSD\_R1\_IS\_CARD\_LOCKED, 287

MMCSD\_R1\_STS, 287

mmcsdGetCardCapacity, 287

MMC\_NICE\_WAITING  
Configuration, 219

MMC over SPI Driver, 262

MMCConfig, 565  
hscfg, 567  
lscfg, 566  
spip, 566

MMCDriver, 567  
config, 568  
vmt, 568

MMCDriverVMT, 569

MMCSDBLOCK\_SIZE  
MMC/SD Block Device, 286

MMCSDCID\_SDC\_CRC\_SLICE  
MMC/SD Block Device, 286

MMCSDCMD8\_PATTERN  
MMC/SD Block Device, 286

MMCSDCSD\_MMCCSD\_STRUCTURE\_SLICE  
MMC/SD Block Device, 286

MMCSDR1\_ERROR  
MMC/SD Block Device, 287

MMCSDR1\_ERROR\_MASK  
MMC/SD Block Device, 286

MMCSDR1\_IS\_CARD\_LOCKED  
MMC/SD Block Device, 287

MMCSDR1\_STS  
MMC/SD Block Device, 287

MMCSDBlockDevice, 570  
vmt, 572

MMCSDBlockDeviceVMT, 572

mac.c, 667

mac.h, 668

mac\_address  
MACConfig, 562

mac\_lld.c, 669

mac\_lld.h, 670

mac\_lld\_get\_next\_receive\_buffer  
MAC Driver, 211

mac\_lld\_get\_next\_transmit\_buffer  
MAC Driver, 210

mac\_lld\_get\_receive\_descriptor  
MAC Driver, 209

mac\_lld\_get\_transmit\_descriptor  
MAC Driver, 208

mac\_lld\_init  
MAC Driver, 207

mac\_lld\_poll\_link\_status  
MAC Driver, 209

mac\_lld\_read\_receive\_descriptor  
MAC Driver, 210

mac\_lld\_release\_receive\_descriptor  
MAC Driver, 209

mac\_lld\_release\_transmit\_descriptor  
MAC Driver, 209

mac\_lld\_start  
MAC Driver, 208

mac\_lld\_stop  
MAC Driver, 208

mac\_lld\_write\_transmit\_descriptor  
MAC Driver, 210

macGetNextReceiveBuffer  
     MAC Driver, 199  
 macGetNextTransmitBuffer  
     MAC Driver, 199  
 macGetReceiveEventSource  
     MAC Driver, 198  
 macInit  
     MAC Driver, 201  
 macObjectInit  
     MAC Driver, 201  
 macPollLinkStatus  
     MAC Driver, 207  
 macReadReceiveDescriptor  
     MAC Driver, 199  
 macReleaseReceiveDescriptor  
     MAC Driver, 206  
 macReleaseTransmitDescriptor  
     MAC Driver, 203  
 macStart  
     MAC Driver, 201  
 macStop  
     MAC Driver, 202  
 macWaitReceiveDescriptor  
     MAC Driver, 205  
 macWaitTransmitDescriptor  
     MAC Driver, 202  
 macWriteTransmitDescriptor  
     MAC Driver, 198  
 macstate\_t  
     MAC Driver, 200  
 mask  
     IOBus, 561  
 mii.h, 671  
 millisecond  
     RTCDateTime, 581  
 mmc\_cmd6\_construct  
     SDC Driver, 358  
 mmc\_detect\_bus\_clk  
     SDC Driver, 360  
 mmc\_init  
     SDC Driver, 356  
 mmc\_set\_bus\_width  
     SDC Driver, 362  
 mmc\_spi.c, 673  
 mmc\_spi.h, 674  
 mmc\_switch\_t  
     SDC Driver, 355  
 mmc\_vmt  
     MMC over SPI Driver, 280  
 mmcConnect  
     MMC over SPI Driver, 270  
 mmcDisconnect  
     MMC over SPI Driver, 271  
 mmcErase  
     MMC over SPI Driver, 279  
 mmcGetInfo  
     MMC over SPI Driver, 279  
 mmclnIt  
     OSAL, 227  
     MMC over SPI Driver, 269  
 mmclsCardInserted  
     MMC over SPI Driver, 262  
 mmclsWriteProtected  
     MMC over SPI Driver, 262  
 mmcObjectInit  
     MMC over SPI Driver, 269  
 mmcSequentialRead  
     MMC over SPI Driver, 273  
 mmcSequentialWrite  
     MMC over SPI Driver, 276  
 mmcStart  
     MMC over SPI Driver, 269  
 mmcStartSequentialRead  
     MMC over SPI Driver, 272  
 mmcStartSequentialWrite  
     MMC over SPI Driver, 275  
 mmcStop  
     MMC over SPI Driver, 269  
 mmcStopSequentialRead  
     MMC over SPI Driver, 273  
 mmcStopSequentialWrite  
     MMC over SPI Driver, 276  
 mmcSync  
     MMC over SPI Driver, 278  
 mmcSdGetCardCapacity  
     MMC/SD Block Device, 287  
 mode  
     EXTChannelConfig, 542  
     ICUConfig, 555  
     PWMChannelConfig, 575  
 mode\_detect  
     SDC Driver, 356  
 month  
     RTCDDateTime, 580  
 msg\_t  
     OSAL, 237  
 mutex  
     ADCDriver, 515  
     DACDriver, 540  
     SPIDriver, 598  
     UARTDriver, 602  
 mutex\_t  
     OSAL, 238  
     NWAYTEST\_LOOPBACK  
         MII/RMII Header, 259  
     NWAYTEST\_RESV1  
         MII/RMII Header, 258  
     NWAYTEST\_RESV2  
         MII/RMII Header, 259  
 Normal Drivers, 222  
 notify  
     io\_buffers\_queue, 559  
 num\_channels  
     ADCConversionGroup, 513  
     DACConversionGroup, 538

event\_source\_t, 238  
eventcallback\_t, 238  
eventflags\_t, 238  
msg\_t, 237  
mutex\_t, 238  
OSAL\_DBG\_ENABLE\_ASSERTS, 232  
OSAL\_DBG\_ENABLE\_CHECKS, 232  
OSAL\_IRQ\_EPILOGUE, 234  
OSAL\_IRQ\_HANDLER, 234  
OSAL\_IRQ\_IS\_VALID\_PRIORITY, 233  
OSAL\_IRQ\_MAXIMUM\_PRIORITY, 232  
OSAL\_IRQ\_PRIORITY\_LEVELS, 232  
OSAL\_IRQ\_PROLOGUE, 234  
OSAL\_MS2RTC, 236  
OSAL\_MS2ST, 234  
OSAL\_S2RTC, 235  
OSAL\_S2ST, 234  
OSAL\_ST\_FREQUENCY, 232  
OSAL\_ST\_MODE, 232  
OSAL\_ST\_RESOLUTION, 232  
OSAL\_US2RTC, 236  
OSAL\_US2ST, 235  
osal\_halt\_msg, 248  
osalDbgAssert, 232  
osalDbgCheck, 233  
osalDbgCheckClassI, 233  
osalDbgCheckClassS, 233  
osalEventBroadcastFlags, 244  
osalEventBroadcastFlagsI, 243  
osalEventObjectInit, 248  
osalEventSetCallback, 244  
osalInit, 238  
osalMutexLock, 245  
osalMutexObjectInit, 248  
osalMutexUnlock, 245  
osalOsGetSystemTimeX, 240  
osalOsIsTimeWithinX, 247  
osalOsRescheduleS, 239  
osalOsTimerHandlerI, 239  
osalSysDisable, 245  
osalSysEnable, 245  
osalSysGetStatusAndLockX, 246  
osalSysHalt, 239  
osalSysLock, 246  
osalSysLockFromISR, 246  
osalSysPolledDelayX, 239  
osalSysRestoreStatusX, 247  
osalSysUnlock, 246  
osalSysUnlockFromISR, 246  
osalThreadDequeueAll, 243  
osalThreadDequeueNextI, 243  
osalThreadEnqueueTimeoutS, 242  
osalThreadQueueObjectInit, 248  
osalThreadResumel, 242  
osalThreadResumeS, 242  
osalThreadSleep, 240  
osalThreadSleepMicroseconds, 237  
osalThreadSleepMilliseconds, 237  
osalThreadSleepS, 240  
osalThreadSleepSeconds, 236  
osalThreadSuspendS, 241  
osalThreadSuspendTimeoutS, 241  
rtcnt\_t, 238  
syssts\_t, 237  
systime\_t, 238  
thread\_reference\_t, 238  
OSAL\_DBG\_ENABLE\_ASSERTS  
    OSAL, 232  
OSAL\_DBG\_ENABLE\_CHECKS  
    OSAL, 232  
OSAL\_IRQ\_EPILOGUE  
    OSAL, 234  
OSAL\_IRQ\_HANDLER  
    OSAL, 234  
OSAL\_IRQ\_IS\_VALID\_PRIORITY  
    OSAL, 233  
OSAL\_IRQ\_MAXIMUM\_PRIORITY  
    OSAL, 232  
OSAL\_IRQ\_PRIORITY\_LEVELS  
    OSAL, 232  
OSAL\_IRQ\_PROLOGUE  
    OSAL, 234  
OSAL\_MS2RTC  
    OSAL, 236  
OSAL\_MS2ST  
    OSAL, 234  
OSAL\_S2RTC  
    OSAL, 235  
OSAL\_S2ST  
    OSAL, 234  
OSAL\_ST\_FREQUENCY  
    OSAL, 232  
OSAL\_ST\_MODE  
    OSAL, 232  
OSAL\_ST\_RESOLUTION  
    OSAL, 232  
OSAL\_US2RTC  
    OSAL, 236  
OSAL\_US2ST  
    OSAL, 235  
obnotify  
    Serial over USB Driver, 397  
obqFlush  
    I/O Buffers Queues, 117  
obqGetEmptyBufferTimeout  
    I/O Buffers Queues, 111  
obqGetEmptyBufferTimeoutS  
    I/O Buffers Queues, 112  
obqGetFullBufferI  
    I/O Buffers Queues, 110  
obqlIsEmptyI  
    I/O Buffers Queues, 101  
obqlIsFullI  
    I/O Buffers Queues, 102  
obqObjectInit  
    I/O Buffers Queues, 109

obqPostFullBuffer  
     I/O Buffers Queues, 113  
 obqPostFullBufferS  
     I/O Buffers Queues, 114  
 obqPutTimeout  
     I/O Buffers Queues, 114  
 obqReleaseEmptyBufferI  
     I/O Buffers Queues, 111  
 obqResetI  
     I/O Buffers Queues, 110  
 obqTryFlushI  
     I/O Buffers Queues, 116  
 obqWriteTimeout  
     I/O Buffers Queues, 115  
 offset  
     IOBus, 561  
     MACReceiveDescriptor, 564  
     MACTransmitDescriptor, 565  
 oqGetEmptyI  
     I/O Bytes Queues, 142  
 oqGetFullI  
     I/O Bytes Queues, 142  
 oqGetI  
     I/O Bytes Queues, 150  
 oqIsEmptyI  
     I/O Bytes Queues, 143  
 oqIsFullI  
     I/O Bytes Queues, 143  
 oqObjectInit  
     I/O Bytes Queues, 148  
 oqPut  
     I/O Bytes Queues, 143  
 oqPutTimeout  
     I/O Bytes Queues, 149  
 oqResetI  
     I/O Bytes Queues, 149  
 oqWriteTimeout  
     I/O Bytes Queues, 151  
 osal.c, 676  
 osal.h, 677  
 osal\_halt\_msg  
     OSAL, 248  
 osalDbgAssert  
     OSAL, 232  
 osalDbgCheck  
     OSAL, 233  
 osalDbgCheckClassI  
     OSAL, 233  
 osalDbgCheckClassS  
     OSAL, 233  
 osalEventBroadcastFlags  
     OSAL, 244  
 osalEventBroadcastFlagsI  
     OSAL, 243  
 osalEventObjectInit  
     OSAL, 248  
 osalEventSetCallback  
     OSAL, 244  
 osallInit  
     OSAL, 238  
 osalMutexLock  
     OSAL, 245  
 osalMutexObjectInit  
     OSAL, 248  
 osalMutexUnlock  
     OSAL, 245  
 osalOsGetSystemTimeX  
     OSAL, 240  
 osalOsIsTimeWithinX  
     OSAL, 247  
 osalOsRescheduleS  
     OSAL, 239  
 osalOsTimerHandlerI  
     OSAL, 239  
 osalSysDisable  
     OSAL, 245  
 osalSysEnable  
     OSAL, 245  
 osalSysGetStatusAndLockX  
     OSAL, 246  
 osalSysHalt  
     OSAL, 239  
 osalSysLock  
     OSAL, 246  
 osalSysLockFromISR  
     OSAL, 246  
 osalSysPolledDelayX  
     OSAL, 239  
 osalSysRestoreStatusX  
     OSAL, 247  
 osalSysUnlock  
     OSAL, 246  
 osalSysUnlockFromISR  
     OSAL, 246  
 osalThreadDequeueAllI  
     OSAL, 243  
 osalThreadDequeueNextI  
     OSAL, 243  
 osalThreadEnqueueTimeoutS  
     OSAL, 242  
 osalThreadQueueObjectInit  
     OSAL, 248  
 osalThreadResumeI  
     OSAL, 242  
 osalThreadResumeS  
     OSAL, 242  
 osalThreadSleep  
     OSAL, 240  
 osalThreadSleepMicroseconds  
     OSAL, 237  
 osalThreadSleepMilliseconds  
     OSAL, 237  
 osalThreadSleepSeconds  
     OSAL, 240  
 osalThreadSleepSeconds  
     OSAL, 236

osalThreadSuspendS  
    OSAL, 241  
osalThreadSuspendTimeoutS  
    OSAL, 241  
out  
    USB Driver, 502  
    usb\_lld.c, 723  
out\_cb  
    USBEndpointConfig, 613  
out\_maxsize  
    USBEndpointConfig, 614  
out\_params  
    USBDriver, 611  
out\_state  
    USBEndpointConfig, 614  
output\_buffers\_queue\_t  
    I/O Buffers Queues, 103  
output\_queue\_t  
    I/O Bytes Queues, 144  
overflow\_cb  
    ICUConfig, 555  
  
PAL Driver, 292  
    \_IOBUS\_DATA, 297  
    \_pal\_lld\_init, 315  
    \_pal\_lld\_setgroupmode, 315  
    IOBUS\_DECL, 298  
    IOPORT1, 307  
    ioline\_t, 314  
    iomode\_t, 314  
    ioportid\_t, 314  
    ioportmask\_t, 314  
    PAL\_GROUP\_MASK, 297  
    PAL\_HIGH, 297  
    PAL\_IOPORTS\_WIDTH, 306  
    PAL\_LINE, 306  
    PAL\_LOW, 297  
    PAL\_MODE\_INPUT, 296  
    PAL\_MODE\_INPUT\_ANALOG, 296  
    PAL\_MODE\_INPUT\_PULLDOWN, 296  
    PAL\_MODE\_INPUT\_PULLUP, 296  
    PAL\_MODE\_OUTPUT\_OPENDRAIN, 297  
    PAL\_MODE\_OUTPUT\_PUSH\_PULL, 296  
    PAL\_MODE\_RESET, 296  
    PAL\_MODE\_UNCONNECTED, 296  
    PAL\_NOLINE, 307  
    PAL\_PAD, 307  
    PAL\_PORT, 306  
    PAL\_PORT\_BIT, 297  
    PAL\_WHOLE\_PORT, 306  
    pal\_lld\_clearpad, 312  
    pal\_lld\_clearport, 308  
    pal\_lld\_init, 307  
    pal\_lld\_readgroup, 309  
    pal\_lld\_readlatch, 307  
    pal\_lld\_readpad, 311  
    pal\_lld\_readport, 307  
    pal\_lld\_setgroupmode, 310  
    pal\_lld\_setpad, 312  
    pal\_lld\_setpadmode, 313  
    pal\_lld\_setport, 308  
    pal\_lld\_togglepad, 313  
    pal\_lld\_toggleport, 309  
    pal\_lld\_writegroup, 310  
    pal\_lld\_writepad, 311  
    pal\_lld\_writeport, 308  
    palClearLine, 305  
    palClearPad, 303  
    palClearPort, 300  
    palInit, 298  
    palReadBus, 314  
    palReadGroup, 300  
    palReadLatch, 298  
    palReadLine, 304  
    palReadPad, 302  
    palReadPort, 298  
    palSetBusMode, 315  
    palSetGroupMode, 301  
    palSetLine, 305  
    palSetLineMode, 306  
    palSetPad, 303  
    palSetPadMode, 304  
    palSetPort, 299  
    palToggleLine, 306  
    palTogglePad, 303  
    palTogglePort, 300  
    palWriteBus, 314  
    palWriteGroup, 301  
    palWriteLine, 305  
    palWritePad, 302  
    palWritePort, 299  
PAL\_GROUP\_MASK  
    PAL Driver, 297  
PAL\_HIGH  
    PAL Driver, 297  
PAL\_IOPORTS\_WIDTH  
    PAL Driver, 306  
PAL\_LINE  
    PAL Driver, 306  
PAL\_LOW  
    PAL Driver, 297  
PAL\_MODE\_INPUT  
    PAL Driver, 296  
PAL\_MODE\_INPUT\_ANALOG  
    PAL Driver, 296  
PAL\_MODE\_INPUT\_PULLDOWN  
    PAL Driver, 296  
PAL\_MODE\_OUTPUT\_OPENDRAIN  
    PAL Driver, 297  
PAL\_MODE\_OUTPUT\_PUSH\_PULL  
    PAL Driver, 296  
PAL\_MODE\_RESET  
    PAL Driver, 296  
PAL\_MODE\_UNCONNECTED  
    PAL Driver, 296

PAL\_NOLINE  
 PAL Driver, 307

PAL\_PAD  
 PAL Driver, 307

PAL\_PORT  
 PAL Driver, 306

PAL\_PORT\_BIT  
 PAL Driver, 297

PAL\_WHOLE\_PORT  
 PAL Driver, 306

PALConfig, 574

PLATFORM\_ADC\_USE\_ADC1  
 ADC Driver, 24

PLATFORM\_CAN\_USE\_CAN1  
 CAN Driver, 38

PLATFORM\_DAC\_USE\_DAC1  
 DAC Driver, 57

PLATFORM\_EXT\_USE\_EXT1  
 EXT Driver, 72

PLATFORM\_GPT\_USE\_GPT1  
 GPT Driver, 83

PLATFORM\_I2C\_USE\_I2C1  
 I2C Driver, 160

PLATFORM\_I2S\_USE\_I2S1  
 I2S Driver, 173

PLATFORM\_ICU\_USE\_ICU1  
 ICU Driver, 185

PLATFORM\_MAC\_USE\_MAC1  
 MAC Driver, 200

PLATFORM\_PWM\_USE\_PWM1  
 PWM Driver, 327

PLATFORM\_RTC\_USE\_RTC1  
 RTC Driver, 342

PLATFORM\_SDC\_USE\_SDC1  
 SDC Driver, 355

PLATFORM\_SERIAL\_USE\_USART1  
 Serial Driver, 385

PLATFORM\_SPI\_USE\_SPI1  
 SPI Driver, 413

PLATFORM\_UART\_USE\_UART1  
 UART Driver, 444

PLATFORM\_USB\_USE\_USB1  
 USB Driver, 477

PLATFORM\_WDG\_USE\_WDG1  
 WDG Driver, 506

PWM Driver, 318

PLATFORM\_PWM\_USE\_PWM1, 327

PWM\_CHANNELS, 326

PWM\_DEGREES\_TO\_WIDTH, 322

PWM\_FRACTION\_TO\_WIDTH, 321

PWM\_OUTPUT\_ACTIVE\_HIGH, 321

PWM\_OUTPUT\_ACTIVE\_LOW, 321

PWM\_OUTPUT\_DISABLED, 321

PWM\_OUTPUT\_MASK, 321

PWM\_PERCENTAGE\_TO\_WIDTH, 322

PWM\_READY, 328

PWM\_STOP, 328

PWM\_UNINIT, 328

PWMD1, 339

PWMDriver, 327

pwm\_lld\_change\_period, 327

pwm\_lld\_disable\_channel, 337

pwm\_lld\_disable\_channel\_notification, 338

pwm\_lld\_disable\_periodic\_notification, 337

pwm\_lld\_enable\_channel, 336

pwm\_lld\_enable\_channel\_notification, 338

pwm\_lld\_enable\_periodic\_notification, 337

pwm\_lld\_init, 335

pwm\_lld\_start, 336

pwm\_lld\_stop, 336

pwmChangePeriod, 330

pwmChangePeriodl, 323

pwmDisableChannel, 332

pwmDisableChannell, 324

pwmDisableChannelNotification, 335

pwmDisableChannelNotificationl, 326

pwmDisablePeriodicNotification, 333

pwmDisablePeriodicNotificationl, 325

pwmEnableChannel, 331

pwmEnableChannell, 323

pwmEnableChannelNotification, 334

pwmEnableChannelNotificationl, 326

pwmEnablePeriodicNotification, 333

pwmEnablePeriodicNotificationl, 325

pwmInit, 328

pwmIsChannelEnabledl, 325

pwmObjectInit, 329

pwmStart, 329

pwmStop, 330

pwmcallback\_t, 327

pwmchannel\_t, 328

pwmchnmsk\_t, 328

pwmcnt\_t, 328

pwmmode\_t, 328

pwmstate\_t, 328

PWM\_CHANNELS  
 PWM Driver, 326

PWM\_DEGREES\_TO\_WIDTH  
 PWM Driver, 322

PWM\_FRACTION\_TO\_WIDTH  
 PWM Driver, 321

PWM\_OUTPUT\_ACTIVE\_HIGH  
 PWM Driver, 321

PWM\_OUTPUT\_ACTIVE\_LOW  
 PWM Driver, 321

PWM\_OUTPUT\_DISABLED  
 PWM Driver, 321

PWM\_OUTPUT\_MASK  
 PWM Driver, 321

PWM\_PERCENTAGE\_TO\_WIDTH  
 PWM Driver, 322

PWM\_READY  
 PWM Driver, 328

PWM\_STOP  
 PWM Driver, 328

PWM\_UNINIT

PWM Driver, 328  
PWMChannelConfig, 574  
callback, 575  
mode, 575  
PWMConfig, 576  
callback, 577  
channels, 577  
frequency, 577  
period, 577  
PWMD1  
    PWM Driver, 339  
PWMDriver, 577  
    channels, 579  
    config, 579  
    enabled, 579  
    PWM Driver, 327  
    period, 579  
    state, 578  
pal.c, 681  
pal.h, 681  
pal\_lld.c, 683  
pal\_lld.h, 684  
pal\_lld\_clearpad  
    PAL Driver, 312  
pal\_lld\_clearport  
    PAL Driver, 308  
pal\_lld\_init  
    PAL Driver, 307  
pal\_lld\_readgroup  
    PAL Driver, 309  
pal\_lld\_readlatch  
    PAL Driver, 307  
pal\_lld\_readpad  
    PAL Driver, 311  
pal\_lld\_readport  
    PAL Driver, 307  
pal\_lld\_setgroupmode  
    PAL Driver, 310  
pal\_lld\_setpad  
    PAL Driver, 312  
pal\_lld\_setpadmode  
    PAL Driver, 313  
pal\_lld\_setport  
    PAL Driver, 308  
pal\_lld\_togglepad  
    PAL Driver, 313  
pal\_lld\_toggleport  
    PAL Driver, 309  
pal\_lld\_writegroup  
    PAL Driver, 310  
pal\_lld\_writepad  
    PAL Driver, 311  
pal\_lld\_writeport  
    PAL Driver, 308  
palClearLine  
    PAL Driver, 305  
palClearPad  
    PAL Driver, 303  
palClearPort  
    PAL Driver, 300  
palInit  
    PAL Driver, 298  
palReadBus  
    PAL Driver, 314  
palReadGroup  
    PAL Driver, 300  
palReadLatch  
    PAL Driver, 298  
palReadLine  
    PAL Driver, 304  
palReadPad  
    PAL Driver, 302  
palReadPort  
    PAL Driver, 298  
palSetBusMode  
    PAL Driver, 315  
palSetGroupMode  
    PAL Driver, 301  
palSetLine  
    PAL Driver, 305  
palSetLineMode  
    PAL Driver, 306  
palSetPad  
    PAL Driver, 303  
palSetPadMode  
    PAL Driver, 304  
palSetPort  
    PAL Driver, 299  
palToggleLine  
    PAL Driver, 306  
palTogglePad  
    PAL Driver, 303  
palTogglePort  
    PAL Driver, 300  
palWriteBus  
    PAL Driver, 314  
palWriteGroup  
    PAL Driver, 301  
palWriteLine  
    PAL Driver, 305  
palWritePad  
    PAL Driver, 302  
palWritePort  
    PAL Driver, 299  
param  
    event\_source, 541  
period  
    PWMConfig, 577  
    PWMDriver, 579  
period\_cb  
    ICUConfig, 555  
portid  
    IOBus, 561  
ptr  
    io\_buffers\_queue, 558  
pwm.c, 685

pwm.h, 686  
 pwm\_lld.c, 688  
 pwm\_lld.h, 689  
 pwm\_lld\_change\_period  
     PWM Driver, 327  
 pwm\_lld\_disable\_channel  
     PWM Driver, 337  
 pwm\_lld\_disable\_channel\_notification  
     PWM Driver, 338  
 pwm\_lld\_disable\_periodic\_notification  
     PWM Driver, 337  
 pwm\_lld\_enable\_channel  
     PWM Driver, 336  
 pwm\_lld\_enable\_channel\_notification  
     PWM Driver, 338  
 pwm\_lld\_enable\_periodic\_notification  
     PWM Driver, 337  
 pwm\_lld\_init  
     PWM Driver, 335  
 pwm\_lld\_start  
     PWM Driver, 336  
 pwm\_lld\_stop  
     PWM Driver, 336  
 pwmChangePeriod  
     PWM Driver, 330  
 pwmChangePeriodI  
     PWM Driver, 323  
 pwmDisableChannel  
     PWM Driver, 332  
 pwmDisableChannell  
     PWM Driver, 324  
 pwmDisableChannelNotification  
     PWM Driver, 335  
 pwmDisableChannelNotificationI  
     PWM Driver, 326  
 pwmDisablePeriodicNotification  
     PWM Driver, 333  
 pwmDisablePeriodicNotificationI  
     PWM Driver, 325  
 pwmEnableChannel  
     PWM Driver, 331  
 pwmEnableChannell  
     PWM Driver, 323  
 pwmEnableChannelNotification  
     PWM Driver, 334  
 pwmEnableChannelNotificationI  
     PWM Driver, 326  
 pwmEnablePeriodicNotification  
     PWM Driver, 333  
 pwmEnablePeriodicNotificationI  
     PWM Driver, 325  
 pwmlInit  
     PWM Driver, 328  
 pwmlsChannelEnabledI  
     PWM Driver, 325  
 pwmObjectInit  
     PWM Driver, 329  
 pwmStart  
     PWM Driver, 329  
 pwmStop  
     PWM Driver, 330  
 pwmcallback\_t  
     PWM Driver, 327  
 pwmchannel\_t  
     PWM Driver, 328  
 pwmchnmsk\_t  
     PWM Driver, 328  
 pwmcnt\_t  
     PWM Driver, 328  
 pwmmode\_t  
     PWM Driver, 328  
 pwmstate\_t  
     PWM Driver, 328  
 Q\_EMPTY  
     I/O Bytes Queues, 139  
 Q\_FULL  
     I/O Bytes Queues, 139  
 Q\_OK  
     I/O Bytes Queues, 138  
 Q\_RESET  
     I/O Bytes Queues, 139  
 Q\_TIMEOUT  
     I/O Bytes Queues, 138  
 q\_buffer  
     io\_queue, 560  
 q\_counter  
     io\_queue, 560  
 q\_link  
     io\_queue, 560  
 q\_notify  
     io\_queue, 560  
 q\_rptr  
     io\_queue, 560  
 q\_top  
     io\_queue, 560  
 q\_waiting  
     io\_queue, 560  
 q\_wptr  
     io\_queue, 560  
 qGetLink  
     I/O Bytes Queues, 139  
 qSizeX  
     I/O Bytes Queues, 139  
 qSpaceI  
     I/O Bytes Queues, 139  
 qnotify\_t  
     I/O Bytes Queues, 144  
 RTC Driver, 340  
      rtc\_driver\_methods, 342  
     PLATFORM\_RTC\_USE\_RTC1, 342  
     RTC\_ALARMS, 342  
     RTC\_BASE\_YEAR, 342  
     RTC\_HAS\_STORAGE, 342  
     RTC\_SUPPORTS\_CALLBACKS, 342  
     RTCD1, 349

RTCDriver, 343  
rtc\_lld\_get\_alarm, 349  
rtc\_lld\_get\_time, 348  
rtc\_lld\_init, 347  
rtc\_lld\_set\_alarm, 348  
rtc\_lld\_set\_time, 348  
rtcConvertDateTimeToFAT, 347  
rtcConvertDateTimeToStructTm, 346  
rtcConvertStructTmToDateTm, 347  
rtcGetAlarm, 345  
rtcGetTime, 344  
rtcInit, 343  
rtcObjectInit, 343  
rtcSetAlarm, 345  
rtcSetCallback, 346  
rtcSetTime, 344  
rtcalarm\_t, 343  
rtccb\_t, 343  
rtcevent\_t, 343  
RTC\_ALARMS  
    RTC Driver, 342  
RTC\_BASE\_YEAR  
    RTC Driver, 342  
RTC\_HAS\_STORAGE  
    RTC Driver, 342  
RTC\_SUPPORTS\_CALLBACKS  
    RTC Driver, 342  
RTCAlarm, 579  
RTCD1  
    RTC Driver, 349  
RTCDateTime, 579  
    day, 581  
    dayofweek, 580  
    dstflag, 580  
    millisecond, 581  
    month, 580  
    year, 580  
RTCDriver, 581  
    RTC Driver, 343  
    vmt, 582  
RTCDriverVMT, 582  
RTR  
    CANRxFrame, 533  
    CANTxFrame, 535  
rca  
    SDCDriver, 586  
rdevent  
    MACDriver, 564  
rdqueue  
    MACDriver, 563  
read\_CxD  
    MMC over SPI Driver, 267  
receiving  
    USBDriver, 610  
recv1  
    MMC over SPI Driver, 264  
recv3  
    MMC over SPI Driver, 265  
requests\_hook\_cb  
    USBConfig, 608  
rtc.c, 690  
rtc.h, 691  
rtc\_lld.c, 692  
rtc\_lld.h, 692  
rtc\_lld\_get\_alarm  
    RTC Driver, 349  
rtc\_lld\_get\_time  
    RTC Driver, 348  
rtc\_lld\_init  
    RTC Driver, 347  
rtc\_lld\_set\_alarm  
    RTC Driver, 348  
rtc\_lld\_set\_time  
    RTC Driver, 348  
rtcConvertDateTimeToFAT  
    RTC Driver, 347  
rtcConvertDateTimeToStructTm  
    RTC Driver, 346  
rtcConvertStructTmToDateTm  
    RTC Driver, 347  
rtcGetAlarm  
    RTC Driver, 345  
rtcGetTime  
    RTC Driver, 344  
rtcInit  
    RTC Driver, 343  
rtcObjectInit  
    RTC Driver, 343  
rtcSetAlarm  
    RTC Driver, 345  
rtcSetCallback  
    RTC Driver, 346  
rtcSetTime  
    RTC Driver, 344  
rtcalarm\_t  
    RTC Driver, 343  
rtccb\_t  
    RTC Driver, 343  
rtcevent\_t  
    RTC Driver, 343  
rtcnt\_t  
    OSAL, 238  
rx\_buffer  
    I2SConfig, 552  
rbxbuf  
    USBOutEndpointState, 616  
rxchar\_cb  
    UARTConfig, 600  
rxcnt  
    USBOutEndpointState, 616  
rxend\_cb  
    UARTConfig, 600  
rxerr\_cb  
    UARTConfig, 600  
rfull\_event  
    CANDriver, 531

rxqueue  
     CANDriver, 531  
 rxsize  
     USBOutEndpointState, 616  
 rxstate  
     UARTDriver, 602  
  
 SD1  
     Serial Driver, 392  
 SD\_BREAK\_DETECTED  
     Serial Driver, 382  
 SD\_FRAMING\_ERROR  
     Serial Driver, 381  
 SD\_NOISE\_ERROR  
     Serial Driver, 382  
 SD\_OVERRUN\_ERROR  
     Serial Driver, 382  
 SD\_PARITY\_ERROR  
     Serial Driver, 381  
 SD\_READY  
     Serial Driver, 386  
 SD\_STOP  
     Serial Driver, 386  
 SD\_UNINIT  
     Serial Driver, 386  
 SDC Driver, 350  
     \_sdc\_driver\_methods, 355  
     \_sdc\_wait\_for\_transfer\_state, 364  
     detect\_bus\_clk, 361  
     mmc\_cmd6\_construct, 358  
     mmc\_detect\_bus\_clk, 360  
     mmc\_init, 356  
     mmc\_set\_bus\_width, 362  
     mmc\_switch\_t, 355  
     mode\_detect, 356  
     PLATFORM\_SDC\_USE\_SDC1, 355  
     SDC\_INIT\_RETRY, 353  
     SDC\_MMIC\_SUPPORT, 353  
     SDC\_NICE\_WAITING, 354  
     SDCD1, 378  
     SDCDriver, 355  
     sd\_switch\_function\_t, 355  
     sd\_switch\_t, 355  
     sdc\_cmd6\_check\_status, 359  
     sdc\_cmd6\_construct, 358  
     sdc\_cmd6\_extract\_info, 359  
     sdc\_detect\_bus\_clk, 359  
     sdc\_init, 357  
     sdc\_lld\_init, 373  
     sdc\_lld\_read, 376  
     sdc\_lld\_send\_cmd\_long\_crc, 376  
     sdc\_lld\_send\_cmd\_none, 375  
     sdc\_lld\_send\_cmd\_short, 375  
     sdc\_lld\_send\_cmd\_short\_crc, 375  
     sdc\_lld\_set\_bus\_mode, 374  
     sdc\_lld\_set\_data\_clk, 374  
     sdc\_lld\_start, 373  
     sdc\_lld\_start\_clk, 374  
     sdc\_lld\_stop, 373  
  
 sdc\_lld\_stop\_clk, 374  
 sdc\_lld\_sync, 377  
 sdc\_lld\_write, 377  
 sdc\_set\_bus\_width, 362  
 sdc\_vmt, 378  
 sdcConnect, 366  
 sdcDisconnect, 367  
 sdcErase, 372  
 sdcGetAndClearErrors, 370  
 sdcGetInfo, 372  
 sdcInit, 365  
 sdclsCardInserted, 354  
 sdclsWriteProtected, 354  
 sdcObjectInit, 365  
 sdcRead, 369  
 sdcStart, 365  
 sdcStop, 366  
 sdcSync, 371  
 sdcWrite, 370  
 sdcbusclk\_t, 356  
 sdcbusmode\_t, 356  
 sdcflags\_t, 355  
 sdcmode\_t, 355  
  
 SDC\_INIT\_RETRY  
     Configuration, 219  
     SDC Driver, 353  
  
 SDC\_MMIC\_SUPPORT  
     Configuration, 220  
     SDC Driver, 353  
  
 SDC\_NICE\_WAITING  
     Configuration, 220  
     SDC Driver, 354  
  
 SDCCConfig, 583  
     bus\_width, 584  
     scratchpad, 584  
  
 SDCD1  
     SDC Driver, 378  
  
 SDCDriver, 584  
     cardmode, 586  
     config, 585  
     errors, 586  
     rca, 586  
     SDC Driver, 355  
     vmt, 585  
  
 SDCDriverVMT, 586  
  
 SDU\_READY  
     Serial over USB Driver, 397  
  
 SDU\_STOP  
     Serial over USB Driver, 397  
  
 SDU\_UNINIT  
     Serial over USB Driver, 397  
  
 SERIAL\_BUFFERS\_SIZE  
     Configuration, 220  
     Serial Driver, 382  
  
 SERIAL\_DEFAULT\_BITRATE  
     Configuration, 220  
     Serial Driver, 382  
  
 SERIAL\_USB\_BUFFERS\_NUMBER

Configuration, 220  
Serial over USB Driver, 396  
**SERIAL\_USB\_BUFFERS\_SIZE**  
    Configuration, 220  
    Serial over USB Driver, 396  
**SID**  
    CANRxFrame, 534  
    CANTxFrame, 535  
**SPI Driver**, 405  
    \_spi\_isr\_code, 412  
    \_spi\_wakeup\_isr, 412  
    PLATFORM\_SPI\_USE\_SPI1, 413  
    **SPI\_ACTIVE**, 413  
    **SPI\_COMPLETE**, 413  
    **SPI\_READY**, 413  
    **SPI\_STOP**, 413  
    **SPI\_UNINIT**, 413  
    **SPI\_USE\_MUTUAL\_EXCLUSION**, 408  
    **SPI\_USE\_WAIT**, 408  
    **SPID1**, 428  
    **SPIDriver**, 413  
    spi\_lld\_exchange, 426  
    spi\_lld\_ignore, 426  
    spi\_lld\_init, 425  
    spi\_lld\_polled\_exchange, 428  
    spi\_lld\_receive, 427  
    spi\_lld\_select, 426  
    spi\_lld\_send, 427  
    spi\_lld\_start, 425  
    spi\_lld\_stop, 425  
    spi\_lld\_unselect, 426  
    spiAcquireBus, 424  
    spiExchange, 421  
    spilgnore, 420  
    spilinit, 413  
    spiObjectInit, 414  
    spiPolledExchange, 411  
    spiReceive, 423  
    spiReleaseBus, 424  
    spiSelect, 415  
    spiSelectl, 408  
    spiSend, 422  
    spiStart, 414  
    spiStartExchange, 418  
    spiStartExchangel, 409  
    spiStartIgnore, 417  
    spiStartIgnorel, 409  
    spiStartReceive, 420  
    spiStartReceivel, 410  
    spiStartSend, 419  
    spiStartSendl, 410  
    spiStop, 415  
    spiUnselect, 417  
    spiUnselectl, 408  
    spicallback\_t, 413  
    spistate\_t, 413  
**SPI\_ACTIVE**  
    SPI Driver, 413  
    **SPI\_COMPLETE**  
        SPI Driver, 413  
    **SPI\_READY**  
        SPI Driver, 413  
    **SPI\_STOP**  
        SPI Driver, 413  
    **SPI\_UNINIT**  
        SPI Driver, 413  
    **SPI\_USE\_WAIT**  
        Configuration, 220  
        SPI Driver, 408  
    **SPIConfig**, 596  
        end\_cb, 596  
**SPID1**  
    SPI Driver, 428  
**SPIDriver**, 597  
    config, 597  
    mutex, 598  
    SPI Driver, 413  
    state, 597  
    thread, 598  
**ST Driver**, 429  
    st\_lld\_get\_alarm, 433  
    st\_lld\_get\_counter, 432  
    st\_lld\_init, 432  
    st\_lld\_is\_alarm\_active, 433  
    st\_lld\_set\_alarm, 433  
    st\_lld\_start\_alarm, 433  
    st\_lld\_stop\_alarm, 433  
    stGetAlarm, 432  
    stGetCounter, 429  
    stInit, 430  
    stIsAlarmActive, 430  
    stSetAlarm, 431  
    stStartAlarm, 430  
    stStopAlarm, 431  
**samples**  
    ADCDriver, 515  
    DACDriver, 540  
**scratchpad**  
    SDCConfig, 584  
**sd\_lld\_init**  
    Serial Driver, 391  
**sd\_lld\_start**  
    Serial Driver, 392  
**sd\_lld\_stop**  
    Serial Driver, 392  
**sd\_switch\_function\_t**  
    SDC Driver, 355  
**sd\_switch\_t**  
    SDC Driver, 355  
**sdAsynchronousRead**  
    Serial Driver, 385  
**sdAsynchronousWrite**  
    Serial Driver, 384

sdGet  
    Serial Driver, 383  
sdGetTimeout  
    Serial Driver, 383  
sdGetWouldBlock  
    Serial Driver, 391  
sdIncomingData  
    Serial Driver, 388  
sdInit  
    Serial Driver, 386  
sdObjectInit  
    Serial Driver, 387  
sdPut  
    Serial Driver, 382  
sdPutTimeout  
    Serial Driver, 382  
sdPutWouldBlock  
    Serial Driver, 390  
sdRead  
    Serial Driver, 384  
sdReadTimeout  
    Serial Driver, 385  
sdRequestData  
    Serial Driver, 389  
sdStart  
    Serial Driver, 387  
sdStop  
    Serial Driver, 388  
sdWrite  
    Serial Driver, 383  
sdWriteTimeout  
    Serial Driver, 384  
sdc.c, 694  
sdc.h, 695  
sdc\_cmd6\_check\_status  
    SDC Driver, 359  
sdc\_cmd6\_construct  
    SDC Driver, 358  
sdc\_cmd6\_extract\_info  
    SDC Driver, 359  
sdc\_detect\_bus\_clk  
    SDC Driver, 359  
sdc\_init  
    SDC Driver, 357  
sdc\_lld.c, 697  
sdc\_lld.h, 698  
sdc\_lld\_init  
    SDC Driver, 373  
sdc\_lld\_read  
    SDC Driver, 376  
sdc\_lld\_send\_cmd\_long\_crc  
    SDC Driver, 376  
sdc\_lld\_send\_cmd\_none  
    SDC Driver, 375  
sdc\_lld\_send\_cmd\_short  
    SDC Driver, 375  
sdc\_lld\_send\_cmd\_short\_crc  
    SDC Driver, 375  
sdc\_lld\_set\_bus\_mode  
    SDC Driver, 374  
sdc\_lld\_set\_data\_clk  
    SDC Driver, 374  
sdc\_lld\_start  
    SDC Driver, 373  
sdc\_lld\_start\_clk  
    SDC Driver, 374  
sdc\_lld\_stop  
    SDC Driver, 373  
sdc\_lld\_stop\_clk  
    SDC Driver, 374  
sdc\_lld\_sync  
    SDC Driver, 377  
sdc\_lld\_write  
    SDC Driver, 377  
sdc\_set\_bus\_width  
    SDC Driver, 362  
sdc\_vmt  
    SDC Driver, 378  
sdcConnect  
    SDC Driver, 366  
sdcDisconnect  
    SDC Driver, 367  
sdcErase  
    SDC Driver, 372  
sdcGetAndClearErrors  
    SDC Driver, 370  
sdcGetInfo  
    SDC Driver, 372  
sdclInit  
    SDC Driver, 365  
sdclsCardInserted  
    SDC Driver, 354  
sdclsWriteProtected  
    SDC Driver, 354  
sdcObjectInit  
    SDC Driver, 365  
sdcRead  
    SDC Driver, 369  
sdcStart  
    SDC Driver, 365  
sdcStop  
    SDC Driver, 366  
sdcSync  
    SDC Driver, 371  
sdcWrite  
    SDC Driver, 370  
sdcbusclk\_t  
    SDC Driver, 356  
sdcbusmode\_t  
    SDC Driver, 356  
sdcflags\_t  
    SDC Driver, 355  
sdemode\_t  
    SDC Driver, 355  
sdstate\_t  
    Serial Driver, 386

sduConfigureHookI  
    Serial over USB Driver, 400  
sduDataReceived  
    Serial over USB Driver, 402  
sduDataTransmitted  
    Serial over USB Driver, 402  
sduDisconnectI  
    Serial over USB Driver, 399  
sduInit  
    Serial over USB Driver, 397  
sduInterruptTransmitted  
    Serial over USB Driver, 404  
sduObjectInit  
    Serial over USB Driver, 398  
sduRequestsHook  
    Serial over USB Driver, 401  
sduSOFHookI  
    Serial over USB Driver, 401  
sduStart  
    Serial over USB Driver, 398  
sduStop  
    Serial over USB Driver, 399  
sdustate\_t  
    Serial over USB Driver, 396  
send\_command\_R1  
    MMC over SPI Driver, 266  
send\_command\_R3  
    MMC over SPI Driver, 266  
send\_hdr  
    MMC over SPI Driver, 264  
Serial Driver, 379  
    \_serial\_driver\_data, 385  
    \_serial\_driver\_methods, 382  
    default\_config, 392  
    PLATFORM\_SERIAL\_USE\_USART1, 385  
SD1, 392  
SD\_BREAK\_DETECTED, 382  
SD\_FRAMING\_ERROR, 381  
SD\_NOISE\_ERROR, 382  
SD\_OVERRUN\_ERROR, 382  
SD\_PARITY\_ERROR, 381  
SD\_READY, 386  
SD\_STOP, 386  
SD\_UNINIT, 386  
SERIAL\_BUFFERS\_SIZE, 382  
SERIAL\_DEFAULT\_BITRATE, 382  
sd\_lld\_init, 391  
sd\_lld\_start, 392  
sd\_lld\_stop, 392  
sdAsynchronousRead, 385  
sdAsynchronousWrite, 384  
sdGet, 383  
sdGetTimeout, 383  
sdGetWouldBlock, 391  
sdIncomingDataI, 388  
sdInit, 386  
sdObjectInit, 387  
sdPut, 382  
sdPutTimeout, 382  
sdPutWouldBlock, 390  
sdRead, 384  
sdReadTimeout, 385  
sdRequestDataI, 389  
sdStart, 387  
sdStop, 388  
sdWrite, 383  
sdWriteTimeout, 384  
sdstate\_t, 386  
SerialDriver, 386  
Serial over USB Driver, 394  
    \_serial\_usb\_driver\_data, 396  
    \_serial\_usb\_driver\_methods, 396  
ibnotify, 397  
obnotify, 397  
SDU\_READY, 397  
SDU\_STOP, 397  
SDU\_UNINIT, 397  
SERIAL\_USB\_BUFFERS\_NUMBER, 396  
SERIAL\_USB\_BUFFERS\_SIZE, 396  
sduConfigureHookI, 400  
sduDataReceived, 402  
sduDataTransmitted, 402  
sduDisconnectI, 399  
sduInit, 397  
sduInterruptTransmitted, 404  
sduObjectInit, 398  
sduRequestsHook, 401  
sduSOFHookI, 401  
sduStart, 398  
sduStop, 399  
sdustate\_t, 396  
SerialUSBDriver, 396  
serial.c, 699  
serial.h, 700  
serial\_lld.c, 701  
serial\_lld.h, 702  
serial\_usb.c, 703  
serial\_usb.h, 703  
SerialConfig, 587  
    speed, 588  
SerialDriver, 588  
    Serial Driver, 386  
    vmt, 589  
SerialDriverVMT, 589  
SerialUSBConfig, 591  
    bulk\_in, 592  
    bulk\_out, 592  
    int\_in, 592  
    usbp, 592  
SerialUSBDriver, 592  
    Serial over USB Driver, 396  
    vmt, 594  
SerialUSBDriverVMT, 594  
set\_address  
    USB Driver, 481  
setup

USBDriver, 611  
 setup\_cb  
     USBEndpointConfig, 613  
 size  
     I2SConfig, 553  
     MACReceiveDescriptor, 564  
     MACTransmitDescriptor, 565  
 sleep\_event  
     CANDriver, 532  
 sof\_cb  
     USBConfig, 608  
 speed  
     SerialConfig, 588  
 spi.c, 705  
 spi.h, 706  
 spi\_lld.c, 707  
 spi\_lld.h, 708  
 spi\_lld\_exchange  
     SPI Driver, 426  
 spi\_lld\_ignore  
     SPI Driver, 426  
 spi\_lld\_init  
     SPI Driver, 425  
 spi\_lld\_polled\_exchange  
     SPI Driver, 428  
 spi\_lld\_receive  
     SPI Driver, 427  
 spi\_lld\_select  
     SPI Driver, 426  
 spi\_lld\_send  
     SPI Driver, 427  
 spi\_lld\_start  
     SPI Driver, 425  
 spi\_lld\_stop  
     SPI Driver, 425  
 spi\_lld\_unselect  
     SPI Driver, 426  
 spiAcquireBus  
     SPI Driver, 424  
 spiExchange  
     SPI Driver, 421  
 spilgnore  
     SPI Driver, 420  
 spiInit  
     SPI Driver, 413  
 spiObjectInit  
     SPI Driver, 414  
 spiPolledExchange  
     SPI Driver, 411  
 spiReceive  
     SPI Driver, 423  
 spiReleaseBus  
     SPI Driver, 424  
 spiSelect  
     SPI Driver, 415  
 spiSelectl  
     SPI Driver, 408  
 spiSend  
     SPI Driver, 422  
 spiStart  
     SPI Driver, 414  
 spiStartExchange  
     SPI Driver, 418  
 spiStartExchangel  
     SPI Driver, 409  
 spiStartIgnore  
     SPI Driver, 417  
 spiStartIgnoreL  
     SPI Driver, 409  
 spiStartReceive  
     SPI Driver, 420  
 spiStartReceiveL  
     SPI Driver, 410  
 spiStartSend  
     SPI Driver, 419  
 spiStartSendL  
     SPI Driver, 410  
 spiStop  
     SPI Driver, 415  
 spiUnselect  
     SPI Driver, 417  
 spiUnselectl  
     SPI Driver, 408  
 spicallback\_t  
     SPI Driver, 413  
 spip  
     MMCConfig, 566  
 spistate\_t  
     SPI Driver, 413  
 st.c, 709  
 st.h, 710  
 st\_lld.c, 710  
 st\_lld.h, 711  
 st\_lld\_get\_alarm  
     ST Driver, 433  
 st\_lld\_get\_counter  
     ST Driver, 432  
 st\_lld\_init  
     ST Driver, 432  
 st\_lld\_is\_alarm\_active  
     ST Driver, 433  
 st\_lld\_set\_alarm  
     ST Driver, 433  
 st\_lld\_start\_alarm  
     ST Driver, 433  
 st\_lld\_stop\_alarm  
     ST Driver, 433  
 stGetAlarm  
     ST Driver, 432  
 stGetCounter  
     ST Driver, 429  
 stInit  
     ST Driver, 430  
 stIsAlarmActive  
     ST Driver, 430  
 stSetAlarm

ST Driver, 431  
stStartAlarm  
    ST Driver, 430  
stStopAlarm  
    ST Driver, 431  
state  
    ADCDriver, 515  
    CANDriver, 531  
    DACDriver, 540  
    EXTDriver, 544  
    GPTDriver, 549  
    I2CDriver, 551  
    I2SDriver, 554  
    ICUDriver, 556  
    MACDriver, 563  
    PWMDriver, 578  
    SPIDriver, 597  
    UARTDriver, 602  
    USBDriver, 610  
    WDGDriver, 617  
status  
    USBDriver, 611  
streamGet  
    Abstract Streams, 155  
streamPut  
    Abstract Streams, 154  
streamRead  
    Abstract Streams, 154  
streamWrite  
    Abstract Streams, 154  
Support Code, 226  
sync  
    MMC over SPI Driver, 268  
syssts\_t  
    OSAL, 237  
system\_time\_t  
    OSAL, 238

TIME  
    CANRxFrame, 533

tdqueue  
    MACDriver, 563

thread  
    ADCDriver, 515  
    DACDriver, 540  
    SPIDriver, 598  
    USBInEndpointState, 615  
    USBOutEndpointState, 616

thread\_reference\_t  
    OSAL, 238

threadrx  
    UARTDriver, 602

threads\_queue\_t, 598

threadtx  
    UARTDriver, 602

top  
    io\_buffers\_queue, 558

transmitting  
    USBDriver, 610

tx\_buffer  
    I2SConfig, 552

txbuf  
    USBInEndpointState, 615

txcnt  
    USBInEndpointState, 615

txempty\_event  
    CANDriver, 531

txend1\_cb  
    UARTConfig, 599

txend2\_cb  
    UARTConfig, 600

txqueue  
    CANDriver, 531

txsize  
    USBInEndpointState, 615

txstate  
    UARTDriver, 602

UART Driver, 435  
    \_uart\_rx\_complete\_isr\_code, 443  
    \_uart\_rx\_error\_isr\_code, 443  
    \_uart\_rx\_idle\_code, 444  
    \_uart\_tx1\_isr\_code, 442  
    \_uart\_tx2\_isr\_code, 442  
    \_uart\_wakeup\_rx\_complete\_isr, 441  
    \_uart\_wakeup\_rx\_error\_isr, 441  
    \_uart\_wakeup\_tx1\_isr, 440  
    \_uart\_wakeup\_tx2\_isr, 440  
    PLATFORM\_UART\_USE\_UART1, 444  
    UART\_BREAK\_DETECTED, 440  
    UART\_FRAMING\_ERROR, 439  
    UART\_NO\_ERROR, 439  
    UART\_NOISE\_ERROR, 440  
    UART\_OVERRUN\_ERROR, 439  
    UART\_PARITY\_ERROR, 439  
    UART\_READY, 445  
    UART\_RX\_ACTIVE, 446  
    UART\_RX\_COMPLETE, 446  
    UART\_RX\_IDLE, 446  
    UART\_STOP, 445  
    UART\_TX\_ACTIVE, 445  
    UART\_TX\_COMPLETE, 446  
    UART\_TX\_IDLE, 445  
    UART\_UNINIT, 445  
    UART\_USE\_MUTUAL\_EXCLUSION, 440  
    UART\_USE\_WAIT, 440  
    UARTD1, 460  
    UARTDriver, 445  
    uart\_lld\_init, 457  
    uart\_lld\_start, 458  
    uart\_lld\_start\_receive, 459  
    uart\_lld\_start\_send, 458  
    uart\_lld\_stop, 458  
    uart\_lld\_stop\_receive, 459  
    uart\_lld\_stop\_send, 458  
    uartAcquireBus, 456  
    uartInit, 446  
    uartObjectInit, 446

uartReceiveTimeout, 455  
 uartReleaseBus, 457  
 uartSendFullTimeout, 454  
 uartSendTimeout, 453  
 uartStart, 447  
 uartStartReceive, 450  
 uartStartReceivel, 451  
 uartStartSend, 448  
 uartStartSendl, 448  
 uartStop, 447  
 uartStopReceive, 452  
 uartStopReceivel, 453  
 uartStopSend, 449  
 uartStopSendl, 450  
 uartccb\_t, 445  
 uartecb\_t, 445  
 uartflags\_t, 445  
 uartrxstate\_t, 446  
 uartstate\_t, 445  
 uarttxstate\_t, 445  
**UART\_BREAK\_DETECTED**  
 UART Driver, 440  
**UART\_FRAMING\_ERROR**  
 UART Driver, 439  
**UART\_NO\_ERROR**  
 UART Driver, 439  
**UART\_NOISE\_ERROR**  
 UART Driver, 440  
**UART\_OVERRUN\_ERROR**  
 UART Driver, 439  
**UART\_PARITY\_ERROR**  
 UART Driver, 439  
**UART\_READY**  
 UART Driver, 445  
**UART\_RX\_ACTIVE**  
 UART Driver, 446  
**UART\_RX\_COMPLETE**  
 UART Driver, 446  
**UART\_RX\_IDLE**  
 UART Driver, 446  
**UART\_STOP**  
 UART Driver, 445  
**UART\_TX\_ACTIVE**  
 UART Driver, 445  
**UART\_TX\_COMPLETE**  
 UART Driver, 446  
**UART\_TX\_IDLE**  
 UART Driver, 445  
**UART\_UNINIT**  
 UART Driver, 445  
**UART\_USE\_MUTUAL\_EXCLUSION**  
 Configuration, 221  
 UART Driver, 440  
**UART\_USE\_WAIT**  
 Configuration, 221  
 UART Driver, 440  
**UARTConfig**, 598  
 rxchar\_cb, 600  
 rxend\_cb, 600  
 rxerr\_cb, 600  
 txend1\_cb, 599  
 txend2\_cb, 600  
**UARTD1**  
 UART Driver, 460  
**UARTDriver**, 600  
 config, 602  
 early, 602  
 mutex, 602  
 rxstate, 602  
 state, 602  
 threadrx, 602  
 threadtx, 602  
 txstate, 602  
 UART Driver, 445  
**USB\_CDC\_Header**, 503  
**USB\_Driver**, 461  
 \_usb\_ep0in, 493  
 \_usb\_ep0out, 494  
 \_usb\_ep0setup, 492  
 \_usb\_isr\_invoke\_event\_cb, 474  
 \_usb\_isr\_invoke\_in\_cb, 476  
 \_usb\_isr\_invoke\_out\_cb, 476  
 \_usb\_isr\_invoke\_setup\_cb, 476  
 \_usb\_isr\_invoke\_sof\_cb, 474  
 \_usb\_reset, 490  
 \_usb\_suspend, 491  
 \_usb\_wakeup, 491  
 default\_handler, 482  
 EP\_STATUS\_ACTIVE, 480  
 EP\_STATUS\_DISABLED, 480  
 EP\_STATUS\_STALLED, 480  
 ep0\_state, 502  
 ep0config, 502  
 in, 502  
 out, 502  
 PLATFORM\_USB\_USE\_USB1, 477  
 set\_address, 481  
 USB\_ACTIVE, 480  
 USB\_DESC\_BCD, 468  
 USB\_DESC\_BYTE, 468  
 USB\_DESC\_CONFIGURATION, 469  
 USB\_DESC\_CONFIGURATION\_SIZE, 469  
 USB\_DESC\_DEVICE, 468  
 USB\_DESC\_ENDPOINT, 470  
 USB\_DESC\_ENDPOINT\_SIZE, 470  
 USB\_DESC\_INDEX, 468  
 USB\_DESC\_INTERFACE, 469  
 USB\_DESC\_INTERFACE\_ASSOCIATION, 470  
 USB\_DESC\_INTERFACE\_ASSOCIATION\_SIZE, 470  
 USB\_DESC\_INTERFACE\_SIZE, 469  
 USB\_DESC\_WORD, 468  
 USB\_EP0\_ERROR, 480  
 USB\_EP0\_RX, 480  
 USB\_EP0\_SENDING\_STS, 480

USB\_EP0\_STATUS\_STAGE, 477  
USB\_EP0\_TX, 480  
USB\_EP0\_WAITING\_SETUP, 480  
USB\_EP0\_WAITING\_STS, 480  
USB\_EP0\_WAITING\_TX0, 480  
USB\_EP\_MODE\_TYPE, 470  
USB\_EP\_MODE\_TYPE\_BULK, 471  
USB\_EP\_MODE\_TYPE\_CTRL, 470  
USB\_EP\_MODE\_TYPE\_INTR, 471  
USB\_EP\_MODE\_TYPE\_ISOC, 471  
USB\_EVENT\_ADDRESS, 480  
USB\_EVENT\_CONFIGURED, 480  
USB\_EVENT\_RESET, 480  
USB\_EVENT\_STALLED, 480  
USB\_EVENT\_SUSPEND, 480  
USB\_EVENT\_WAKEUP, 480  
USB\_MAX\_ENDPOINTS, 477  
USB\_READY, 480  
USB\_SELECTED, 480  
USB\_SET\_ADDRESS\_ACK\_HANDLING, 477  
USB\_SET\_ADDRESS\_MODE, 477  
USB\_STOP, 480  
USB\_SUSPENDED, 480  
USB\_UNINIT, 480  
USB\_USE\_WAIT, 471  
USBD1, 502  
USBDriver, 478  
usb\_lld\_clear\_in, 501  
usb\_lld\_clear\_out, 501  
usb\_lld\_connect\_bus, 478  
usb\_lld\_disable\_endpoints, 497  
usb\_lld\_disconnect\_bus, 478  
usb\_lld\_get\_frame\_number, 477  
usb\_lld\_get\_status\_in, 499  
usb\_lld\_get\_status\_out, 499  
usb\_lld\_get\_transaction\_size, 478  
usb\_lld\_init, 495  
usb\_lld\_init\_endpoint, 497  
usb\_lld\_prepare\_receive, 500  
usb\_lld\_prepare\_transmit, 500  
usb\_lld\_read\_setup, 499  
usb\_lld\_reset, 497  
usb\_lld\_set\_address, 497  
usb\_lld\_stall\_in, 501  
usb\_lld\_stall\_out, 501  
usb\_lld\_start, 495  
usb\_lld\_start\_in, 500  
usb\_lld\_start\_out, 500  
usb\_lld\_stop, 495  
usbConnectBus, 471  
usbDisableEndpointsI, 485  
usbDisconnectBus, 471  
usbGetDriverStatI, 471  
usbGetFrameNumberX, 472  
usbGetReceiveStatusI, 472  
usbGetReceiveTransactionSizeX, 473  
usbGetTransmitStatusI, 472  
usblInit, 483  
usbInitEndpointI, 485  
usbObjectInit, 483  
usbReadSetup, 474  
usbReceive, 487  
usbSetupTransfer, 473  
usbStallReceiveI, 489  
usbStallTransmitI, 490  
usbStart, 484  
usbStartReceiveI, 486  
usbStartTransmitI, 487  
usbStop, 484  
usbTransmit, 488  
usbcallback\_t, 479  
usbep0state\_t, 480  
usbep\_t, 478  
usbepcallback\_t, 479  
usbepstatus\_t, 480  
usbevent\_t, 480  
usbeventcb\_t, 479  
usbgetdescriptor\_t, 479  
usbreqhandler\_t, 479  
usbstate\_t, 480  
USB\_ACTIVE  
    USB Driver, 480  
USB\_DESC\_BCD  
    USB Driver, 468  
USB\_DESC\_BYTE  
    USB Driver, 468  
USB\_DESC\_CONFIGURATION  
    USB Driver, 469  
USB\_DESC\_CONFIGURATION\_SIZE  
    USB Driver, 469  
USB\_DESC\_DEVICE  
    USB Driver, 468  
USB\_DESC\_ENDPOINT  
    USB Driver, 470  
USB\_DESC\_ENDPOINT\_SIZE  
    USB Driver, 470  
USB\_DESC\_INDEX  
    USB Driver, 468  
USB\_DESC\_INTERFACE  
    USB Driver, 469  
USB\_DESC\_INTERFACE\_ASSOCIATION  
    USB Driver, 470  
USB\_DESC\_INTERFACE\_ASSOCIATION\_SIZE  
    USB Driver, 470  
USB\_DESC\_INTERFACE\_SIZE  
    USB Driver, 469  
USB\_DESC\_WORD  
    USB Driver, 468  
USB\_EP0\_ERROR  
    USB Driver, 480  
USB\_EP0\_RX  
    USB Driver, 480  
USB\_EP0\_SENDING\_STS  
    USB Driver, 480  
USB\_EP0\_STATUS\_STAGE  
    USB Driver, 477

USB\_EP0\_TX  
     USB Driver, 480

USB\_EP0\_WAITING\_SETUP  
     USB Driver, 480

USB\_EP0\_WAITING\_STS  
     USB Driver, 480

USB\_EP0\_WAITING\_TX0  
     USB Driver, 480

USB\_EP\_MODE\_TYPE  
     USB Driver, 470

USB\_EP\_MODE\_TYPE\_BULK  
     USB Driver, 471

USB\_EP\_MODE\_TYPE\_CTRL  
     USB Driver, 470

USB\_EP\_MODE\_TYPE\_INTR  
     USB Driver, 471

USB\_EP\_MODE\_TYPE\_ISOC  
     USB Driver, 471

USB\_EVENT\_ADDRESS  
     USB Driver, 480

USB\_EVENT\_CONFIGURED  
     USB Driver, 480

USB\_EVENT\_RESET  
     USB Driver, 480

USB\_EVENT\_STALLED  
     USB Driver, 480

USB\_EVENT\_SUSPEND  
     USB Driver, 480

USB\_EVENT\_WAKEUP  
     USB Driver, 480

USB\_MAX\_ENDPOINTS  
     USB Driver, 477

USB\_READY  
     USB Driver, 480

USB\_SELECTED  
     USB Driver, 480

USB\_SET\_ADDRESS\_ACK\_HANDLING  
     USB Driver, 477

USB\_SET\_ADDRESS\_MODE  
     USB Driver, 477

USB\_STOP  
     USB Driver, 480

USB\_SUSPENDED  
     USB Driver, 480

USB\_UNINIT  
     USB Driver, 480

USB\_USE\_WAIT  
     Configuration, 221  
     USB Driver, 471

USBConfig, 606  
     event\_cb, 608  
     get\_descriptor\_cb, 608  
     requests\_hook\_cb, 608  
     sof\_cb, 608

USBD1  
     USB Driver, 502

USBDescriptor, 608  
     ud\_size, 609

    ud\_string, 609

USBDriver, 609  
     address, 611  
     config, 610  
     configuration, 611  
     ep0endcb, 611  
     ep0n, 611  
     ep0next, 611  
     ep0state, 611  
     epc, 610  
     in\_params, 611  
     out\_params, 611  
     receiving, 610  
     setup, 611  
     state, 610  
     status, 611  
     transmitting, 610  
     USB Driver, 478

USBEndpointConfig, 612  
     ep\_mode, 613  
     in\_cb, 613  
     in\_maxsize, 614  
     in\_state, 614  
     out\_cb, 613  
     out\_maxsize, 614  
     out\_state, 614  
     setup\_cb, 613

USBInEndpointState, 614  
     thread, 615  
     txbuf, 615  
     txcnt, 615  
     txsize, 615

USBOutEndpointState, 615  
     rdbuf, 616  
     rcnt, 616  
     rxsize, 616  
     thread, 616

uart.c, 711  
 uart.h, 712  
 uart\_lld.c, 714  
 uart\_lld.h, 715  
 uart\_lld\_init  
     UART Driver, 457

uart\_lld\_start  
     UART Driver, 458

uart\_lld\_start\_receive  
     UART Driver, 459

uart\_lld\_start\_send  
     UART Driver, 458

uart\_lld\_stop  
     UART Driver, 458

uart\_lld\_start\_receive  
     UART Driver, 459

uart\_lld\_stop\_send  
     UART Driver, 458

uartAcquireBus  
     UART Driver, 456

uartInit

UART Driver, 446  
uartObjectInit  
    UART Driver, 446  
uartReceiveTimeout  
    UART Driver, 455  
uartReleaseBus  
    UART Driver, 457  
uartSendFullTimeout  
    UART Driver, 454  
uartSendTimeout  
    UART Driver, 453  
uartStart  
    UART Driver, 447  
uartStartReceive  
    UART Driver, 450  
uartStartReceive1  
    UART Driver, 451  
uartStartSend  
    UART Driver, 448  
uartStartSend1  
    UART Driver, 448  
uartStop  
    UART Driver, 447  
uartStopReceive  
    UART Driver, 452  
uartStopReceive1  
    UART Driver, 453  
uartStopSend  
    UART Driver, 449  
uartStopSend1  
    UART Driver, 450  
uart tcb\_t  
    UART Driver, 445  
uartccb\_t  
    UART Driver, 445  
uartecb\_t  
    UART Driver, 445  
uartflags\_t  
    UART Driver, 445  
uartrxstate\_t  
    UART Driver, 446  
uartstate\_t  
    UART Driver, 445  
uartxstate\_t  
    UART Driver, 445  
ud\_size  
    USBDescriptor, 609  
ud\_string  
    USBDescriptor, 609  
unpacked\_mmc\_cid\_t, 602  
unpacked\_mmc\_csd\_t, 603  
unpacked\_sdc\_cid\_t, 604  
unpacked\_sdc\_csd\_10\_t, 604  
unpacked\_sdc\_csd\_20\_t, 605  
usb.c, 716  
usb.h, 717  
usb\_cdc.h, 720  
usb\_lld.c, 722  
    in, 723  
    out, 723  
usb\_lld.h, 723  
usb\_lld\_clear\_in  
    USB Driver, 501  
usb\_lld\_clear\_out  
    USB Driver, 501  
usb\_lld\_connect\_bus  
    USB Driver, 478  
usb\_lld\_disable\_endpoints  
    USB Driver, 497  
usb\_lld\_disconnect\_bus  
    USB Driver, 478  
usb\_lld\_get\_frame\_number  
    USB Driver, 477  
usb\_lld\_get\_status\_in  
    USB Driver, 499  
usb\_lld\_get\_status\_out  
    USB Driver, 499  
usb\_lld\_get\_transaction\_size  
    USB Driver, 478  
usb\_lld\_init  
    USB Driver, 495  
usb\_lld\_init\_endpoint  
    USB Driver, 497  
usb\_lld\_prepare\_receive  
    USB Driver, 500  
usb\_lld\_prepare\_transmit  
    USB Driver, 500  
usb\_lld\_read\_setup  
    USB Driver, 499  
usb\_lld\_reset  
    USB Driver, 497  
usb\_lld\_set\_address  
    USB Driver, 497  
usb\_lld\_stall\_in  
    USB Driver, 501  
usb\_lld\_stall\_out  
    USB Driver, 501  
usb\_lld\_start  
    USB Driver, 495  
usb\_lld\_start\_in  
    USB Driver, 500  
usb\_lld\_start\_out  
    USB Driver, 500  
usb\_lld\_stop  
    USB Driver, 495  
usbConnectBus  
    USB Driver, 471  
usbDisableEndpoints1  
    USB Driver, 485  
usbDisconnectBus  
    USB Driver, 471  
usbGetDriverState1  
    USB Driver, 471  
usbGetFrameNumberX  
    USB Driver, 472  
usbGetReceiveStatus1

USB Driver, 472  
**usbGetReceiveTransactionSizeX**  
  USB Driver, 473  
**usbGetTransmitStatus!**  
  USB Driver, 472  
**usbInit**  
  USB Driver, 483  
**usbInitEndpoint!**  
  USB Driver, 485  
**usbObjectInit**  
  USB Driver, 483  
**usbReadSetup**  
  USB Driver, 474  
**usbReceive**  
  USB Driver, 487  
**usbSetupTransfer**  
  USB Driver, 473  
**usbStallReceive!**  
  USB Driver, 489  
**usbStallTransmit!**  
  USB Driver, 490  
**usbStart**  
  USB Driver, 484  
**usbStartReceive!**  
  USB Driver, 486  
**usbStartTransmit!**  
  USB Driver, 487  
**usbStop**  
  USB Driver, 484  
**usbTransmit**  
  USB Driver, 488  
**usbcallback\_t**  
  USB Driver, 479  
**usbep0state\_t**  
  USB Driver, 480  
**usbep\_t**  
  USB Driver, 478  
**usbepcallback\_t**  
  USB Driver, 479  
**usbepstatus\_t**  
  USB Driver, 480  
**usbevent\_t**  
  USB Driver, 480  
**usbeventcb\_t**  
  USB Driver, 479  
**usbgetdescriptor\_t**  
  USB Driver, 479  
**usbp**  
  SerialUSBConfig, 592  
**usbreqhandler\_t**  
  USB Driver, 479  
**usbstate\_t**  
  USB Driver, 480

vmt  
  BaseAsynchronousChannel, 517  
  BaseBlockDevice, 521  
  BaseChannel, 524  
  BaseSequentialStream, 527

FileStream, 546  
**MMCDriver**, 568  
**MMCSDBlockDevice**, 572  
**RTCDriver**, 582  
**SDCDriver**, 585  
**SerialDriver**, 589  
**SerialUSBDriver**, 594

**WDG Driver**, 505  
  PLATFORM\_WDG\_USE\_WDG1, 506  
**WDG\_READY**, 506  
**WDG\_STOP**, 506  
**WDG\_UNINIT**, 506  
**WDGDriver**, 506  
**wdg\_lld\_init**, 509  
**wdg\_lld\_reset**, 510  
**wdg\_lld\_start**, 510  
**wdg\_lld\_stop**, 510  
**wdgInit**, 507  
**wdgReset**, 509  
**wdgResetl**, 506  
**wdgStart**, 507  
**wdgStop**, 507  
**wdgstate\_t**, 506

**WDG\_READY**  
  WDG Driver, 506  
**WDG\_STOP**  
  WDG Driver, 506  
**WDG\_UNINIT**  
  WDG Driver, 506  
**WDGConfig**, 616  
**WDGDriver**, 617  
  config, 617  
  state, 617  
  WDG Driver, 506

**wait**  
  MMC over SPI Driver, 263

**waiting**  
  io\_buffers\_queue, 558

**wakeup\_event**  
  CANDriver, 532

**wdg.c**, 725  
**wdg.h**, 725  
**wdg\_lld.c**, 726  
**wdg\_lld.h**, 726  
**wdg\_lld\_init**  
  WDG Driver, 509  
**wdg\_lld\_reset**  
  WDG Driver, 510  
**wdg\_lld\_start**  
  WDG Driver, 510  
**wdg\_lld\_stop**  
  WDG Driver, 510  
**wdgInit**  
  WDG Driver, 507  
**wdgReset**  
  WDG Driver, 509  
**wdgResetl**  
  WDG Driver, 506

wdgStart  
    WDG Driver, [507](#)

wdgStop  
    WDG Driver, [507](#)

wdgstate\_t  
    WDG Driver, [506](#)

width\_cb  
    ICUConfig, [555](#)

year  
    RTCDateTime, [580](#)